Proyecto Fin de Carrera

Ingeniería de Telecomunicación

# Detección de objetos en imágenes utilizando técnicas de aprendizaje profundo (Deep-Learning)

Autor: Daniel Estévez Trigo

Tutor: Rubén Martín Clemente

**Dpto. Teoría de la Señal y Comunicaciones**
**Escuela Técnica Superior de Ingeniería**
**Universidad de Sevilla**

Sevilla, 2021

Proyecto Fin de Grado

Grado en Ingeniería de las Tecnologías de la Telecomunicación

# Detección de objetos en imágenes utilizando técnicas de aprendizaje profundo (Deep Learning)


Autor:

Daniel Estévez Trigo

Tutor:

Rubén Martín Clemente

Profesor titular

**Dpto. de Teoría de la Señal y Comunicaciones**

**Escuela Técnica Superior de Ingeniería**

**Universidad de Sevilla**



**Sevilla, 2021**

Proyecto Fin de Carrera: Detección de objetos en imágenes utilizando técnicas de aprendizaje profundo (Deep-Learning)

Autor: Daniel Estévez Trigo
Tutor: Rubén Martín Clemente

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2021

El secretario del Tribunal

# Agradecimientos

A mi familia, mi padre, mi madre y mi hermano, el pilar más importante y sin los cuales no podría haber conseguido llegar a dónde hoy me encuentro. Gracias por su apoyo anímico, económico y por ser una figura referente en mi vida. Les debo mis éxitos, presentes y futuros.

A mis amigos de la infancia, que me han acompañado desde que comencé mi aventura en secundaria y así hasta el día de hoy, con los que sé que siempre puedo y podré contar. Siempre hemos logrado mantener la amistad a pesar de la distancia.

A mis compañeros de carrera, sois parte de mi familia, habéis sido la guía para no perder de vista la luz que se ve al final del túnel. Gracias por estar en las duras y en las maduras, todos juntos hemos pasado por pruebas difíciles y hemos aprendido a buscar soluciones y trabajar en equipo.

A Rocío, por estar siempre ahí, por haberme soportado en el tramo final de mi aventura, por haber sido mi apoyo en los buenos y sobre todo en los malos momentos, gracias.

Finalmente, a todos aquellos profesores que han impartido su docencia con pasión y emoción a su alumnado, por haber demostrado entrega y vocación por nuestro aprendizaje, especialmente a todos aquellos que han sido capaces de llevarnos al nivel que existe más allá de los simples libros. Gracias a Rubén, mi tutor, por haberme dado las pautas necesarias para llevar a cabo este proyecto, por haberme avivado la ilusión y el interés por las aplicaciones de la ingeniería.

*Daniel Estévez Trigo*

*Sevilla, septiembre de 2021*

# Resumen

En este proyecto se tratará de procesar y analizar diferentes retransmisiones deportivas utilizando la herramienta ImageAI. Se explica brevemente cómo se realiza de forma tradicional el procesamiento de imágenes y vídeos. Posteriormente, a través de la herramienta anteriormente mencionada trataremos de obtener la mayor cantidad de información posible a través de la automatización.

El objetivo es obtener un vídeo respuesta al código utilizado, detectando el movimiento de los jugadores y los objetos empleados que puede ser utilizado para un sinfín de aplicaciones.

# Índice

## Contenido

# Índice de Figuras

# Índice de Tablas

# Capítulo primero: Introducción

En este primer capítulo se expondrá de forma general las principales materias en las que se basará este proyecto, detallando su uso, el contexto en el que aplicamos las técnicas y el posible alcance que tienen a la hora de su implementación actual. Se delimitará el objetivo concreto del proyecto, exponiendo y explicando los resultados que se esperan del mismo. Finalmente, se mencionarán algunos de los casos en los que podremos aplicar estas técnicas en la actualidad.

## 1.1 Historia, evolución y aplicaciones de la visión por ordenador.

La visión por ordenador, también conocida como visión artificial o visión técnica es un campo interdisciplinario que incluye métodos para adquirir, procesar, analizar y comprender las imágenes del mundo real con el fin de producir información numérica o simbólica para que puedan ser tratados por un ordenador. De la misma manera en la que nosotros, los humanos usamos nuestros ojos y cerebro para comprender el entorno que nos rodea, la visión artificial trata de reproducir el mismo efecto de forma artificial y automatizada utilizando un ordenador (Wikipedia, la enciclopedia libre, 2021). Esta comprensión se consigue gracias a distintos campos como la geometría, la estadística o la física entre otros. La adquisición de los datos se consigue por varios medios como secuencias de imágenes, obtenidas desde varias cámaras de vídeo en diferentes puntos del espacio o datos multidimensionales desde un escáner. La evolución y aplicaciones de la visión artificial han estado estrechamente relacionadas con el desarrollo de las cámaras fotográficas y la obtención de imágenes desde sus orígenes.

El subdominio de la visión por ordenador incluye reconstrucción de escenas, detección de eventos, seguimiento de video, reconocimiento de objetos, estimación de pose 3D, aprendizaje, indexación, estimación de movimiento y restauración de imágenes.

A fines de la década de 1960, la visión por ordenador comenzó en universidades que eran pioneras en inteligencia artificial. Su principal tarea era imitar el sistema visual humano, como punto de partida para dotar a los robots de un comportamiento inteligente. En 1966, mediante un proyecto se trató de conectar una cámara a un ordenador y hacer que describiera todo lo que percibía visualmente.

Uno de los primeros investigadores de los que se tiene constancia en estudiar esta disciplina es Larry Roberts (Zdziaarski, 2018), quien es considerado como uno de los padres de internet. El objetivo de su investigación fue el de extraer información 3D a partir de imágenes 2D, de forma que pudiera ver distintas perspectivas de un objeto. Esto lo consiguió captando diferentes

imágenes de un sólido desde diferentes puntos de vista, y, procesándolas posteriormente para detectar bordes según cambios bruscos en la intensidad de grises.

La historia de la visión artificial marca su hito en la década de los 80 con el desarrollo de la ingeniería informática y la creación de procesadores más sofisticados y rápidos, dando lugar a microprocesadores capaces de captar, procesar y reproducir imágenes tomadas por una cámara a la que podían estar conectada de forma remota.

Desde entonces, esta rama ha seguido evolucionando casi con forma exponencial, debido a los avances de la tecnología ya que cada vez los ordenadores tienen mayor potencia y son más accesibles y para tanto en la captación de imágenes, consiguiendo cada vez mayor calidad y cantidad de información, como en el postprocesado, contando con equipo de una gran capacidad de computación. En la visión por ordenador se ha tratado de aprender de la propia naturaleza, en cómo ven los seres humanos, y posteriormente tratar de reproducirlo con la tecnología a nuestro alcance. El ojo es el órgano que dota del sentido de la vista, sobre el cual incide la luz provocando una serie de impulsos nerviosos en la retina. Esto es gracias a los foto-receptores, los conos y los bastones. Los conos son células sensibles a los colores, de forma que existen tres tipos de conos, cada uno sensible a una longitud de onda distinta, las correspondientes a los colores rojo, verde y azul. Mientras que, los bastones serán células sensibles a la intensidad de luz, es decir el brillo. Los responsables de emular este funcionamiento en el ámbito de la visión artificial son las cámaras y sensores ópticos, de forma que se dispone de una serie de fotodiodos dispuestos en matriz, cada uno capaz de captar la intensidad para los colores rojo, verde o azul, y posteriormente almacenando una carga correspondiente a los fotones incidentes. Esta imagen es posteriormente digitalizada y procesada.

Como se observa, la visión por ordenador tiene gran cantidad de aplicaciones en distintos ámbitos, como puede ser la edición y filtrado para el ámbito fotográfico, la capacidad de dotar de percepción a un autómata para que pueda tomar ciertas decisiones, manipulación de objetos en la industria, realizar un diagnóstico por medio de una imagen muy útil en el campo de la medicina como son los rayos X o TAC, tomar y recomponer imágenes tomadas por un satélite, vigilancia automática en el ámbito de la seguridad o capacidad de detectar vehículos y leer matrículas para el control del tráfico. Su misión principal en todos ellos es facilitar la vida del ser humano y, sobre todo, extraer la mayor cantidad de información de las imágenes de cara a poder automatizar al máximo nivel posible distintos procesos. En todos estos ámbitos, y otros más que no se han nombrado, la visión artificial está jugando un importante papel, siendo una de las principales ramas a investigar para los futuros avances tecnológicos. Uno de los principales relacionados tenemos la inteligencia artificial, ya que, por un lado, la visión ayuda a la extracción de características con los que formar los datasets con los que entrenar los distintos algoritmos y redes neuronales, y por otro lado la visión se ayuda de la inteligencia artificial para la detección automática de ciertos elementos o patrones en una imagen, como por ejemplo el reconocimiento facial. Otro campo en el cual también se basa de forma directa la visión artificial es el de procesamiento de señales ya que, gran parte de los métodos usados para procesar señales de una variable, se extienden de manera natural a señales de dos variables o multivariable que forman la visión.

Por otro lado, se tiene la optoelectrónica, de la que se depende directamente ya que es la encargada de fabricar los distintos sensores con los que se captan las imágenes, de forma que ayuda con la investigación de distintos tipos de sensores que se ajusten a las características concretas que se deseen para determinada aplicación.

también hay que destacar la aportación de la ciencia de computación, ya que el procesamiento de las imágenes conlleva un gran costo computacional, teniendo que operar sobre miles de píxeles por frame, y muchas veces, con la necesidad de procesar dichos frames a tiempo real, cosa que sería impensable sin los equipos actuales.

Como conclusión hay que puntualizar que la visión artificial es un campo que se encuentra en el apogeo de su crecimiento y que es una tecnología que sin duda es muy utilizada en el día a día en una gran variedad de aplicaciones.

## 1.2 Objetivos.

En el capítulo anterior se expusieron las bases de la visión artificial y los distintos ámbitos en los que esta puede actuar, a continuación, se procede a explicar la función que desempeña en este proyecto. Como se ha visto en el apartado anterior, la visión por ordenador tiene infinidad de aplicaciones, muchas de ellas orientadas a poder automatizar una tarea para la que antes se necesitaba de la dedicación de una o más personas. El programa que se va a desarrollar se centrará en la detección y seguimiento automático de los jugadores en el terreno de juego a partir de las imágenes que se obtienen con las diferentes cámaras ubicadas en el recinto. El objetivo será el de recolectar datos y parámetros de interés para poder elaborar ciertas estadísticas, como puede ser distancia total recorrida por cada jugador, mapas de calor que indiquen las zonas más transitadas o porcentajes de posesión del balón entre una gran cantidad de estadísticas disponibles que pueden ser automatizadas, y obtenerlas a tiempo real, lo cual es muy interesante tanto por el lado del espectador como por la parte técnica, ambas ofrecen información de interés. También se ofrece un seguimiento del balón ya que a veces se dan situaciones en las que el conjunto de árbitros no pueden ver con claridad en ciertas ocasiones y necesitan de una repetición para saber si por ejemplo el balón ha sobrepasado la línea de gol o incluso para saber si una falta se ha producido dentro o fuera del área de penalti a la hora de tomar una decisión que puede influir el ritmo del partido. Todas estas estadísticas también sirven de recopilación para decidir los ganadores de ciertos premios a final de temporadas o para el análisis de los técnicos de los equipos para su posterior estudio.

Las estadísticas tienen una gran influencia en un mundo deportivo en el que los datos y su análisis juegan cada vez un papel más importante. Todas las tecnologías utilizadas para recopilar las estadísticas mencionadas no tienen ninguna repercusión directa sobre los jugadores ya que se utilizan las cámaras de los eventos deportivos profesionales que se encuentran fuera del terreno de juego, de este modo se evita tener que equipar al jugador con nanotecnología y sensores que permitan de igual manera recopilar datos.

Se utilizará el sistema de reconocimiento y extracción de características de la herramienta ImageAI.

Tradicionalmente se han utilizado técnicas morfológicas, como puede ser la eliminación del fondo según unos márgenes en las intensidades de los colores RGB, el gradiente de Sobel (Wikipedia, la enciclopedia libre, 2021) para la detección de elementos en una imagen, la transformada de Hough (Wikipedia, la enciclopedia libre, 2020) para la detección de rectas, el filtro de Kalman (Wikipedia, 2021), y otras técnicas morfológicas.

ImageAI (Olafenwa & Olafenwa, Official English Documentation for ImageAI (GITGUB), 2021) es una biblioteca de Python de código abierto creada para permitir a los desarrolladores crear aplicaciones y sistemas con capacidades autónomas de Deep Learning[1] y Computer Vision utilizando pocas líneas de código simples.

En consecuencia, el objetivo será aplicar esta técnica de la forma correcta, definiendo los parámetros concretos que se ajusten a los mejores resultados posibles. Para esto, se estudiará cómo rinden los diferentes modelos que se pueden aplicar. También se atenderá al coste computacional que todo esto requiere, de forma que podamos valorar las capacidades del programar para actuar en tiempo real o no. Una vez ajustados todos estos a los resultados, usaremos estos datos para calcular algunas estadísticas de forma automática, a modo de ejemplo del alcance y potencial que pueden tener estas técnicas aplicadas en casos reales.

## 1.3 Software y herramientas.

El software ya antes mencionado que vamos a utilizar es ImageAI. Esta biblioteca está basada en el lenguaje de programación Python, en concreto utilizaremos su versión 3.7.6.

También se utilizará TensorFlow (Buhigas, 2018). Se trata de una biblioteca de software de código abierto para computación numérica, que utiliza gráficos de flujo de datos. Los nodos en las gráficas representan operaciones matemáticas, mientras que los bordes de las gráficas representan las matrices de datos multidimensionales (tensores) comunicadas entre ellos. Actualmente su flexibilidad y gran comunidad de desarrolladores lo ha posicionado como la herramienta líder en el sector del Deep Learning (Sas, 2020).

Algunas de las aplicaciones actuales de TensorFlow son la mejora de fotografías en Smartphone, ayudar al diagnóstico médico por ejemplo analizando radiografías. También es utilizado en procesamiento de imágenes que se complementa con lo anteriormente mencionado y es el ejemplo más claro del uso que le damos en este programa. Una de las aplicaciones más conocidas de TensorFlow es el software automatizado de procesamiento de imágenes, DeepDream. Se trata de un programa de visión artificial creado por el ingeniero de Google Alexander Mordvintsev, que utiliza una red neuronal convolucional para encontrar y mejorar patrones en imágenes mediante pareidolia algorítmica, creando así una apariencia alucinógena, similar a un sueño, creando imágenes deliberadamente sobre procesadas.

TensorFlow se construyó pensando en el código abierto y en la facilidad de ejecución y escalabilidad. Permite ser ejecutado en local y en la nube. La idea es que cualquiera pueda ejecutarlo. La disparidad en su uso es muy alta.

El futuro de TensorFlow se encuentra muy ligado al IoT[2]. Se calcula que todo ser humano está rodeado, al menos, por un total de aproximadamente 1000 a 5000 objetos. Pues IoT (Wikipedia, la enciclopedia libre, 2021) es la forma en la que cada objeto puede conectarse a internet.

[1]Deep learning es un tipo de machine learning que entrena a una computadora para realizar tareas como las de los seres humanos, como el reconocimiento facial, identificación de imágenes o predicción.

[2] Sus siglas significan “Internet of things” o “Internet de las cosas”. Se trata de tecnología puntera como Big Data, visión artificial, 5G o blockchain entre otras.

ImageAI se encuentra disponible en la web de GitHub[3]. Utiliza OpenCV, PyImageSearch perteneciente a la librería imutils.video. También Numpy, time y math, para diversas operaciones necesarias a lo largo del programa.

[3] GitHub es una plataforma de desarrollo colaborativo de software para alojar proyectos.

# Capítulo segundo: Desarrollo de la aplicación en imágenes

## 2.1 Tratamiento de imágenes convencional.

Como ya hemos mencionado antes una de las aplicaciones de ImageAI es la de detección de objetos, extracción de características y reconocimiento facial.

Tradicionalmente para hacer tratamiento de imágenes lo primero que se hace es convertir la imagen a escala de grises[4] consiguiendo una imagen con solo tonos de gris (incluidos blanco y negro). Esta matriz se representa digitalmente mediante una matriz f de M x N elementos f i, j = f (i; j). Usualmente se utilizan valores de intensidad enteros en [0; 255] por lo que es necesario pasar la imagen a formato doble.

En la herramienta de MATLAB el comando necesario para pasar una imagen a escala de grises es rgb2gray() y pasar la imagen a formato doble es double(). Para facilitar la explicación proponemos un ejemplo en el entorno de MATLAB.

```
f=imread('imagen.jpg');
f=rgb2gray(f);
figure(1), imshow(f);
figure(2), imhist(f);
f=double(f);
[M, N]=size(f);
```

El histograma (Wikipedia, la enciclopedia libre, 2021) es una gráfica usada en estadística que aplicada a imágenes permite sistematizar algunos tratamientos. Es una representación gráfica de una variable en forma de barras, donde la superficie de cada barra es proporcional a la frecuencia de los valores representados, en este caso, el número de veces que se repiten los valores de intensidades una vez la imagen se ha pasado a escala de grises.

La herramienta histograma puede generar las siguientes estadísticas:

[4] En computación una escala de grises es empleada en la imagen digital. Cada valor del píxel posee un valor equivalente a una graduación de gris. Las imágenes representadas de este tipo están compuestas por sombra de grises.

- Contabilización de pixeles
- Valor de píxeles medio, mediano y modal
- Valores de píxeles mínimos y máximos
- Desviación estándar y variación de valores de píxeles

A partir de este punto se pueden realizar diversas operaciones para tratamiento de imagen. El tratamiento de imágenes es una técnica que permite modificar una imagen con el objetivo de lograr una mayor calidad o realismo, o bien para obtener una composición totalmente diferente que distorsione la realidad.

Para ello existen varias técnicas aplicables: eliminación de ruido, corrección de ojos rojos, desemborronado, mejora de contraste, resalte de contornos, ajuste de color, etc.

El resultado es una modificación de la imagen. En muchos casos el tratamiento es un paso previo al análisis. El suavizado es una técnica muy habitual y sirve de ejemplo de fácil comprensión. Es un procedimiento para homogenizar la imagen con el fin de eliminar el ruido. Un ejemplo de técnica de suavizado es el siguiente:

```
fMedian=zeros(size(f));
for i=9:M-8
    for j=9:N-8
        fMedian(i,j)=median(reshape(f(i-8:i+8,j-8:j+8),17*17,1)); %máscara 3x3
    end
end
figure(2),imshow(uint8(fMedian));
```

Cada punto de la imagen resultado g es un promedio de un fragmento de la original f Se puede expresar mediante:

$$g_{i,j} = \frac{1}{Nv} \sum_{(l,m)\in V(i,j)} f_{l,m}$$

siendo V (i; j) una vecindad centrada en (i; j).

A continuación, una vez que se dispone de la imagen en blanco y negro con el análisis previo realizado se puede aplicar separación de regiones, esto es por ejemplo separar el terreno de un campo de fútbol de la grada donde se sitúan los espectadores.

La separación de regiones (Arahal, 2018) es una división de la imagen en diferentes regiones que mantienen una relación entre sí con el fin de diferenciar objetos en la imagen. (niveles de grises, textura etc.) Es un paso previo para obtener información sobre localización de objetos. Sirve

como tratamiento para zonas localizadas, cálculo de áreas, ángulos y distancias, seguimiento de objetos e interpretar escenas.

Para el cálculo de regiones se utilizarán distintos tipos de información como son los niveles de intensidad, variación de intensidad, color o textura. Se pueden aplicar diferentes métodos para calcular la separación de regiones. Algunos de ellos requieren binarizar[5] la imagen y esto se puede realizar mediante un umbral. Se establece un nivel de gris (umbral) y a partir de ese nivel todos los valores por encima pasan a ser blanco (1) y todos los valores por debajo pasan a ser negro (0). De esta forma se pueden separar un objeto perteneciente a una imagen, volviendo a el ejemplo utilizando este método podríamos separar fácilmente el terreno de juego que en su mayoría es verde de la grada del público que toma otros colores diferentes. La precisión es un detalle muy importante a la hora de separar regiones.

Por otro lado, se tiene la técnica de los contornos (Arahal, 2018). Se trata de una técnica de separación de regiones en la que se utilizarán fronteras, líneas que delimitan regiones diferentes basándose en los niveles de intensidad de dos o más píxeles contiguos.

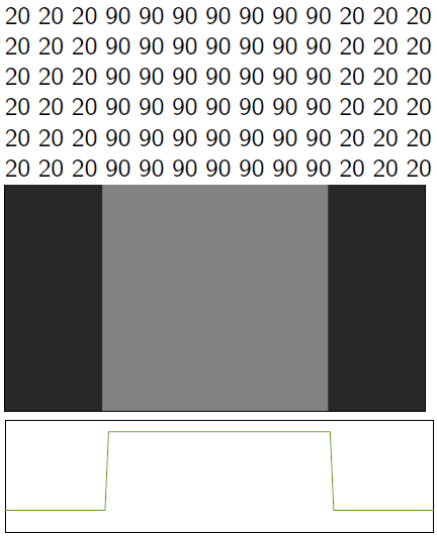


*Ilustración 1: Niveles de contorno.*

En la imagen se puede observar un ejemplo de detección de contornos por niveles de intensidad con su respectiva representación en escala de grises. Los cambios bruscos en los niveles de intensidad se pueden corresponder con contornos, este cambio se percibe recorriendo la imagen en una dirección, en este ejemplo se recorre horizontalmente. Como la intensidad lumínica varía de un punto a otro se produce un gradiente, este gradiente nos indica la dirección y sentido de máximo cambio mientras que su módulo nos indica el ritmo de cambio sobre la dirección.

[5] La imagen resultante al binarizar solo puede obtener dos valores posibles 0 o 1. En el procesamiento de imágenes se le atribuye al negro y al blanco.

Estos gradientes pueden calcularse mediante plantillas[6]. Las plantillas más usuales son las llamadas operadores de Prewitt, Sobel y Roberts.

El operador Sobel (Wikipedia, la enciclopedia libre, 2021) es utilizado en procesamiento de imágenes, especialmente en algoritmos de detección de bordes. Técnicamente es un operador diferencial discreto que calcula una aproximación al gradiente de la función de intensidad de una imagen. Para cada punto de la imagen a procesar, el resultado del operador Sobel es tanto el vector gradiente correspondiente como la norma de este vector. También sirve para captar la variación de intensidad luminosa, muy útil para seguir el recorrido de los jugadores.

| Dir. | Roberts | Prewitt | Sobel |
|---|---|---|---|
| Y | $\begin{matrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{matrix}$ | $\begin{matrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{matrix}$ | $\begin{matrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{matrix}$ |
| X | $\begin{matrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{matrix}$ | $\begin{matrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{matrix}$ | $\begin{matrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{matrix}$ |

Nota: las plantillas suelen llevar un coeficiente que se ha omitido aquí.

*Ilustración 2: Plantillas de gradientes.*

El coeficiente de escala sigue la siguiente notación: $1/(2+K)$ siendo K = 0,1,2 en los operadores de Roberts, Prewitt y Sobel respectivamente.

Otra técnica muy utilizada para la separación de regiones es la transformada de Hough (Wikipedia, la enciclopedia libre, 2020). Esta transformada se especializa en líneas rectas, aunque se ha de ser cautos ya que los contornos obtenidos con gradiente o laplacianas pueden pertenecer o no a rectas. Cada punto del contorno vota a todas las posibles rectas que pasan por él. Estas rectas están parametrizadas en coordenadas polares quedando definida por el módulo y su ángulo.

La transformada de Hough fue propuesta y patentada en 1962 por Paul Hough. Inicialmente esta técnica solo se aplicaba a la detección de rectas en una imagen. Más tarde se extendió para identificar cualquier figura que se pudiera describir con unos cuantos parámetros; más comúnmente circunferencias y elipses. La transformada de Hough, como se usa actualmente, fue inventada por Richard Duda y Peter Hart en 1972, quienes lo llamaron "Transformada de Hough Generalizada". Dana H. Ballard popularizó este método en la comunidad de Visión por Computadora en un artículo publicado en 1981, llamado Generalizando la transformada de Hough para detectar figuras arbitrarias. La condición de que pueda utilizarse con otras formas geométricas es que admita parametrización.

[6] Las plantillas son matrices que junto a la imagen original se convoluciona para formar una imagen nueva.

En MATLAB las funciones hough(), houghpeaks() o houghlines() se utilizan para implementar esta técnica. Ejemplo de código:

```
f = imread('cameraman.tif');[7]

plc = edge( f,'canny');

[ H,theta,rho] = hough( plc);

picos = houghpeaks(H, 2);

selecc = houghlines( plc,theta,rho,picos);
```

Finalmente se destacan los métodos morfológicos [ (EDMANS & Marcos, 2006)] que se caracterizan por ser operaciones no lineales y son interesantes por la flexibilidad y variedad de aplicaciones como el realce o la separación, su interpretación geométrica intuitiva, alta robustez y fácil implementación en los programas de ordenador. Se trata de una mejora que ayuda a intensificar la separación de regiones. Existen dos operaciones esenciales: apertura que consiste en una erosión[8] de la imagen seguido de una dilatación[9] y cierre que consiste en una dilatación seguido de una erosión. La erosión es la función dual de la dilatación (pero no es la inversa, es importante destacarlo ya que al aplicar cualquiera de estas dos operaciones a una imagen el resultado final no es el mismo que el original). El objetivo de utilizar apertura es eliminar motas y puntos espurios que aparecen como consecuencia de utilizar suavizados y el cierre por contraposición se utiliza para rellenar fisuras. Estos métodos morfológicos suelen utilizarse en siluetas.

Ejemplo de código en la herramienta MATLAB para realizar el proceso de binarización:

```
g = uint8(zeros(size(fMedian)));

for i = 1:M

    for j = 1:N

        if fMedian(i,j) < umbral %El umbral puede ser averiguado por ensayo y
error.

            g(i,j) = 255;

        end

    end

end
```

[7] cameraman.tif es una imagen que se encuentra implícita en los archivos de MATLAB.

[8] Erosión: Sea una imagen F y una plantilla p, el proceso de erosión $F - p$ (– es el operador mínimo) consiste en solapar la plantilla en la imagen, y ver si todos los puntos a uno de la plantilla coinciden con los de la imagen, poniéndolo a uno y cero en caso contrario.

[9] Dilatación: Sea una imagen F y una plantilla p el proceso de dilatación $F + p$ (+ es el operador máximo) consiste en ir desplazando el elemento estructurante p por toda la imagen, y si alguno de los elementos de la matriz coincide con un pixel del entorno, entonces el pixel donde está centrado en F se pone a uno.

El siguiente código es un ejemplo de aplicación de métodos morfológicos:

```
mask1 = strel('disk',19); %Ajustamos el parámetro de la máscara para que quede
lo más precisa posible.

fClose = Cierre(g, mask1); %Llamamos a la función de cierre.

mask2 = strel('disk',20);

fOpen=Apertura(fClose,mask2); %Llamamos a la función de apertura.

figure(9); imshow(fOpen-g),title('Resultado menos imagen original');
```

*Ilustración 3: Imagen de estadio de fútbol.*

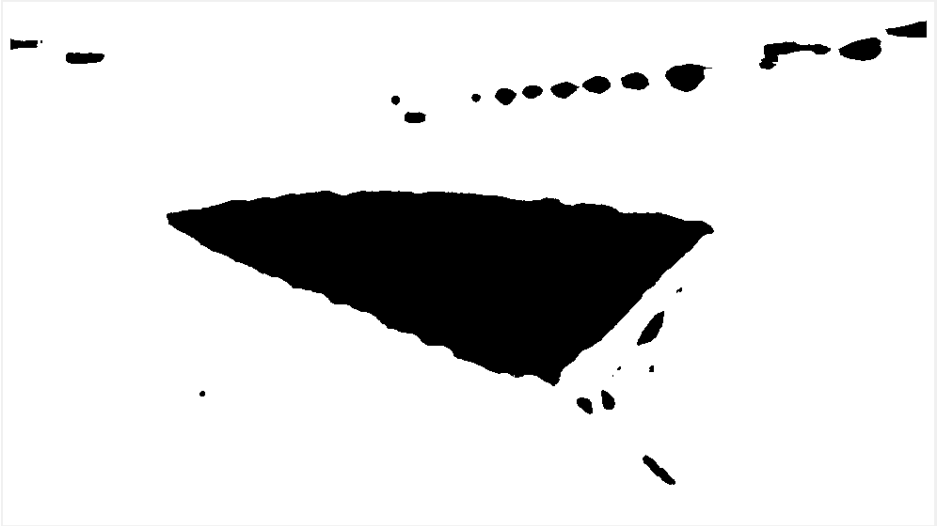

*Ilustración 4: Separación de regiones campo-grada.*

*Ilustración 5: Binarización del terreno de juego.*

## 2.2 Instalación y dependencias de la herramienta ImageAI.

ImageAI (Olafenwa & Olafenwa, Official English Documentation for ImageAI (GITGUB), 2021) admite una lista de algoritmos de aprendizaje automático de última generación para la detección de objetos, detección de video, seguimiento de objetos de video y entrenamientos de predicción de imágenes. ImageAI actualmente admite la predicción y el entrenamiento de imágenes utilizando cuatro algoritmos de aprendizaje automático diferentes, entrenados en el conjunto de datos ImageNet-1000 [ (Lab, University, & University, 2021)]. ImageNet es una base de datos de imágenes organizada según la jerarquía de WordNet, en la que cada nodo está representado por una infinidad de imágenes. Es fundamental en el campo de la visión artificial y en la investigación de Deep-learning, además los datos se encuentran disponibles de forma gratuita con uso no comercial para los investigadores. Utiliza RetinaNet, YOLOv3 y TinyYOLOv3 capacitados en el conjunto de datos COCO[10]. Finalmente, ImageAI permite entrenar modelos personalizados para realizar la detección y el reconocimiento de nuevos objetos.

Se emplea la versión **2.1.6** de ImageAI. Las novedades respecto de su última actualización son:

- Soporta Tensorflow 2.4.0.
- SqueezeNet reemplazado por MobileNetV2.
- Añadido TensorFlow 2.x y entrenamiento para los modelos de ResNet y RetinaNet.
- Sustituido '.predictImage()' function por '.classifyImage()'
- Renombrado los modelos:
    - ResNet >> ResNet50.
    - DenseNet >> DenseNet121.

Para su uso es necesario instalar las siguientes dependencias:

```
pip install tensorflow==2.4.0

pip install keras==2.4.3 numpy==1.19.3 pillow==7.0.0 scipy==1.4.1 h5py==2.10.0
matplotlib==3.3.2 opencv-python keras-resnet==0.2.0

pip install folium==0.2.1

pip install imgaug==0.2.5

pip install imageai --upgrade
```

Se utiliza el siguiente comando para establecer una carpeta en drive como directorio predeterminado, dónde se encuentran los archivos necesarios para la ejecución del programa como los modelos o las imágenes necesarias.

[10] (Common Objects in Context) para la segmentación de imágenes en Python con bibliotecas que incluyen PyCoco y Tensorflow Keras. Es un conjunto de datos de referencia para la detección de objetos.

```
from google.colab import drive

drive.mount('/content/drive/')

%cd /content/drive/MyDrive/TFG/ImageAI-master/
```

## 2.3 Desarrollo de la herramienta ImageAI para detección de objetos.

Se va a explicar detenidamente la ejecución del código detallando en cada momento cual es el resultado de la instrucción que se lleva a cabo. Para empezar, se encuentra a continuación el código empleado para la detección de objetos en imágenes (Olafenwa, 2021). El modelo utilizado en este caso es YOLOv3 y el archivo utilizado es *"yolo.h5"*

```
from imageai.Detection import ObjectDetection
import os

execution_path = os.getcwd()
# USAR YoloV3
detector = ObjectDetection()
detector.setModelTypeAsYOLOv3()
detector.setModelPath( os.path.join(execution_path , "yolo.h5"))
detector.loadModel()

detections =
detector.detectObjectsFromImage(input_image=os.path.join(execution_path ,
"data-images/seleccion_1.jpg"), output_image_path=os.path.join(execution_path
, "data-images/seleccion_1_nueva.jpg"), minimum_percentage_probability=10)
```

Comenzamos importando la clase *ObjectDetection* perteneciente a la librería *ImageAI.Detection* e importamos *os*[11].

Se define el detector creando una nueva instancia de la clase *ObjectDetection* y estableciendo el tipo de modelo como "yolo.h5". Cargamos el modelo.

Se ejecuta la función *detectObjectsFromImage* y añadimos la ruta a la imagen sobre la que vamos a aplicar la detección, y se establece la ruta a la nueva imagen que nos devuelve la función. Los directorios de entrada y salida de imágenes son las carpetas creadas en el entorno de Google Drive en la que se encuentran las imágenes que analizaremos y la salida dónde queremos que se guarden los resultados.

[11] Varias interfaces de sistema operativo.

La función devuelve una matriz de diccionarios. Cada diccionario corresponde al número de objetos detectados en la imagen con las propiedades (nombre del objeto), percent_probability (porcentaje de probabilidad de detección) y box_points (las coordenadas x1, y1, x2 e y2 del cuadro delimitador del objeto). Se propone reconocer los objetos de manera independiente en la siguiente imagen que corresponde con la plantilla de la selección española de fútbol en 2021.

El parámetro mínimum_percentage_probability se utiliza para determinar la integridad de los resultados de la detección. Reducir el valor muestra más objetos detectados en la imagen, pero aumenta el error de detección pudiendo llegar a confundir dos objetos diferentes o llegando a detectar erróneamente objetos no deseados mientras que aumentar el valor asegura que se detecten los objetos con la mayor precisión.

*Ilustración 6: Imagen de la plantilla de la selección de fútbol en el año 2021.*

*Ilustración 7: Imagen anterior con algoritmo de detección de objetos aplicado.*

De igual manera se procede a aplicar el mismo algoritmo sobre la plantilla de la selección ganadora del mundial de fútbol de 2010 en Sudáfrica.


*Ilustración 8: Imagen con algoritmo de detección de objetos aplicado.*

Cabe destacar que en este caso se ha utilizado el algoritmo para la detección de los jugadores (personas), este algoritmo también funciona con una diversidad mayor de tipos de objetos: animales, muebles, vehículos etc. Estas líneas de código son funciones que quedan definidas en el archivo que se encuentra en el directorio *imageai/Detection/_init_.py* que quedará expuesto en un capítulo posterior.

```
person  :  92.86527037620544  :  [141, 41, 339, 250]
--------------------------------
person  :  99.4719922542572  :  [393, 267, 622, 560]
--------------------------------
person  :  95.09682059288025  :  [540, 288, 771, 564]
--------------------------------
person  :  98.23715686798096  :  [431, 22, 584, 296]
--------------------------------
person  :  89.85116481781006  :  [550, 15, 700, 322]
--------------------------------
person  :  96.48433327674866  :  [308, 41, 451, 309]
--------------------------------
person  :  96.3990867137909  :  [677, 48, 832, 339]
--------------------------------
person  :  99.34040904045105  :  [64, 43, 201, 521]
--------------------------------
person  :  98.67589473724365  :  [147, 176, 303, 523]
--------------------------------
person  :  90.58893322944641  :  [284, 215, 434, 519]
--------------------------------
person  :  64.0281081199646  :  [715, 294, 874, 546]
--------------------------------
```

*Ilustración 9: Tipo de objeto | Grado de confianza | coords bounding box. (YOLOv3)*

A continuación, se procede a extraer cada objeto de la imagen de entrada y guardar de forma independiente. El código es prácticamente idéntico al anterior, se guardará cada objeto detectado como una imagen separada mediante un bucle for y el parámetro extract_detected_objects.

```
from imageai.Detection import ObjectDetection

import os

execution_path = os.getcwd()

detector = ObjectDetection()

detector.setModelTypeAsYOLOv3()

detector.setModelPath( os.path.join(execution_path , "yolo.h5"))

detector.loadModel()


detections, objects_path =
detector.detectObjectsFromImage(input_image=os.path.join(execution_path ,
"data-images/seleccion-2021.jpg"),
output_image_path=os.path.join(execution_path , "data-images/indep_2021"),
minimum_percentage_probability=15,  extract_detected_objects=True)
```

```
for eachObject, eachObjectPath in zip(detections, objects_path):

    print(eachObject["name"] , " : " , eachObject["percentage_probability"], "
: ", eachObject["box_points"] )

    print("Object's image saved in " + eachObjectPath)

    print("--------------------------------")
```

Se llama a detectObjectsFromImage (), se analiza en la ruta de la imagen de entrada, se establece una ruta de salida y un parámetro adicional extract_detected_objects = True. Este parámetro indica que la función debe extraer cada objeto detectado de la imagen y guardarlo en una imagen separada. Este parámetro se define como False por defecto. La función creará un nuevo directorio que es la ruta de la imagen de salida + "-objetos" y guarda todas las imágenes extraídas en este nuevo directorio, siendo el nombre de cada imagen el nombre del objeto detectado + "-" + un número que corresponde al orden en el que se detectaron los objetos.

Este nuevo parámetro, que se configura para extraer y guardar los objetos detectados como una imagen separada, hará que la función devuelva dos valores. Una matriz de diccionarios con cada diccionario correspondiente a un objeto detectado y una matriz de las rutas a las imágenes guardadas de cada objeto detectado y extraído, organizadas en el orden en que los objetos están en la primera matriz.

La función *detectObjectsFromImage ()* devuelve el porcentaje de probabilidad para cada objeto detectado. La función tiene un parámetro minimum_percentage_probability, cuyo valor predeterminado es 50 pudiendo variar en el intervalo [0,100]. Para este ejemplo se establece dicho valor en 15 lo que significa que para devolver un objeto detectado su porcentaje de probabilidad debe ser 15 o superior. Se puede ajustar la detección de objetos estableciendo un minimum_percentage_probability igual a un valor más pequeño para detectar una mayor cantidad de objetos o un valor más alto para detectar una menor cantidad de objetos. Es importante experimentar con este parámetro para tratar de detectar el mayor número de objetos de interés sin que se detecten objetos no deseados.

Las siguientes imágenes muestran algunas de las extracciones que el algoritmo detectó y guardó independientemente.


*Ilustración 10: Extracciones de objetos de manera independiente.*

```
person  :  91.43697619438171  :  [28, 13, 68, 74]
Object's image saved in /content/drive/My Drive/TFG/ImageAI-master/data-images/seleccion2010_indepenyolo.jpg-objects/person-1.jpg
--------------------------------
person  :  96.8235731124878  :  [62, 6, 104, 76]
Object's image saved in /content/drive/My Drive/TFG/ImageAI-master/data-images/seleccion2010_indepenyolo.jpg-objects/person-2.jpg
--------------------------------
person  :  93.91452074050903  :  [139, 17, 182, 78]
Object's image saved in /content/drive/My Drive/TFG/ImageAI-master/data-images/seleccion2010_indepenyolo.jpg-objects/person-3.jpg
--------------------------------
person  :  90.73410630226135  :  [181, 18, 226, 82]
Object's image saved in /content/drive/My Drive/TFG/ImageAI-master/data-images/seleccion2010_indepenyolo.jpg-objects/person-4.jpg
--------------------------------
person  :  96.06874585151672  :  [224, 17, 264, 83]
Object's image saved in /content/drive/My Drive/TFG/ImageAI-master/data-images/seleccion2010_indepenyolo.jpg-objects/person-5.jpg
--------------------------------
person  :  98.90502691268921  :  [263, 20, 302, 86]
Object's image saved in /content/drive/My Drive/TFG/ImageAI-master/data-images/seleccion2010_indepenyolo.jpg-objects/person-6.jpg
--------------------------------
person  :  73.20943474769592  :  [0, 6, 31, 97]
Object's image saved in /content/drive/My Drive/TFG/ImageAI-master/data-images/seleccion2010_indepenyolo.jpg-objects/person-7.jpg
--------------------------------
person  :  84.16476845741272  :  [99, 17, 141, 96]
Object's image saved in /content/drive/My Drive/TFG/ImageAI-master/data-images/seleccion2010_indepenyolo.jpg-objects/person-8.jpg
--------------------------------
person  :  69.96843218803406  :  [70, 48, 109, 112]
Object's image saved in /content/drive/My Drive/TFG/ImageAI-master/data-images/seleccion2010_indepenyolo.jpg-objects/person-9.jpg
--------------------------------
person  :  69.11114454269409  :  [110, 47, 151, 114]
Object's image saved in /content/drive/My Drive/TFG/ImageAI-master/data-images/seleccion2010_indepenyolo.jpg-objects/person-10.jpg
--------------------------------
person  :  98.35211038589478  :  [146, 49, 193, 120]
Object's image saved in /content/drive/My Drive/TFG/ImageAI-master/data-images/seleccion2010_indepenyolo.jpg-objects/person-11.jpg
--------------------------------
person  :  90.35298228263855  :  [189, 54, 233, 118]
Object's image saved in /content/drive/My Drive/TFG/ImageAI-master/data-images/seleccion2010_indepenyolo.jpg-objects/person-12.jpg
--------------------------------
person  :  96.85490131378174  :  [227, 53, 267, 122]
Object's image saved in /content/drive/My Drive/TFG/ImageAI-master/data-images/seleccion2010_indepenyolo.jpg-objects/person-13.jpg
--------------------------------
person  :  70.64650058746338  :  [262, 61, 300, 121]
Object's image saved in /content/drive/My Drive/TFG/ImageAI-master/data-images/seleccion2010_indepenyolo.jpg-objects/person-14.jpg
--------------------------------
person  :  97.45500087738037  :  [2, 44, 33, 157]
Object's image saved in /content/drive/My Drive/TFG/ImageAI-master/data-images/seleccion2010_indepenyolo.jpg-objects/person-15.jpg
--------------------------------
person  :  91.75204038619995  :  [40, 47, 73, 156]
Object's image saved in /content/drive/My Drive/TFG/ImageAI-master/data-images/seleccion2010_indepenyolo.jpg-objects/person-16.jpg
--------------------------------
person  :  99.50358271598816  :  [302, 55, 329, 190]
Object's image saved in /content/drive/My Drive/TFG/ImageAI-master/data-images/seleccion2010_indepenyolo.jpg-objects/person-17.jpg
--------------------------------
```

*Ilustración 11: Extracciones de objetos de manera independiente.*

Existe la función *detectCustomObjectsFromImage () que* permite extraer de manera independiente un objeto en concreto dentro de una larga lista de objetos permitidos. El ejemplo que se muestra solamente contiene personas, pero se define a continuación una sentencia para que pueda quedar de ejemplo de cómo filtrar varios objetos específicos.

```
custom_objects = detector.CustomObjects(car=True, motorcycle=True)

detections =
detector.detectCustomObjectsFromImage(custom_objects=custom_objects,
input_image=os.path.join(execution_path , "data-images/image3.jpg"),
output_image_path=os.path.join(execution_path , "data-
images/image3custom.jpg"), minimum_percentage_probability=30)
```

```
car   :  98.19014072418213   :  [191, 159, 368, 309]
--------------------------------
motorcycle   :  95.87432146072388   :  [266, 202, 344, 316]
--------------------------------
```

*Ilustración 12: Extracciones de objetos específicos de manera independiente.*

Esta función puede seguir siendo utilizada, pero sin embargo con la última actualización se categorizó como obsoleta. Se recomienda el uso de *detectObjectsFromImage ()* en su lugar, ya que tiene todos los posibles parámetros de entrada y devuelve todos los valores de *detectCustomObjectsFromImage().* Se le pasa como argumento de entrada un parámetro que indique si existe algún objeto que quiera ser extraído únicamente de la imagen como por ejemplo custom_objects = custom, siendo custom = detector.CustomObjects(person=True).

ImageAI proporciona diferentes velocidades de detección. Estas velocidades permiten reducir el tiempo de detección en un (20% - 80%) con cambios leves y resultados de detección precisos. Junto con la reducción del parámetro de probabilidad de porcentaje mínimo, las detecciones pueden igualar la velocidad normal reduciendo drásticamente el tiempo de detección. Las velocidades de detección disponibles son "normal" (predeterminada), "fast", "faster", "fastest" y "flash".

```
detector.loadModel(detection_speed="normal")

# detector.loadModel(detection_speed="fast")

# detector.loadModel(detection_speed="faster")

# detector.loadModel(detection_speed="fastest")

# detector.loadModel(detection_speed="flash")
```

```
person   :  67.08927750587463  :  [362, 49, 644, 314]
--------------------------------
person   :  92.5142228603363  :  [163, 36, 338, 280]
--------------------------------
person   :  60.25691032409668  :  [313, 51, 457, 297]
--------------------------------
person   :  80.21875023841858  :  [550, 33, 702, 317]
--------------------------------
person   :  95.47942280769348  :  [671, 47, 837, 499]
--------------------------------
person   :  99.78219866752625  :  [72, 55, 213, 510]
--------------------------------
person   :  99.07151460647583  :  [149, 178, 323, 519]
--------------------------------
person   :  95.87342143058777  :  [268, 218, 437, 524]
--------------------------------
person   :  94.14601922035217  :  [416, 256, 610, 544]
--------------------------------
person   :  89.23340439796448  :  [554, 272, 749, 543]
--------------------------------
person   :  88.05240988731384  :  [716, 302, 890, 553]
--------------------------------
sports ball   :  31.33937418460846  :  [412, 393, 445, 424]
--------------------------------
```

*Ilustración 13: Resultados de aplicar velocidad "fast" con el modelo YOLOv3.*

```
person  :  72.59347438812256  :  [437, 77, 593, 415]
--------------------------------
person  :  60.84123253822327  :  [570, 83, 711, 432]
--------------------------------
person  :  86.8051290512085  :  [681, 118, 839, 467]
--------------------------------
person  :  90.99949598312378  :  [89, 99, 242, 527]
--------------------------------
person  :  64.87907767295837  :  [182, 184, 328, 559]
--------------------------------
person  :  78.74552011489868  :  [307, 200, 447, 576]
--------------------------------
person  :  71.78152203559875  :  [421, 195, 586, 588]
--------------------------------
person  :  57.82100558280945  :  [190, 46, 332, 239]
--------------------------------
person  :  44.47240233421326  :  [545, 26, 690, 312]
--------------------------------
person  :  83.25980305671692  :  [311, 60, 454, 308]
--------------------------------
person  :  67.78962016105652  :  [571, 285, 717, 515]
--------------------------------
person  :  60.20341515541077  :  [743, 346, 884, 558]
--------------------------------
```

*Ilustración 14: Resultados de aplicar velocidad "fastest" con el modelo YOLOv3.*

```
person  :  48.92940819263458  :  [421, 56, 580, 333]
--------------------------------
person  :  74.88034963607788  :  [554, 59, 711, 348]
--------------------------------
person  :  51.957279443740845  :  [695, 109, 847, 450]
--------------------------------
person  :  88.39104175567627  :  [66, 63, 202, 522]
--------------------------------
person  :  59.913796186447144  :  [108, 107, 258, 512]
--------------------------------
person  :  54.70535159111023  :  [175, 133, 326, 508]
--------------------------------
person  :  60.905832052230835  :  [278, 133, 426, 523]
--------------------------------
person  :  47.10249602794647  :  [407, 272, 587, 551]
--------------------------------
person  :  78.05079221725464  :  [562, 284, 722, 542]
--------------------------------
person  :  56.34065866470337  :  [705, 327, 870, 561]
--------------------------------
```

*Ilustración 15: Resultados de aplicar velocidad "flash" con el modelo YOLOv3.*

Como se puede observar existe una relación entre la probabilidad de acierto y la velocidad de detección que, cuanto menor es, mayor resultan ser las dimensiones de las bounding box correspondientes a cada objeto detectado, lo que implica una probabilidad peor. Se han obtenido tiempos de ejecución para las diferentes opciones de velocidad siendo 9s el tiempo de ejecución para la velocidad "normal", 7s para velocidad "fast", 4s para velocidad "fastest" y finalmente 4s para la velocidad "flash".

ImageAI ofrece opciones para ocultar el nombre de los objetos detectados y/o el porcentaje de probabilidad de la imagen detectada devuelta. Usando las funciones *detectObjectsFromImage ()* y *detectCustomObjectsFromImage (),* los parámetros *display_object_name* y *display_percentage_probability* se pueden establecer en True o False individualmente.

```
detections =
detector.detectObjectsFromImage(input_image=os.path.join(execution_path ,
"data-images/image3.jpg"), output_image_path=os.path.join(execution_path ,
"data-images/image3new_nodetails.jpg"), minimum_percentage_probability=30,
display_percentage_probability=False, display_object_name=False)
```

Como se ha mencionado al principio de la sección, el modelo que se ha utilizado hasta el momento en la ejecución del código ha sido *"yolo.h5"* de YOLOv3. En los siguientes ejemplos vamos a realizar el mismo procedimiento con los modelos "*Yolo-tiny.h5*" usado por TinyYOLOv3 y el modelo *"Resnet50_coco_best_v2.0.1.h5*" usado por RetinaNet. La diferencia entre estos modelos se basa en el peso del archivo y en la velocidad de detección siendo una más robusta que otra. RetinaNet Tiene un tamaño de 145 Mb. Ofrece un buen rendimiento y precisión con un tiempo de detección mayor y TinyYOLOv3 tiene un tamaño de 34mb, pesa poco en comparación a los modelos anteriores porque está optimizado para ser rápido y ofrecer un rendimiento decente. YOLOv3 pesa 237mb, es el más pesado de todos y tiene una actuación y precisión moderada con un tiempo de detección normal.

A continuación, vamos a realizar el mismo procedimiento, pero aplicando el modelo YOLOTinyV3 con el archivo *"Yolo-tiny.h5":*

```
from imageai.Detection import ObjectDetection

import os

execution_path = os.getcwd()
```

```
# USAR Yolo-tiny

detector = ObjectDetection()

detector.setModelTypeAsTinyYOLOv3()

detector.setModelPath( os.path.join(execution_path , "yolo-tiny.h5"))

detector.loadModel()

# detector.loadModel(detection_speed="fast")

detections =
detector.detectObjectsFromImage(input_image=os.path.join(execution_path ,
"data-images/seleccion_2010.jpg"),
output_image_path=os.path.join(execution_path , "data-
images/seleccion_2010_tiny.jpg"), minimum_percentage_probability=15)


for eachObject in detections:

    print(eachObject["name"] , " : ", eachObject["percentage_probability"], "
: ", eachObject["box_points"] )

    print("--------------------------------")
```

```
person  :  47.2014456987381  :  [159, 31, 360, 262]
--------------------------------
person  :  25.047898292541504  :  [303, 29, 459, 333]
--------------------------------
person  :  18.169431388378143  :  [408, 26, 608, 339]
--------------------------------
person  :  45.525574684143066  :  [59, 15, 194, 541]
--------------------------------
person  :  91.7362630367279  :  [132, 135, 313, 559]
--------------------------------
person  :  82.24655389785767  :  [261, 178, 417, 543]
--------------------------------
person  :  24.08575415611267  :  [404, 226, 619, 516]
--------------------------------
person  :  41.29774868488312  :  [553, 247, 751, 545]
--------------------------------
person  :  59.94440317153931  :  [703, 284, 886, 554]
--------------------------------
person  :  16.70837700366974  :  [647, 327, 793, 546]
--------------------------------
sports ball  :  57.30002522468567  :  [410, 372, 449, 450]
--------------------------------
```

*Ilustración 16: Tipo de objeto | porcentaje de prob (%) | coords bounding box. (TinyYOLOv3).*

Como se puede observar la tabla resultante de utilizar el modelo *"YOLOv3"* respecto de *"TinyYOLOv3"* difiere en la precisión, estas probabilidades son bastante peores a consecuencia de realizar la detección en un tiempo menor. Se puede observar que en la tabla aparece un objeto denominado "sport ball", este objeto es un ejemplo de fallo de precisión porque confunde la rodilla de un jugador que tiene forma esférica con un balón de fútbol.

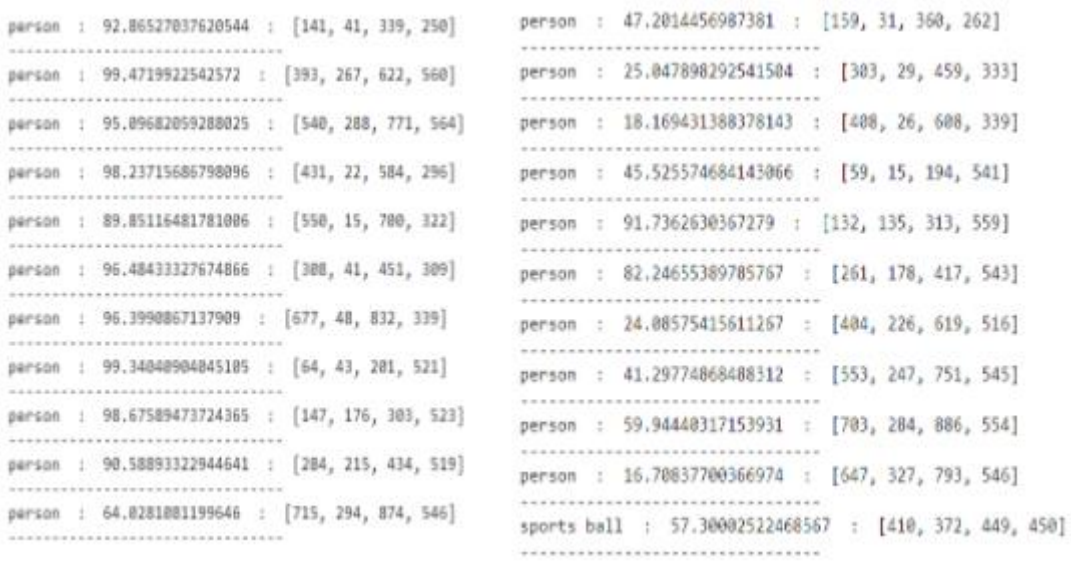


*Ilustración 17: Tabla comparativa entre YOLOv3 y TinyYOLOv3*

Se han obtenido tiempos de ejecución para las diferentes opciones de velocidad siendo 6s el tiempo de ejecución para la velocidad “normal”, 3s para velocidad “fast”, 2s para velocidad “fastest” y finalmente 2s para la velocidad “flash”.

Para acabar este capítulo vamos a utilizar el modelo de detección RetinaNet con el archivo "*Resnet50_coco_best_v2.0.1.h5*, el código utilizado es el siguiente:

```
from imageai.Detection import ObjectDetection

import os


execution_path = os.getcwd()


# USAR RetinaNet

detector = ObjectDetection()

detector.setModelTypeAsRetinaNet()

detector.setModelPath(os.path.join(execution_path,
"resnet50_coco_best_v2.1.0.h5"))

detector.loadModel()


#detector.loadModel(detection_speed="fast")

detections =
             detector.detectObjectsFromImage(input_image=os.path.join(executio
```

```
            n_path , "data-images/seleccion_2010.jpg"),
            output_image_path=os.path.join(execution_path , "data-
            images/seleccion_2010_resNet.jpg"),
            minimum_percentage_probability=15)


for eachObject in detections:

    print(eachObject["name"] , " : ", eachObject["percentage_probability"], "
: ", eachObject["box_points"] )

    print("--------------------------------")
```

| | Velocidad | | | | |
|---|---|---|---|---|---|
| **Modelo** | **normal** | **fast** | **fastest** | **flash** | **Tamaño** |
| **YOLOv3** | **9s** | **7s** | **4s** | **4s** | **237mb** |
| **TinyYOLOv3** | **6s** | **3s** | **2s** | **2s** | **34mb** |
| **ResNet** | **17s** | **17s** | **17s** | **17s** | **145mb** |

*Tabla 1: Tabla comparativa de datos para clasificación de imágenes.*

## 2.4 Desarrollo de la herramienta ImageAI para predicción de imágenes.

ImageAI (Olafenwa & Olafenwa, GitHub, s.f.) proporciona cuatro algoritmos y tipos de modelos diferentes para realizar predicciones de imágenes. Los cuatro algoritmos proporcionados para la predicción de imágenes son: MobileNetV2, ResNet50, InceptionV3 y DenseNet121. Cada uno de estos algoritmos tiene archivos de modelo individuales que utilizaremos según el algoritmo elegido.

En esta sección se trabaja con algunas imágenes de estadios de fútbol de manera que el algoritmo es capaz de detectar si existen gradas, anfiteatros, marcadores o si lo asocia a otro tipo de estructuras como teatros, pistas de aterrizaje, etc.

El código utilizado se plasma a continuación. Comenzamos por el modelo ResNet50 del cual se ha obtenido un tiempo de ejecución de 3s.

La imagen utilizada es la siguiente:

*Ilustración 18: Imagen de estadio2 de fútbol.*

```
from imageai.Classification import ImageClassification

import os

execution_path = os.getcwd()


prediction = ImageClassification()

prediction.setModelTypeAsResNet50()

prediction.setModelPath(os.path.join(execution_path,
"resnet50_imagenet_tf.2.0.h5"))
```

```
prediction.loadModel()


predictions, probabilities =
prediction.classifyImage(os.path.join(execution_path, "data-
images/estadio2.jpg"), result_count=5 )

for eachPrediction, eachProbability in zip(predictions, probabilities):

    print(eachPrediction , " : " , eachProbability)
```

```
WARNING:tensorflow:6 out of the last 6 calls to <function Model.make_predict_function.<locals>.predict_function at
Downloading data from https://storage.googleapis.com/download.tensorflow.org/data/imagenet_class_index.json
40960/35363 [==================================] - 0s 0us/step
49152/35363 [==========================================] - 0s 0us/step
monitor  :  19.894567131996155
prayer_rug  :  10.945137590169907
theater_curtain  :  9.362854808568954
scoreboard  :  7.561834156513214
mosque  :  6.818679720163345
```

*Ilustración 19: Resultado de ejecución ResNet50.*

Se puede observar que compara el terreno de juego con una alfombra para rezar, no es el resultado deseado, pero se puede entender que el césped artificial es una gran alfombra plana. Además, es capaz de detectar otros objetos como un marcador o un monitor. Se procede a comparar los resultados con otra imagen que se muestra a continuación:

*Ilustración 20: Terreno de juego.*

```
maze  :  99.20275807380676
tennis_ball  :  0.086994533194229
racket  :  0.050797645235434175
balloon  :  0.0499393732752651
scoreboard  :  0.03912507090717554
```

*Ilustración 21: Resultado de ejecución ResNet50 para la imagen 2.*

En este caso, el algoritmo asigna casi toda la probabilidad de detección a un laberinto basándose principalmente en el color verde, sin embargo, mantiene la detección de un scoreboard el cual no se aprecia en la imagen. Cabe destacar que no existe un resultado que se trate de un "terreno de juego".

El siguiente código hace referencia al modelo DenseNet121. Se ha obtenido un tiempo de ejecución de 5s utilizando la primera imagen. Result_cont indica el número de los parámetros devueltos con mayor porcentaje de detección.

```
from imageai.Classification import ImageClassification

import os

execution_path = os.getcwd()


prediction = ImageClassification()

prediction.setModelTypeAsDenseNet121()

prediction.setModelPath(os.path.join(execution_path, "DenseNet-BC-121-32.h5"))

prediction.loadModel()


predictions, probabilities =
prediction.classifyImage(os.path.join(execution_path, "data-
images/estadio2.jpg"), result_count=5 )

for eachPrediction, eachProbability in zip(predictions, probabilities):

    print(eachPrediction , " : " , eachProbability)
```

```
aircraft_carrier  :  39.39239680767059
dam  :  10.03769189119339
prayer_rug  :  8.571392297744751
palace  :  4.591831192374229
dome  :  3.243429586291313
```

*Ilustración 22: Resultado de ejecución DenseNet121 para estadio2.*

```
maze  :  88.11689019203186
tennis_ball  :  3.771398216485977
scoreboard  :  2.0240701735019684
racket  :  1.9683513790369034
sundial  :  0.920239370316267
```

*Ilustración 23: Resultado de ejecución DenseNet121 para la segunda imagen.*

El modelo detecta un portaviones o una presa hidráulica entre otros, es evidente que no está optimizado para detectar estructuras como campos de fútbol. De la segunda imagen se obtienen unos resultados parecidos al primer método utilizado.

Como tercer ejemplo utilizamos el modelo MobileNetV2. Se ha obtenido un tiempo de ejecución de 3s. El código se muestra justo debajo:

```
from imageai.Classification import ImageClassification
import os
execution_path = os.getcwd()

prediction = ImageClassification()
prediction.setModelTypeAsMobileNetV2()
prediction.setModelPath(os.path.join(execution_path, "mobilenet_v2.h5"))
prediction.loadModel()

predictions, probabilities =
prediction.classifyImage(os.path.join(execution_path, "data-
images/estadio2.jpg"), result_count=5 )
for eachPrediction, eachProbability in zip(predictions, probabilities):
    print(eachPrediction , " : " , eachProbability)
```

```
palace  :  9.516901522874832
steel_arch_bridge  :  8.947718143463135
crane  :  7.617664337158203
pier  :  5.1558718085289
scoreboard  :  4.099031537771225
```

*Ilustración 24: Resultado de ejecución MobileNetV2.*

En este caso podemos observar que el modelo es algo más eficiente ya que es capaz de detectar estructuras de metal que son características que forman parte de estadios de fútbol.

Para finalizar se trata con el modelo InceptionV3. Se ha obtenido un tiempo de ejecución de 4s. El código empleado es el siguiente:

```
from imageai.Classification import ImageClassification
import os
execution_path = os.getcwd()

prediction = ImageClassification()
prediction.setModelTypeAsInceptionV3()
prediction.setModelPath(os.path.join(execution_path,
"inception_v3_weights_tf_dim_ordering_tf_kernels.h5"))
prediction.loadModel()

predictions, probabilities =
prediction.classifyImage(os.path.join(execution_path, "data-
images/estadio4.jpg"), result_count=5 )
for eachPrediction, eachProbability in zip(predictions, probabilities):
    print(eachPrediction , " : " , eachProbability)
```

```
stage  :  91.05969071388245
theater_curtain  :  4.161407425999641
planetarium  :  0.9502912871539593
pier  :  0.699000945314765
bannister  :  0.4031045828014612
```

*Ilustración 25: Resultado de ejecución InceptionV3.*

En este caso se detecta un escenario como resultado principal el cual se refiere a un terreno plano, amplio, el cual puede ser un resultado coherente para la imagen.

En la siguiente tabla se encuentra la comparativa de todos los modelos habiendo aplicado diferentes velocidades de predicción. El resultado varía en función del método y la velocidad elegida, siendo más precisos los tiempos lentos. El modelo MobileNetV2 es un modelo de detección rápida y precisión moderada, además es el archivo que menos peso tiene y es utilizado

en imágenes que no requieren de una gran exactitud y precisión ya que es un algoritmo muy rápido. ResNet50 es un modelo de predicción rápida y alta precisión. InceptionV3 es un modelo de detección lenta, pero precisión alta, mayor que la de ResNet50 y finalmente tenemos DenseNet121 que se trata del modelo más lento, pero con mayor precisión.

ImageAI ofrece como en el caso de clasificación de imágenes diferentes velocidades de lectura y procesado, siendo "normal" la predefinida y estándar y otras velocidades como "fast", "fastest" o "flash" que se le deben de pasar como argumento a la llamada prediction.loadModel (classification_speed) y puede reducir la velocidad de detención entre un 20% - 60%.

| | Velocidad | | | | |
|---|---|---|---|---|---|
| **Modelo** | **normal** | **fast** | **fastest** | **flash** | **Tamaño** |
| **ResNet50** | **3s** | **3s** | **3s** | **2s** | **98mb** |
| **DenseNet121** | **5s** | **5s** | **5s** | **5s** | **31.6mb** |
| **MobileNetV2** | **4s** | **2s** | **2s** | **2s** | **4.82mb** |
| **InceptionV3** | **4s** | **4s** | **3s** | **3s** | **91.6mb** |

*Tabla 2: Tabla comparativa de datos para predicción de imágenes.*

Como conclusión para esta sección se puede decir que los modelos MobileNetV2 e InceptionV3 son los que devuelven unos resultados más concretos respecto de la imagen que se ha pasado como argumento de entrada. Los modelos tratan de identificar los objetos de la imagen con los existentes en su base interna de datos. En algunos casos se obtienen resultados razonables y en otros unos resultados completamente indeseados. Las diferentes velocidades devuelven unos resultados muy parecidos para el mismo tipo de modelo en los que la probabilidad de detección apenas varía, aunque para velocidades mayores DenseNet o InceptionV3 proporciona mayor precisión. Se procede a aplicar la predicción de objetos sobre una imagen en la que se muestra el autobús de la selección española y analizar los resultados.


*Ilustración 26: Autobús de la selección española*

```
Downloading data from https://storage.googleapis.com/download.tensorflow.org/data/imagenet_class_index.json
40960/35363 [==============================] - 0s 0us/step
49152/35363 [==============================] - 0s 0us/step
tow_truck   :  98.48928451538086
fire_engine   :  1.2410936877131462
garbage_truck   :  0.2489655278623104
trailer_truck   :  0.0206590979360044
mobile_home   :  6.002274233196658e-07
```

*Ilustración 27: Resultado predicción InceptionV3 en velocidad 'fastest'*

```
ambulance   :  74.24069046974182
fire_engine   :  24.11244809627533
trailer_truck   :  1.251631136983633
moving_van   :  0.1721223467029631
tow_truck   :  0.13606963912025094
```

*Ilustración 28: Resultado DenseNet en velocidad 'fastest'*

```
passenger_car   :  39.39520716667175
school_bus   :  24.116992950439453
trolleybus   :  20.47085165977478
minibus   :  4.694885015487671
trailer_truck   :  2.7369460090994835
```

*Ilustración 29: Resultado DenseNet en velocidad 'normal'*

El modelo DenseNet detecta un vehículo de transporte como principal resultado mientras que el mismo modelo con una velocidad de detección mayor indica que se trata de una ambulancia. En este caso la velocidad de detección es un factor determinante. Por otro lado, el modelo InceptionV3 indica que se trata de otro tipo de vehículo de transporte, además ofrece un

resultado bastante parecido para diferentes velocidades de detección. La precisión para las velocidades en modo ‘normal’ son muy parecidas mientras que si se aplica otro tipo de velocidades los resultados varían en función al modelo usado.

```
passenger_car  :  39.39520716667175
school_bus  :  24.116992950439453
trolleybus  :  20.47085165977478
minibus  :  4.694885015487671
trailer_truck  :  2.7369460090994835
```

*Ilustración 30: Resultado InceptionV3 en velocidad 'normal'*

# Capítulo tercero: Desarrollo de la aplicación en vídeos

## 3.1 Desarrollo tradicional en videos.

El vídeo (Wikipedia, la enciclopedia libre, 2021) es la tecnología de grabación, procesamiento, almacenamiento, transmisión de imágenes y reconstrucción por medios electrónicos digitales o analógicos de una secuencia de imágenes que representan escenas en movimiento. El movimiento aparece al considerar una secuencia de imágenes. Un video consta de varias imágenes por segundo que es técnicamente conocido como fotogramas por segundo (fps). Los estándares de origen europeo PAL y SECAM especifican 25 cuadros por segundo, mientras que NTSC específica 29,97 cuadros por segundo. El cine es más lento con una velocidad de 24 cuadros por segundo. Es el mínimo necesario para que exista una sensación de vídeo fluido.

El movimiento de objetos y o cámara producen cambios en la escena, pero no todos los cambios proceden del movimiento ya que existe el ruido, la iluminación, los reflejos y las sombras.

El movimiento ocurre en 3D, aunque las imágenes son en 2D. Existen varios problemas a la hora de captar el movimiento que se han de tener en cuenta, como son la compresión del vídeo que es un factor determinante para reproducir con una calidad decente, cálculo de profundidades o extracción objeto/fondo en varías imágenes. El movimiento en imágenes tiene aplicaciones en diversos sectores: Navegación y exploración, restauración de películas, compresión de vídeo o realidad aumentada.

La diferencia entre dos imágenes monocromas obtenidas de una sentencia proporciona cierta información sobre el movimiento real. Esta técnica es limitada pero válida en ocasiones.

- Movimiento no tiene componente z.
- Los cambios de iluminación son pequeños.
- Hay pocos objetos con movimiento apreciable.
- La diferencia resultante entre las imágenes I2 - I1 no es una imagen.
- En la imagen resultante, se puede visualizar si se escalan los valores adecuadamente.
- En tal caso:

    - Las zonas negras corresponden a espacios que han pasado de mayor intensidad a menor.
    - Con las zonas blancas ocurre, al contrario.
    - En las zonas grises no ha habido cambio.

*Ilustración 31: Diferencia de imágenes.*

El flujo óptico (Wikipedia, la enciclopedia libre, 2019) es el patrón del movimiento aparente de los objetos, superficies y bordes en una escena causado por el movimiento relativo entre un observador (ojo, cámara, sensor) y la escena.

- Movimiento aparente de las partes de una escena.
- El término se originó en el ámbito de la psicología (J.J. Gibson, 1950).
- Guarda relación con el movimiento real de objetos, pero no es lo mismo.

El movimiento puede percibirse mal si no se ve completamente como por ejemplo ocurre con la ilusión óptica del cartel de un barbero. El poste de barbero crea una ilusión visual, en la cual las rayas parecen que van ascendiendo o descendiendo cuando realmente giran alrededor constantemente.

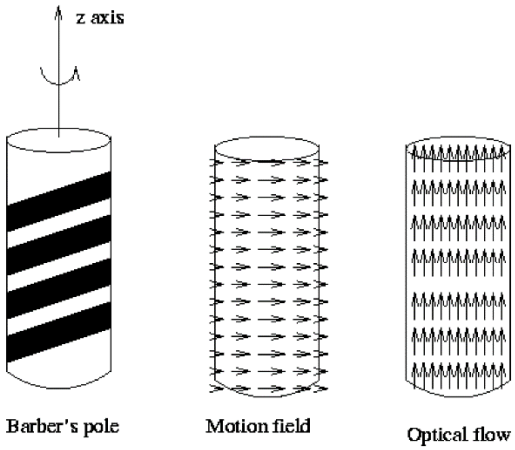


*Ilustración 32: Ilusión óptica cartel de barbero.*

Dados dos instantes t1, t2 siendo t2 = t1 + t se desea calcular el campo aparente de velocidades al pasar de I1 a I2.

Sea una imagen I, se define:

- $I\ (x,\ y,\ t)$ nivel de gris en (x; y) el instante t.
- $I\ (x + x,\ y + y,\ t + t)$ nivel de gris del punto trasladado.

Suponiendo que la iluminación no cambia:

- $I\ (x, y, t) = I\ (x + x,\ y + y,\ t + t)$

Para cada (x; y) se busca la velocidad de cambio $v = [vx, vy]'$[12]. Deberá cumplirse que:

$$I1\ (x, y) = I2\ (x + vxt, y + vyt)$$

Gracias a todas estas ecuaciones (Arahal, 2018) se desarrolla la llamada ecuación fundamental del flujo óptico que se describe de la siguiente manera.

$$I_x \cdot v_x + I_y \cdot v_y = -I_t$$

$$\overline{\nabla I}^{\top} \cdot \bar{v} = -I_t$$

*Ilustración 33: Ecuación del flujo óptico.*

$-I_t$ es la intensidad en el instante t. El vector $\nabla I$ es el gradiente que indica la dirección máxima de la intensidad.

Se mencionan dos métodos que sirven para el cálculo de flujo óptico que han sido de vital importancia para la evolución de la detección de movimiento.

El primer método llamado Método de *Lucas-Kanade*:

En visión artificial, el método Lucas–Kanade (Wikipedia. la enciclopedia libre , 2020) es un método diferencial ampliamente usado para estimar el flujo óptico desarrollado por Bruce D. Lucas y Takeo Kanade. Asume que el flujo es esencialmente constante en la vecindad de un píxel en consideración, y resuelve las ecuaciones básicas de flujo óptico para todos los píxeles vecinos, aplicando el criterio de cuadrados mínimos.

Al combinar información de varios píxeles cercanos, el método Lucas–Kanade frecuentemente resuelve la ambigüedad inherente de la ecuación de flujo óptico. También es más robusto frente al ruido en la imagen que otros métodos locales que se concentran en un punto. Por otro lado, siendo un método puramente local, no provee información de flujo en el interior de regiones uniformes en la imagen, como sí lo hace el método global de Horn-Schunck.

- En una zona homogénea en ambas imágenes $(Ix = Iy = It = 0)$ no se detecta movimiento.
- En un borde $\Delta I \neq 0$ en la dirección perpendicular al borde no se detecta correctamente cualquier movimiento.
- En una esquina $\Delta I \neq 0$ en dos direcciones se detecta correctamente cualquier movimiento.

El segundo método llamado Método de Horn-Schunck:

- La hipótesis es que el movimiento (al pasar el tiempo) es suave en toda la escena.
- Por ello de las infinitas soluciones se prefieren las que corresponden a velocidades con distribución suave.

[12] 'significa matriz transpuesta

El método de Horn-Schunck (Wikipedia, la enciclopedia libre, 2019) sirve para estimar el flujo óptico. Es un método global que introduce una restricción global de suavidad para resolver el problema de apertura.

Ventaja: proporciona el flujo óptico en regiones donde L-K no funciona (menos sensible al problema de la apertura).

Inconveniente: es más sensible al ruido.

## 3.2 Desarrollo de la herramienta ImageAI para detección en videos.

ImageAI (Borda, 2019) proporciona métodos convenientes, flexibles y potentes para realizar la detección de objetos en videos. La clase de detección de objetos de video proporcionada solo es compatible con RetinaNet, YOLOv3 y TinyYOLOv3. Esta versión de ImageAI proporciona funciones de detección de objetos de vídeo de calidad comercial, que incluyen, entre otras, entradas de dispositivo / cámara IP, por cuadro, por segundo, por minuto y análisis de vídeo completo para almacenar en bases de datos y / o visualizaciones en tiempo real.

La detección de objetos de video es una tarea de computación intensiva, se recomienda realizar el experimento en una computadora con una GPU NVIDIA y la versión de GPU de Tensorflow instalada. En este caso se usan las GPU de Google Colab (Na8, 2019).

A continuación, se expone el código para el uso de la herramienta y explicaremos paso a paso lo que se acontece:

```
from imageai.Detection import VideoObjectDetection

import os

execution_path = os.getcwd()


detector = VideoObjectDetection()

detector.setModelTypeAsTinyYOLOv3()

detector.setModelPath( os.path.join(execution_path , "yolo-tiny.h5"))

detector.loadModel()


video_path =
detector.detectObjectsFromVideo(input_file_path=os.path.join(execution_path,
"data-videos/Goliniestacorto.mp4"),

                                output_file_path=os.path.join(execution_path,
"goliniesta-yolotiny")

                                , frames_per_second=10, log_progress=True)

print(video_path)
```

Se importa la clase VideoObjectDetection de la librería en la primera línea, el módulo a ciertas funciones del sistema operativo en la segunda línea y la ruta a la carpeta donde se ejecuta el archivo. Se crea una nueva instancia de la clase VideoObjectDetection, se configura el tipo de modelo en este caso con TinyYoloV3, se configura la ruta del modelo al archivo del modelo *yolo-tiny.h5* este archivo se encuentra en la carpeta del directorio principal. Se ejecuta la función detectObjectsFromVideo (). la ruta al nuevo video (sin la extensión, guarda un video .avi por defecto), el número de cuadros por segundo (fps) que se desea y la opción de registrar el progreso de la detección en la consola. La función devuelve la ruta al video guardado que contiene cuadros y probabilidades porcentuales representadas en el video.

Se han realizado diversas pruebas con diferentes archivos y a continuación se encuentran los enlaces de los vídeos procesados que se encuentran en el directorio de drive.

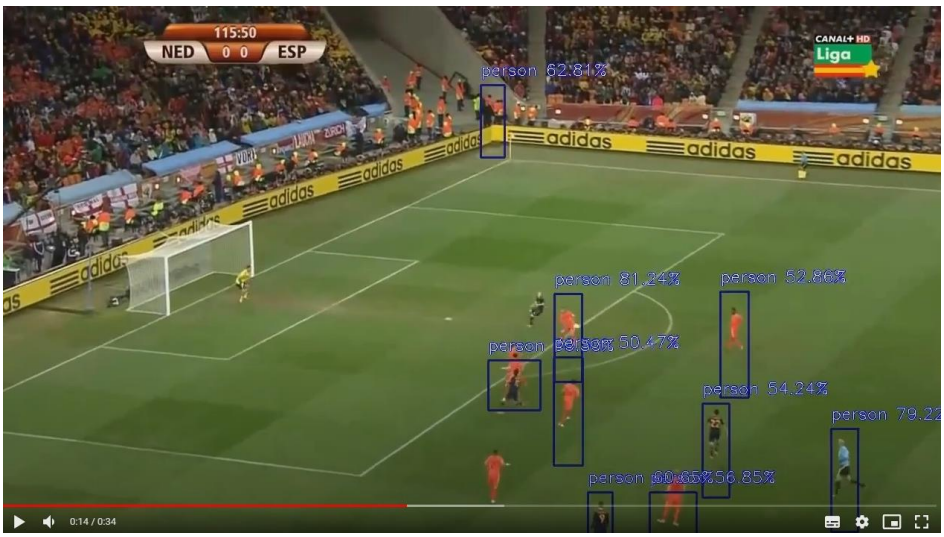


*Ilustración 34: Captura del vídeo procesado.*

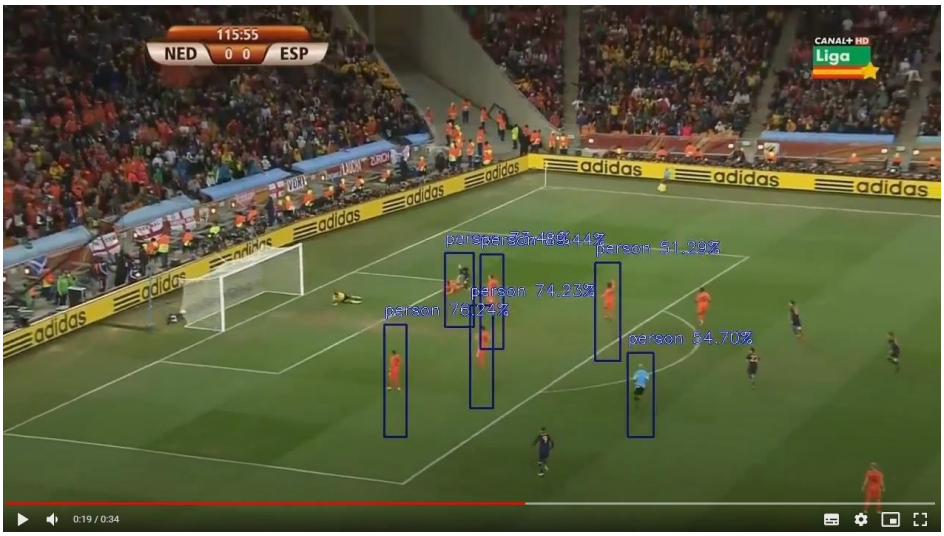


*Ilustración 35: Captura del vídeo procesado.*

En la imagen anterior se puede apreciar como en ese momento exacto el algoritmo no detecta a todos los jugadores en el campo, esto puede deberse a varios factores como son la posición del jugador, la profundidad de campo a la que se encuentre, la calidad del vídeo y el modelo utilizado. Todas estas características proporcionan una probabilidad de detección del objeto que el algoritmo detectará y procesará si supera el umbral dado. Las imágenes a continuación si detectan a todos los jugadores.


*Ilustración 36: Captura del vídeo procesado.*

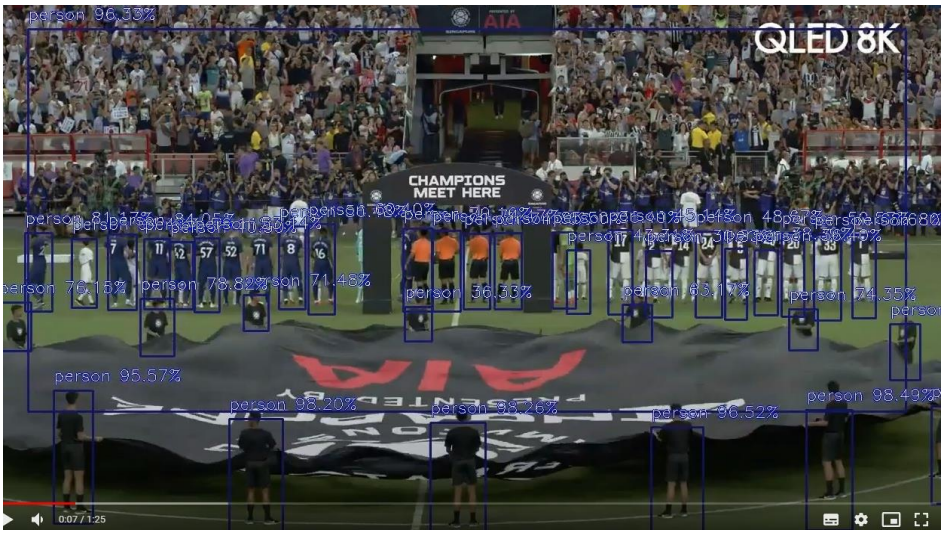


*Ilustración 37: Captura del vídeo procesado.*


*Ilustración 38: Captura del vídeo procesado.*

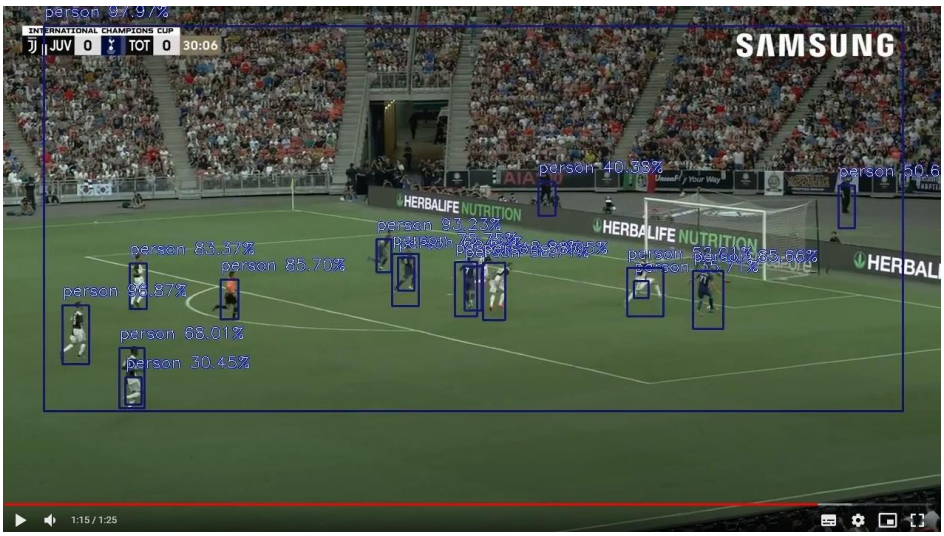


*Ilustración 39: Captura del vídeo procesado.*

Como se puede observar, este vídeo está en resolución 8k, lo que ofrece un resultado muy favorable. Además, en este caso se aplicaron 10fps lo que implica una ralentización del vídeo y una gran mejora de detección. Esto provoca que la velocidad del vídeo sea mayor a consta de un resultado mejor. Todas estas imágenes sirven de ejemplo para demostrar que la variación de los parámetros comentados influya en la precisión de predicción de objetos en el vídeo. El óptimo depende del resultado deseado, por ejemplo, si se desea detectar todos los jugadores lo ideal es utilizar una velocidad de detección baja que implica un tiempo de ejecución mayor.

```
Processing Frame :  395
Processing Frame :  396
Processing Frame :  397
Processing Frame :  398
Processing Frame :  399
Processing Frame :  400
Processing Frame :  401
Processing Frame :  402
Processing Frame :  403
Processing Frame :  404
Processing Frame :  405
Processing Frame :  406
/content/drive/My Drive/TFG/ImageAI-master/golfinal_yo.avi
```

*Ilustración 40: Ejemplo de ejecución.*

ImageAI (Olafenwa & Olafenwa, GitHub, s.f.) permite la detección de video en vivo con soporte para entradas de cámara. Con la función VideoCapture () de OpenCV, puede cargar transmisiones de video en vivo desde la cámara de un dispositivo, cámaras conectadas por cable o cámaras IP, y analizarlas en las funciones detectObjectsFromVideo () y detectCustomObjectsFromVideo () de ImageAI. Todas las funciones están disponibles para detectar objetos en la transmisión de video en vivo de una cámara.

En este nuevo código se define una instancia de OpenCV VideoCapture y se carga la cámara del dispositivo predeterminada. Se analiza la cámara que se define en el parámetro camera_input que reemplaza el input_file_path que se usa para el archivo de video.

- Parámetro input_file_path (obligatorio si no se configuró camera_input): Ruta al archivo de video que desea detectar.

- Parámetro output_file_path (obligatorio si no configuró save_detected_video = False): Se refiere a la ruta en la que se guardará el vídeo detectado. Esta función guarda el formato de video .avi. de forma predeterminada

- Parámetro frames_per_second (opcional, recomendado): este parámetro permite establecer los fotogramas por segundo que desee para el video detectado que se guardará. El valor predeterminado es 20.

- Parámetro log_progress (opcional): la configuración de este parámetro en True muestra el progreso del video o la transmisión en vivo a medida que se detecta en la CLI. Informará cada fotograma detectado a medida que avanza. El valor predeterminado es falso.

- Parámetro return_detected_frame (opcional): este parámetro permite devolver el fotograma detectado como una matriz Numpy en cada fotograma, segundo y minuto del video detectado. La matriz Numpy devuelta se analizará en las respectivas funciones per_frame_function, per_second_function y per_minute_function.
- Parámetro camera_input (opcional): este parámetro se puede configurar en reemplazo de input_file_path si desea detectar objetos en la transmisión en vivo de una cámara. Todo lo que necesita es cargar la cámara con la función VideoCapture () de OpenCV y analizar el objeto en este parámetro. El siguiente ejemplo de código es una muestra de cómo se aplica este parámetro.

```
from imageai.Detection import VideoObjectDetection
import os
import cv2

execution_path = os.getcwd()
camera = cv2.VideoCapture(0)

detector = VideoObjectDetection()
detector.setModelTypeAsTinyYOLOv3()
detector.setModelPath( os.path.join(execution_path , "yolo-tiny.h5"))
detector.loadModel()



video_path = detector.detectObjectsFromVideo(
                camera_input=camera,
                output_file_path=os.path.join(execution_path,
"camera_detected_video_prueba"),
                frames_per_second=20, log_progress=True,
minimum_percentage_probability=40, detection_timeout=120)
```

ImageAI proporciona análisis de video de calidad comercial para detección de objetos de video para entradas de archivos de video y entradas de cámara. Permite obtener información detallada. Se puede visualizar en tiempo real y almacenarse en una base de datos NoSQL para su revisión o análisis en el futuro.

- Parámetro minimum_percentage_probability (opcional): este parámetro se utiliza para determinar la integridad de los resultados de la detección. Reducir el valor muestra más objetos mientras que aumentar el valor asegura que se detecten los objetos con la mayor precisión. El valor predeterminado es 50.

- Parámetro display_percentage_probability (opcional): este parámetro se puede utilizar para ocultar el porcentaje de probabilidad de cada objeto detectado en el video detectado si se establece como False. El valor predeterminado es True.

- Parámetro display_object_name (opcional): este parámetro se puede usar para ocultar el nombre de cada objeto detectado en el video detectado si se establece como False. True es su valor predeterminado.

- Parámetro save_detected_video (opcional): Este parámetro puede usarse para guardar el video detectado o para no guardarlo. Está establecido en True de forma predeterminada.

- Parámetro per_frame_function (opcional): este parámetro permite analizar el nombre de una función que se defina. Para cada fotograma del video que se detecte la función actúa y extrae los datos que los parámetros han definido. Los datos devueltos se pueden visualizar o guardar en una base de datos NoSQL para su procesamiento y visualización futuros.

detectObjectsFromVideo () y detectCustomObjectsFromVideo () se ejecutan para cada fotograma, segundo y / o minuto del video detectado. Las funciones reciben datos analíticos sin procesar completos sobre el índice del cuadro / segundo / minuto, los objetos detectados (nombre, porcentaje de probabilidad y puntos de cuadro), número de instancias de cada objeto único detectado y número promedio de ocurrencia de cada uno.

Para obtener el análisis de vídeo: Especificar una función, indicar los parámetros y analizar el nombre de la función en los parámetros per_frame_function, per_second_function, per_minute_function y en la función de detección.

El código de las funciones se define a continuación:

```
def forFrame(frame_number, output_array, output_count):
    print("FOR FRAME " , frame_number)
    print("Output for each object : ", output_array)
    print("Output count for unique objects : ", output_count)
    print("------------END OF A FRAME --------------")

def forSeconds(second_number, output_arrays, count_arrays,
average_output_count):
    print("SECOND : ", second_number)
    print("Array for the outputs of each frame ", output_arrays)
    print("Array for output count for unique objects in each frame : ",
count_arrays)
    print("Output average count for unique objects in the last second: ",
average_output_count)
    print("------------END OF A SECOND --------------")

def forMinute(minute_number, output_arrays, count_arrays,
average_output_count):
    print("MINUTE : ", minute_number)
    print("Array for the outputs of each frame ", output_arrays)
    print("Array for output count for unique objects in each frame : ",
count_arrays)
    print("Output average count for unique objects in the last minute: ",
average_output_count)
    print("------------END OF A MINUTE --------------")

video detector = Video Object Detection()
video_detector.setModelTypeAsYOLOv3()
video_detector.setModelPath(os.path.join(execution_path, "yolo.h5"))
video_detector.loadModel()

video_detector.detectObjectsFromVideo(
    input_file_path=os.path.join(execution_path, "data-
videos/Goliniestacorto.mp4"),
    output_file_path=os.path.join(execution_path, "Gol_iniesta_yolo"),
```

```
    frames_per_second=10,
    per_second_function=forSeconds,
    per frame function=forFrame,
    per_minute_function=forMinute,
    minimum_percentage_probability=30)
```

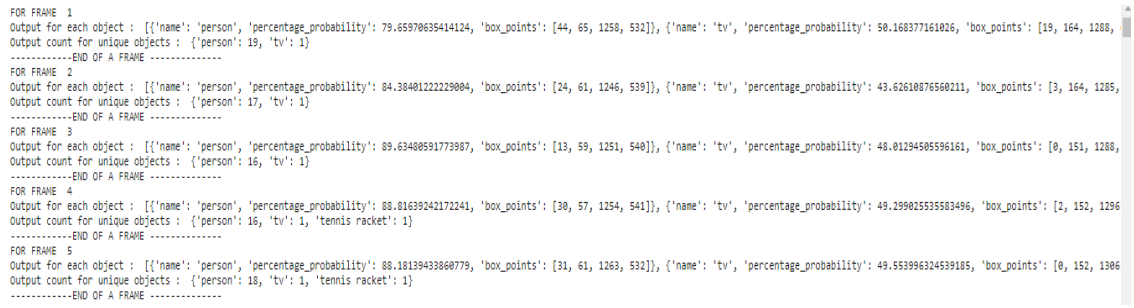


*Ilustración 41: Ejemplo de ejecución.*

```
FOR FRAME  1

Output for each object :  [{'name': 'person', 'percentage_probability':
79.65970635414124, 'box_points': [44, 65, 1258, 532]}, {'name': 'tv',
'percentage_probability': 50.168377161026, 'box_points': [19, 164, 1288,
633]}, {'name': 'person', 'percentage_probability': 97.50775694847107,
'box_points':   [269,   236,   291,   281]},   {'name':   'person',
'percentage_probability': 93.07785034179688, 'box_points': [745, 246,
767,     286]},     {'name':     'person',     'percentage_probability':
30.563098192214966,  'box_points':  [418,  275,  428,  285]},  {'name':
'person',  'percentage_probability':  80.33425211906433,  'box_points':
[583,  257,  618,  297]},  {'name':  'person',  'percentage_probability':
63.094526529312134,  'box_points':  [675,  273,  697,  313]},  {'name':
'person',  'percentage_probability':  95.07280588150024,  'box_points':
[145,  264,  172,  321]},  {'name':  'person',  'percentage_probability':
97.77753949165344,  'box_points':  [458,  285,  483,  329]},  {'name':
'person',  'percentage_probability':  32.994601130485535,  'box_points':
[685,  297,  704,  318]},  {'name':  'person',  'percentage_probability':
84.73087549209595,  'box_points':  [331,  293,  350,  347]},  {'name':
'person',  'percentage_probability':  86.54356002807617,  'box_points':
[654,  314,  680,  367]},  {'name':  'person',  'percentage_probability':
80.85147142410278,  'box_points':  [602,  360,  625,  414]},  {'name':
'person',  'percentage_probability':  63.525670766830444,  'box_points':
[1160,  372,  1187,  424]},  {'name':  'person',  'percentage_probability':
66.75986647605896,  'box_points':  [337,  366,  362,  420]},  {'name':
'person',  'percentage_probability':  66.31612777709961,  'box_points':
[742,  370,  765,  432]},  {'name':  'person',  'percentage_probability':
51.4104425907135,  'box_points':  [802,  388,  828,  431]},  {'name':
'person',  'percentage_probability':  42.097076773643494,  'box_points':
[842,  491,  866,  527]},  {'name':  'person',  'percentage_probability':
87.55117058753967,  'box_points':  [835,  475,  877,  541]},  {'name':
'person',  'percentage_probability':  96.5309202671051,  'box_points':
[635, 632, 680, 713]}]

Output count for unique objects :  {'person': 19, 'tv': 1}
```

```
------------END OF A FRAME --------------
```

La función **per_frame_function**, se ejecutará después de que se procese cada fotograma de vídeo y se analizarán los siguientes parámetros:

Índice de fotogramas: número de posición del fotograma dentro del video (por ejemplo, 1 para el primer fotograma y 30 para el trigésimo fotograma).

Matriz de salida: esta es una matriz de diccionarios. Cada diccionario corresponde a cada objeto detectado en la imagen y contiene los valores de "nombre", "porcentaje de probabilidad" y "puntos de cuadro" (x1, y1, x2, y2) del objeto.

Recuento de salida: este es un diccionario que tiene el nombre de cada objeto único detectado como sus claves y el número de instancias de los objetos detectados como valores.

Esto se repite para todos los frames del vídeo. El resultado de ejecución depende de la cantidad de objetos detectados en cada frame.

Para cualquier función que analice en la función **per_second_function**, se obtendrán:

Segundo índice: Es el número de posición del segundo dentro del video.

Matriz de salida: Es una matriz de matrices que correspondiente con el último fotograma equivalente en el último segundo del video. Cada matriz contiene diccionarios. Cada diccionario corresponde a cada objeto detectado en la imagen y contiene los valores de "nombre", "porcentaje de probabilidad" y "puntos de cuadro" (x1, y1, x2, y2) del objeto. Es decir es una matriz que contiene otra matriz por cada objeto detectado correspondiente al último frame detectado.

Contar matrices: esta es una matriz de diccionarios. Cada diccionario y su posición (índice de matriz + 1) corresponde al fotograma equivalente en el último segundo del video.

Recuento de salida promedio: este es un diccionario que tiene el nombre de cada objeto único detectado en el último segundo como sus claves y el número promedio de instancias de los objetos detectados en el número de fotogramas como valores.

Los cautro parámetros que se devuelven por cada segundo del video procesado son los mismos parámetros que se devolverán por cada minuto. La diferencia es que el índice devuelto corresponde al índice de minutos, output_arrays es una matriz que contiene el número de FPS * 60 matrices, y count_arrays es una matriz que contiene el número de FPS * 60 diccionarios y el average_output_count es un diccionario que cubre todos los objetos detectados en todos los fotogramas contenidos en el último minuto.

Los resultados nos permiten obtener un análisis completo de todo el video procesado. Necesitamos definir una función como la función forSecond o forMinute y establecer el parámetro video_complete_function en su función .detectObjectsFromVideo () o .detectCustomObjectsFromVideo (). Se devolverán los mismos valores para per_second-function y per_minute_function. La diferencia es que no se devolverá ningún índice y se devolverán los otros tres valores, y los tres valores cubrirán todos los fotogramas del vídeo.

Ejemplo de ejecución usando la función forFrame:

```
def forFrame(frame_number, output_array, output_count, detected_frame):
    print("FOR FRAME " , frame_number)
    print("Output for each object : ", output_array)
    print("Output count for unique objects : ", output_count)
    print("Returned Object is : ", type(detected_frame))
    print("------------END OF A FRAME --------------")


video_detector.detectObjectsFromVideo(
    input_file_path=os.path.join(execution_path, "data-
videos/8K_Football_corto.mp4"),
    output_file_path=os.path.join(execution_path, "8K_Football_prueba.mp4"),
    frames_per_second=10,
    per_frame_function=forFrame,
    minimum_percentage_probability=30,
    return_detected_frame=True
)
```

```
FOR FRAME  1
Output for each object :  [{'name': 'person', 'percentage_probability': 48.79430830478668, 'box_points': [120, 40, 1171, 602]}, {'name': 'person', 'percentage_pr
Output count for unique objects :  {'person': 17}
Returned Object is :  <class 'numpy.ndarray'>
------------END OF A FRAME --------------
```

*Ilustración 42: Ejemplo de ejecución.*

Para concluir esta sección, de igual manera que existen diferentes velocidades de procesado en las clasificaciones de imagen, se definen cinco velocidades distintas de procesado en vídeo que son: 'normal', 'fast', 'faster', 'fastest' y 'flash' que se precisan tal como se indica en el siguiente comando:

```
detector.loadModel(detection_speed="fast")
```

| | Velocidad | | | | |
|---|---|---|---|---|---|
| **Modelo** | **normal** | **fast** | **fastest** | **flash** | **Tamaño** |
| **YOLOv3** | **2h 3m 44s** | **1h 7m 30s** | **11m 18s** | **7m 21s** | **237mb** |
| **TinyYOLOv3** | **1h 27m 6s** | **42m 32s** | **13m 36s** | **3m 1s** | **34mb** |
| **ResNet** | **2h 15m 31s** | **1h 50m 19s** | **1h 24m 15s** | **1h 03m 54s** | **145mb** |

*Tabla 3:Tabla comparativa de datos para detección en vídeos.*

Esta tabla muestra una recolección de los tiempos de ejecución dados para los diferentes modelos utilizados y las diferentes velocidades de procesamiento aplicados. La velocidad "normal" ofrece una precisión más exacta detectando una mayor cantidad de objetos mientras que las otras opciones de velocidad de detección ofrecen una precisión peor a consta un tiempo de ejecución menor. A continuación, se muestran diferentes resultados del primer frame aplicando variaciones en los parámetros de entrada. Todos los resultados se corresponden con el mismo archivo utilizado en todos los casos del que se obtienen hasta 1022 frames.

En el primer caso el modelo utilizado es YOLOv3 con 10fps y una velocidad de detección "normal".

```
FOR FRAME  1
Output for each object :  [{'name': 'person',
'percentage_probability': 79.65970635414124, 'box_points': [44, 65,
1258, 532]}, {'name': 'tv', 'percentage_probability':
50.168377161026, 'box_points': [19, 164, 1288, 633]}, {'name':
'person', 'percentage_probability': 97.50775694847107,
'box_points': [269, 236, 291, 281]}, {'name': 'person',
'percentage_probability': 93.07785034179688, 'box_points': [745,
246, 767, 286]}, {'name': 'person', 'percentage_probability':
30.563098192214966, 'box_points': [418, 275, 428, 285]}, {'name':
'person', 'percentage_probability': 80.33425211906433,
'box_points': [583, 257, 618, 297]}, {'name': 'person',
'percentage_probability': 63.094526529312134, 'box_points': [675,
273, 697, 313]}, {'name': 'person', 'percentage_probability':
95.07280588150024, 'box_points': [145, 264, 172, 321]}, {'name':
'person', 'percentage_probability': 97.77753949165344,
'box_points': [458, 285, 483, 329]}, {'name': 'person',
'percentage_probability': 32.994601130485535, 'box_points': [685,
297, 704, 318]}, {'name': 'person', 'percentage_probability':
84.73087549209595, 'box_points': [331, 293, 350, 347]}, {'name':
'person', 'percentage_probability': 86.54356002807617,
'box_points': [654, 314, 680, 367]}, {'name': 'person',
'percentage_probability': 80.85147142410278, 'box_points': [602,
360, 625, 414]}, {'name': 'person', 'percentage_probability':
```

```
63.525670766830444, 'box_points': [1160, 372, 1187, 424]}, {'name':
'person', 'percentage_probability': 66.75986647605896,
'box_points': [337, 366, 362, 420]}, {'name': 'person',
'percentage_probability': 66.31612777709961, 'box_points': [742,
370, 765, 432]}, {'name': 'person', 'percentage_probability':
51.4104425907135, 'box_points': [802, 388, 828, 431]}, {'name':
'person', 'percentage_probability': 42.097076773643494,
'box_points': [842, 491, 866, 527]}, {'name': 'person',
'percentage_probability': 87.55117058753967, 'box_points': [835,
475, 877, 541]}, {'name': 'person', 'percentage_probability':
96.53092026710 51, 'box_points': [635, 632, 680, 713]}]
Output count for unique objects :  {'person': 19, 'tv': 1}
------------END OF A FRAME --------------
```

En el segundo caso se utiliza el modelo YOLOv3 con 20fps y una velocidad de detección “flash”.

```
FOR FRAME  1

Output for each object :  [{'name': 'person',
'percentage_probability': 32.531946897506714, 'box_points': [602,
319, 622, 351]}, {'name': 'person', 'percentage_probability':
33.63240361213684, 'box_points': [672, 358, 694, 392]}, {'name':
'person', 'percentage_probability': 42.198702692985535,
'box_points': [755, 433, 786, 476]}]

Output count for unique objects :  {'person': 3}

------------END OF A FRAME --------------
```

Se puede apreciar la diferencia en la cantidad de objetos detectados, la precisión del primer ejemplo es mucho mejor a consta de un tiempo de ejecución mucho mayor como se ha comentado anteriormente.

El tercer caso emplea el modelo YOLOTinyV3 con 10fps y una velocidad de detección “normal”.

```
FOR FRAME  1
Output for each object :  [{'name': 'person',
'percentage_probability': 43.94983649253845, 'box_points': [450,
257, 491, 358]}, {'name': 'person', 'percentage_probability':
43.878284096717834, 'box_points': [837, 444, 876, 573]}]
Output count for unique objects :  {'person': 2}
------------END OF A FRAME --------------
```

El cuarto caso emplea el modelo YOLOTinyV3 con 20fps y una velocidad de detección “flash”.

```
FOR FRAME  1
Output for each object :  []
Output count for unique objects :  {}
------------END OF A FRAME --------------
```

Se puede observar que el modelo YoloTiny ofrece una precisión mucho peor pues se trata de un modelo mucho más ligero y menos potente. En comparación con el modelo YOLO se puede apreciar la diferencia en la cantidad de objetos detectados teniendo en cuenta tan solo el primer frame del vídeo.

El último ejemplo emplea el modelo ResNet con 10 fps y una velocidad de detección “normal”.

```
FOR FRAME  1
Output for each object :  [{'name': 'fire hydrant',
'percentage_probability': 73.549485206604, 'box_points': [145, 265,
176, 319]}, {'name': 'person', 'percentage_probability':
59.26389694213867, 'box_points': [841, 459, 872, 527]}, {'name':
'person', 'percentage_probability': 54.164665937423706,
'box_points': [641, 623, 680, 701]}, {'name': 'person',
'percentage_probability': 54.02767062187195, 'box_points': [330,
287, 353, 341]}, {'name': 'person', 'percentage_probability':
49.38299357891083, 'box_points': [271, 232, 291, 281]}, {'name':
'person', 'percentage_probability': 46.398332715034485,
'box_points': [1159, 343, 1184, 412]}, {'name': 'person',
'percentage_probability': 44.904518127441406, 'box_points': [359,
140, 386, 169]}, {'name': 'person', 'percentage_probability':
44.73613798618317, 'box_points': [605, 346, 625, 408]}, {'name':
'fire hydrant', 'percentage_probability': 41.66639447212219,
'box_points': [331, 361, 364, 414]}, {'name': 'person',
'percentage_probability': 37.442606687545776, 'box_points': [656,
302, 689, 358]}, {'name': 'bird', 'percentage_probability':
35.89835166931152, 'box_points': [679, 262, 705, 308]}, {'name':
'person', 'percentage_probability': 35.34329533576965,
'box_points': [743, 358, 765, 419]}, {'name': 'person',
'percentage_probability': 33.33216309547424, 'box_points': [797,
376, 828, 423]}, {'name': 'person', 'percentage_probability':
30.632662773132324, 'box_points': [458, 274, 485, 327]}, {'name':
'person', 'percentage_probability': 30.298417806625366,
'box_points': [955, 82, 992, 160]}]
Output count for unique objects :  {'fire hydrant': 2, 'person':
12, 'bird': 1}
------------END OF A FRAME --------------
```

# Código de Funciones

## 4.1 Código e imports de la función imageClassification

Se comienza realizando los imports y adjuntando las librerías necesarias. En esta sección de código explicaremos algunas de las funciones necesarias para la ejecución de la herramienta. Colab tiene preinstalado un amplio repertorio de librerías que facilita su uso. Todo el código se encuentra en los archivos .py

```
import tensorflow as tf

from PIL import Image

import numpy as np

from matplotlib.cbook import deprecated
```

Respecto de la función `class ImageClassification:`

Esta es la clase de clasificación de imágenes perteneciente a la biblioteca ImageAI. Proporciona soporte para 4 tipos de modelos diferentes que son: ResNet, MobileNetV2, DenseNet e Inception V3.

Es necesario llamar a las siguientes funciones antes de poder realizar una clasificación.

* setModelPath (). Se corresponde con uno de los modelos que se va a utilizar.

* loadModel (). Cargar el modelo. Realizar antes de hacer una clasificación.

Una vez que se hayan llamado las funciones anteriores, puede llamar a la función classifyImage () de la instancia de clasificación objeto en cualquier momento para clasificar una imagen.

```
def __init__(self):

        self.__modelType = ""

        self.modelPath = ""

        self.__modelLoaded = False

        self.__model_collection = []

        self.__input_image_size = 224 %Tamaño de la imagen
```

```
    def setModelPath(self, model_path):
        """
         Función que establece el modelo que se va a usar
         """
        self.modelPath = model_path


    def setModelTypeAsMobileNetV2(self):
        """
        Función que asigna el modelo a mobilenetv2
        """
        self.__modelType = "mobilenetv2"


    def setModelTypeAsResNet(self):
         """
        Función que actualiza el modelo
        """
        return self.setModelTypeAsResNet50()


    def setModelTypeAsResNet50(self):
        """
         Función que asigna el modelo a resnet50
        """
        self.__modelType = "resnet50"


    def setModelTypeAsDenseNet(self):
        return self.setModelTypeAsDenseNet121()


    def setModelTypeAsDenseNet121(self):
        """
        Función que establece el modelo como densenet121
        """
        self.__modelType = "densenet121"
```

```
    def setModelTypeAsInceptionV3(self):
        """
        Función que establece el modelo como inceptionv3
        """
        self.__modelType = "inceptionv3"
def loadModel(self, classification_speed="normal"):

        if(classification_speed=="normal"):
            self.__input_image_size = 224
        elif(classification_speed=="fast"):
            self.__input_image_size = 160
        elif(classification_speed=="faster"):
            self.__input_image_size = 120
        elif (classification_speed == "fastest"):
            self.__input_image_size = 100

def classifyImage(self, image_input, result_count=5, input_type="file"):
```

La función loadModel se usa para cargar la estructura del modelo desde la ruta definida. Si la función setModelPath() recibe el parámetro opcional que indica la velocidad de procesado esta redimensiona el tamaño de la imagen original dada.

La función classifyImage clasifica la imagen según se reciben los siguientes parámetros: input_type: el tipo de entrada que se va a analizar, image_input: ruta del archivo y result_count: el número de clasificaciones.

Esta función devuelve 2 matrices, una de ellas corresponde con los resultados de clasificación y la otra contiene las probabilidades de clasificación asociad. Se denominan 'classification_results' y 'classification_probabilities'

Los resultados de clasificación contienen posibles clases de objetos ordenados en orden descendente de sus probabilidades porcentuales y la matriz de probabilidades nos indica el porcentaje de cada objeto detectado comparado con el objeto prestablecido. En el ejemplo que se ejecutó obtuvimos una lista de 5 posibles resultados y la herramienta asociaba una probabilidad a cada posible resultado. En el caso del estadio de fútbol se obtuvo con una alta probabilidad un escenario, un marcador, una grada de anfiteatro incluso una pista de aeropuerto haciendo referencia al terreno de juego abierto.

## 4.2 Código e imports de la función objectDetection y VideoObjectDetection

```
import cv2

from imageai.Detection.keras_retinanet import models as retinanet_models

from imageai.Detection.keras_retinanet.utils.image import read_image_bgr,
preprocess_image, resize_image

from imageai.Detection.keras_retinanet.utils.visualization import draw_box,
draw_caption

import matplotlib.pyplot as plt

import matplotlib.image as pltimage

import numpy as np

import tensorflow as tf

import os

from tensorflow.keras import backend as K

from tensorflow.keras.layers import Input

from PIL import Image

import colorsys

import warnings


from imageai.Detection.YOLO.yolov3 import tiny_yolov3_main, yolov3_main

from imageai.Detection.YOLO.utils import letterbox_image, yolo_eval,
preprocess_input, retrieve_yolo_detections, draw_boxes


class ObjectDetection:

def __init__(self):

        self.__modelType = ""

        self.modelPath = ""

        self.__modelPathAdded = False

        self.__modelLoaded = False


    self.__model_collection = []
```

```
        # Instance variables for RetinaNet Model
        self.__input_image_min = 1333
        self.__input_image_max = 800
        self.numbers_to_names = { número: 'insertar objetos' }
def setModelTypeAsRetinaNet(self):
        self.__modelType = "retinanet"
def setModelTypeAsYOLOv3(self):
        self.__modelType = "yolov3"
def setModelTypeAsTinyYOLOv3(self):
        self.__modelType = "tinyyolov3"
def setModelPath(self, model_path):
        if (self.__modelPathAdded == False):
            self.modelPath = model_path
            self.__modelPathAdded = True

def detectObjectsFromImage(self, input_image="", output_image_path="",
input_type="file", output_type="file", extract_detected_objects=False,
minimum_percentage_probability=50, display_percentage_probability=True,
display_object_name=True, display_box=True, thread_safe=False,
custom_objects=None):

:param input_image:
            :param output_image_path:
            :param input_type:
            :param output_type:
            :param extract_detected_objects:
            :param minimum_percentage_probability:
            :param display_percentage_probability:
            :param display_object_name:
            :param thread_safe:
            :return image_frame:
            :return output_objects_array:
            :return detected_objects_image_array:
        """
```

```
def detectObjectsFromVideo(self, input_file_path="", camera_input=None,
output_file_path="", frames_per_second=20, frame_detection_interval=1,
minimum_percentage_probability=50, log_progress=False,
display_percentage_probability=True, display_object_name=True,
display_box=True, save_detected_video=True, per_frame_function=None,
per_second_function=None, per_minute_function=None,
video_complete_function=None, return_detected_frame=False, detection_timeout =
None,  thread_safe=False, custom_objects=None):


                    :param input_file_path:

                    :param camera_input

                    :param output_file_path:

                    :param save_detected_video:

                    :param frames_per_second:

                    :param frame_detection_interval:

                    :param minimum_percentage_probability:

                    :param log_progress:

                    :param display_percentage_probability:

                    :param display_object_name:

                    :param per_frame_function:

                    :param per_second_function:

                    :param per_minute_function:

                    :param video_complete_function:

                    :param return_detected_frame:

                    :param detection_timeout:

                    :param thread_safe:

                    :return output_video_filepath:

                    :return counting:

                    :return output_objects_array:

                    :return output_objects_count:

                    :return detected_copy:

                    :return this_second_output_object_array:

                    :return this_second_counting_array:

                    :return this_second_counting:

                    :return this_minute_output_object_array:
```

```
                    :return this_minute_counting_array:

                    :return this_minute_counting:

                    :return this_video_output_object_array:

                    :return this_video_counting_array:

                    :return this_video_counting:

                """
```

Class ObjectDetection es la clase de detección de objetos en la biblioteca ImageAI. Proporciona soporte para 3 tipos de modelos: RetinaNet, YoloV3 y TinyYoloV3. Tras crear la instancia se podrá establecer las propiedades y realizar detección con las funciones predefinidas.

De igual manera que se explicó en la sección anterior es necesario llamar las funciones setModelPath() y loadModel(). Tras esto se puede llamar a la función detectObjectsFromImage

La función detecObjectsFromImage() se usa para detectar objetos observables en la imagen dada. Los valores devueltos por esta función dependen de los parámetros dados para analizar. Los posibles valores devueltos son una serie de diccionarios correspondiente a cada objeto detectado que contiene (nombre, probabilidad, box points) La función detecta los posibles objetos pertenecientes a la imagen:

Si extract_detected_objects = false o output_type = false devuelve un array de diccionarios con los objetos detectados en la imagen dada como ruta. Si output_type = 'array' devuelve un numpy array y matriz de diccionarios con los objetos detectados. Si extract_detected_objects = True devuelve una carpeta con cada objeto detectado independientemente.

Finalmente la función detectObjectsFromVideo() se usa para detectar objetos observables en la ruta de video dada o en una entrada de cámara:

Los parámetros devueltos son: input_file_path, que es la ruta del archivo al video de entrada. Solo es necesario si 'camera_input' no está configurado, camera_input, permite analizar la entrada de la cámara para detectar detecciones de video en vivo, output_file_path, que es la ruta al video de salida. Solo es necesario si 'save_detected_video' no está establecido como 'False', frames_per_second, que es el número de fotogramas que se utilizarán en el vídeo de salida, frame_detection_interval (opcional, 1 por defecto), que son los intervalos de fotogramas que se detectarán, minimum_percentage_probability (opcional, 50 por defecto), opción para establecer el porcentaje mínimo de probabilidad para detectar un objeto, log_progress (opcional), que indica si el progreso del marco procesado debe registrarse en la consola, display_percentage_probability (opcional), se puede utilizar para ocultar o mostrar puntuaciones de probabilidad en los fotogramas de vídeo detectados, display_object_name (opcional), se puede usar para mostrar u ocultar nombres de objetos en los fotogramas de video detectados, save_save_detected_video (opcional, True por defecto), se puede configurar o no para guardar el video detectado. Además de las funciones per_minute_function, per_second_function, per_frame_function que ya se mencionaron en el anterior capítulo.

Como conclusión de esta sección vamos a definir el código de las funciones forSecond y forFrame, funciones que nos detallan por la salida estándar toda la información de cada frame que se procesa en el vídeo que se adjunta como ruta de archivo como se puede observar en la *ilustración 34*.

```
def forSecond(frame2_number, output_arrays, count_arrays, average_count,
returned_frame):


    plt.clf()


    this_colors = []

    labels = []

    sizes = []


    counter = 0


    for eachItem in average_count:

        counter += 1

        labels.append(eachItem + " = " + str(average_count[eachItem]))

        sizes.append(average_count[eachItem])

        this_colors.append(color_index[eachItem])


    global resized


    if (resized == False):

        manager = plt.get_current_fig_manager()

        manager.resize(width=1000, height=500)

        resized = True


    plt.subplot(1, 2, 1)

    plt.title("Second : " + str(frame_number))

    plt.axis("off")

    plt.imshow(returned_frame, interpolation="none")


    plt.subplot(1, 2, 2)

    plt.title("Analysis: " + str(frame_number))

    plt.pie(sizes, labels=labels, colors=this_colors, shadow=True,
startangle=140, autopct="%1.1f%%")
```

```
    plt.pause(0.01)

def forFrame(frame_number, output_array, output_count, returned_frame):

    plt.clf()

    this_colors = []
    labels = []
    sizes = []

    counter = 0

    for eachItem in output_count:
        counter += 1
        labels.append(eachItem + " = " + str(output_count[eachItem]))
        sizes.append(output_count[eachItem])
        this_colors.append(color_index[eachItem])

    global resized

    if (resized == False):
        manager = plt.get_current_fig_manager()
        manager.resize(width=1000, height=500)
        resized = True

    plt.subplot(1, 2, 1)
    plt.title("Frame : " + str(frame_number))
    plt.axis("off")
    plt.imshow(returned_frame, interpolation="none")

    plt.subplot(1, 2, 2)
    plt.title("Analysis: " + str(frame_number))
    plt.pie(sizes, labels=labels, colors=this_colors, shadow=True,
startangle=140, autopct="%1.1f%%")

    plt.pause(0.01)
```

# Experimentos

En este capítulo se expondrán las conclusiones que se han obtenido tras realizar diversos experimentos habiendo variado diferentes parámetros de entrada. Se ha llegado a la conclusión por experiencia de que la herramienta es más eficiente cuanto mejor calidad tenga la imagen o el vídeo a procesar. Se han realizado experimentos con vídeos de 320p, 480p, 760p y 1080p y en el primer caso la herramienta apenas consigue detectar objetos mientras que en los vídeos de alta resolución procede obteniendo buenos resultados.

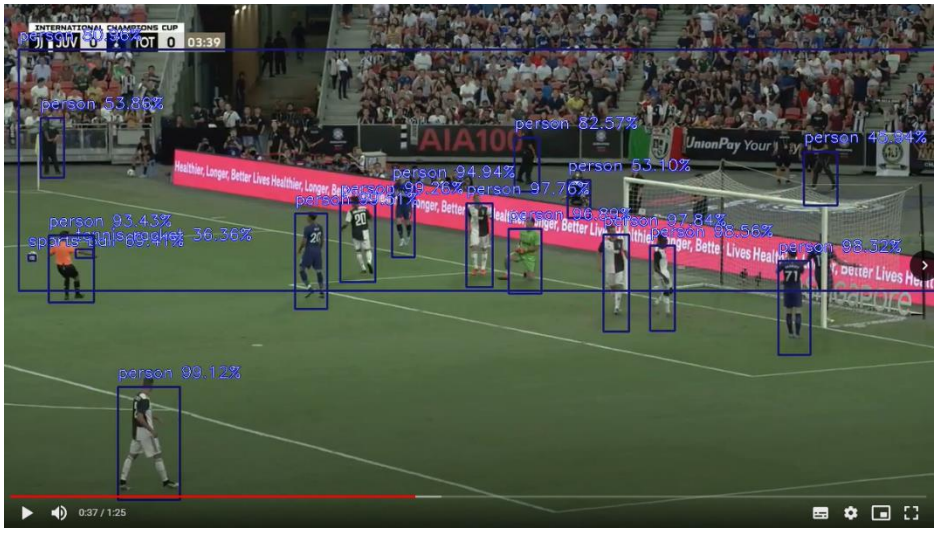


*Ilustración 43: Resultado de procesar un vídeo en calidad 8K*


*Ilustración 44: Resultado de procesar un vídeo en 320p*

Además, el estándar es 20 fotogramas por segundo y de igual manera se han realizado experimentos tanto a 10 como 30 fps y la diferencia es notable. En el caso de 10fps el algoritmo es capaz de detectar una mayor cantidad de objetos al ir ralentizado el vídeo, mientras que, por otro lado, usar 30fps implica procesar el vídeo en cámara rápida.


*Ilustración 45: Resultado de procesar un vídeo a 30fps en 720p*

En este caso se va a utilizar una sola imagen ya que por propia definición una sola imagen no implica movimiento y es complicado demostrar que el vídeo se procesa a 30fps en un formato .png. El algoritmo en este caso mantiene durante un tiempo menor la detección de los diferentes jugadores, aun así es capaz de detectar a casi todos los jugadores, aunque no al mismo tiempo.

Un partido de fútbol es un caso en el que existe mucho movimiento, por ende, los algoritmos YOLO y ResNet funcionan mucho mejor ya que son más robustos. Por otro lado, el modelo YoloTiny es mucho más ligero y se emplea en circunstancias en las que los vídeos no son tan dinámicos como puede ser el caso de un documental sobre la naturaleza en el que se pretenda contar el número de aves en reposo, por ejemplo.

# Conclusión

En este apartado se expondrán las conclusiones que se han sacado de este Trabajo Fin de Grado. ImageAI es una herramienta de implementación en tiempo real con un alto rendimiento que se emplea en las tecnologías de visión artificial, un campo que está en continuo crecimiento. Es una tecnología de vanguardia que puede funcionar con cualquier equipo informático con una capacidad moderada. Aunque para procesamiento y detección de vídeos se recomienda el uso de tecnologías GPU. Con TensorFlow (Anaconda.org, s.f.) las implementaciones de algoritmos de inteligencia artificial y aprendizaje automático es simple y fácil.

Cualquier persona interesada en construir sistemas de visión artificial y usarlo con fines comerciales, económicos, de investigación o sociales puede utilizar esta herramienta. La diferencia entre aplicar un método tradicional o utilizar esta herramienta se puede notar en los tiempos de ejecución, el alcance de cada herramienta o la simplicidad del código utilizado.

Además del uso que se le ha dado en este proyecto para la detección de jugadores en imágenes y en un campo de fútbol se puede aplicar para detectar cualquier objeto de interés como puede ser el tráfico, la circulación de coches o la entrada de personas a un recinto con un simple cambio en el código de las funciones. La herramienta no solo puede aplicarse en jugadores de un partido de fútbol, también un partido de rugby, hockey, baloncesto o Vóley.

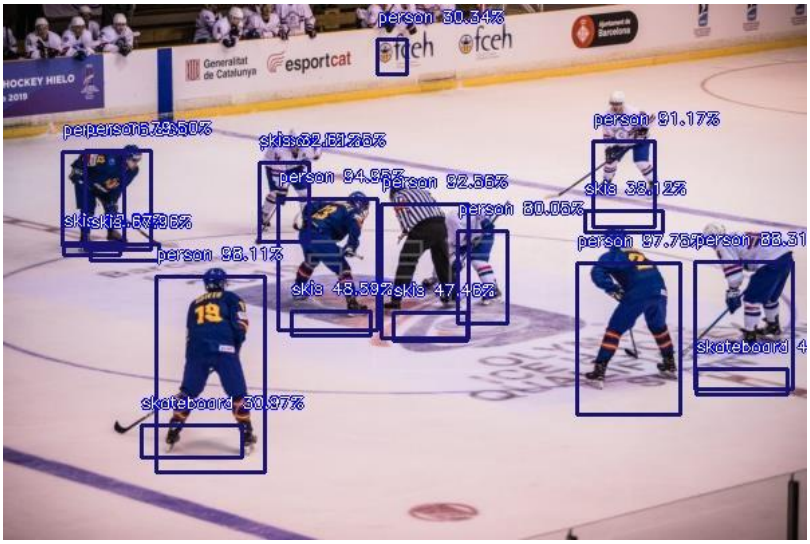


*Ilustración 46: Ejemplo reconocimiento en partido de Hockey sobre hielo*

```
person  :  30.33740222454071  :  [279, 26, 301, 52]
--------------------------------
person  :  91.17082357406616  :  [440, 102, 486, 166]
--------------------------------
person  :  76.85571312904358  :  [44, 110, 90, 180]
--------------------------------
person  :  79.49832677841187  :  [61, 109, 110, 183]
--------------------------------
person  :  82.55444765090942  :  [191, 118, 228, 177]
--------------------------------
skis  :  32.61066675186157  :  [191, 118, 228, 177]
--------------------------------
skis  :  38.12129199504852  :  [434, 154, 492, 170]
--------------------------------
skis  :  32.38912224769592  :  [45, 178, 87, 187]
--------------------------------
skis  :  67.96051859855652  :  [65, 179, 116, 191]
--------------------------------
person  :  94.94777321815491  :  [205, 145, 279, 243]
--------------------------------
person  :  92.56008863449097  :  [282, 149, 347, 249]
--------------------------------
person  :  80.04907369613647  :  [339, 169, 376, 238]
--------------------------------
skis  :  48.59285056591034  :  [215, 229, 274, 247]
--------------------------------
skis  :  47.45813608169556  :  [291, 230, 346, 251]
--------------------------------
person  :  97.74788618087769  :  [428, 193, 505, 306]
--------------------------------
person  :  88.31343650817871  :  [516, 192, 589, 287]
--------------------------------
person  :  98.11171889305115  :  [114, 203, 195, 349]
--------------------------------
skateboard  :  48.893240094184875  :  [517, 272, 585, 291]
--------------------------------
skateboard  :  30.968406796455383  :  [103, 314, 178, 338]
--------------------------------
```

*Ilustración 47: Resultado detección de objeto en la imagen del partido de Hockey*

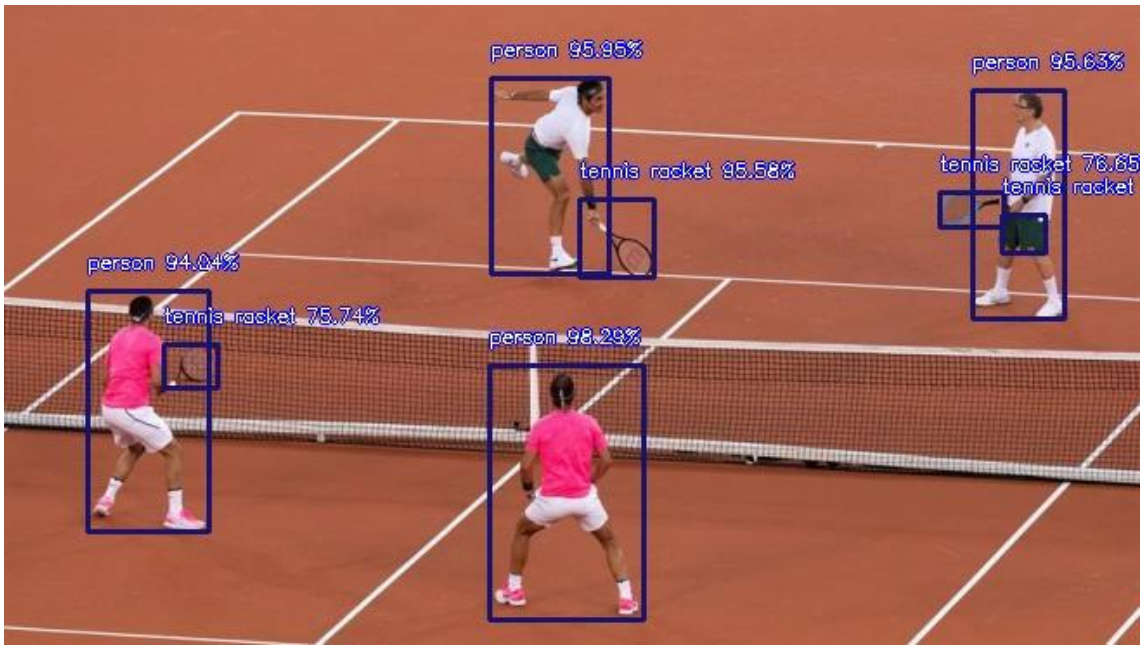


*Ilustración 48: Partido de Tenis*

```
WARNING:tensorflow:No training configuration found in the save file, so the model was *not* compiled. Compile it manually.
person  :  98.29033613204956  :  [280, 208, 369, 355]
--------------------------------
person  :  95.95049619674683  :  [281, 42, 349, 155]
--------------------------------
person  :  95.63030004501343  :  [559, 49, 612, 181]
--------------------------------
tennis racket  :  95.58138251304626  :  [332, 112, 375, 157]
--------------------------------
person  :  94.04250383377075  :  [48, 165, 118, 304]
--------------------------------
tennis racket  :  76.65385603904724  :  [540, 108, 577, 128]
--------------------------------
tennis racket  :  75.7364273071289  :  [92, 196, 123, 221]
--------------------------------
tennis racket  :  51.13970637321472  :  [576, 121, 601, 143]
--------------------------------
```

*Ilustración 49: Ejecución detección objetos partido tenis*


*Ilustración 50: Partido de baloncesto*

```
WARNING:tensorflow:No training configuration found in the save file, so the model was *not* compiled. Compile it manually.
person  :  81.78449273109436  :  [297, 186, 414, 374]
--------------------------------
person  :  79.53939437866211  :  [416, 146, 562, 374]
--------------------------------
person  :  79.42053079605103  :  [208, 154, 278, 366]
--------------------------------
person  :  75.62363743782043  :  [647, 194, 715, 345]
--------------------------------
person  :  68.49392056465149  :  [697, 195, 748, 329]
--------------------------------
person  :  61.84029579162598  :  [215, 84, 315, 365]
--------------------------------
person  :  59.89253520965576  :  [53, 241, 102, 333]
--------------------------------
person  :  59.02734994888306  :  [412, 189, 490, 359]
--------------------------------
person  :  54.23574447631836  :  [265, 66, 336, 272]
--------------------------------
person  :  52.64163613319397  :  [247, 82, 406, 371]
--------------------------------
person  :  46.17352485656738  :  [0, 255, 40, 374]
--------------------------------
person  :  44.7573721408844  :  [0, 184, 53, 374]
--------------------------------
sports ball  :  41.8388694524765  :  [292, 64, 315, 92]
--------------------------------
person  :  41.260191798210144  :  [390, 196, 420, 266]
--------------------------------
person  :  39.45923149585724  :  [371, 224, 397, 268]
--------------------------------
person  :  36.35486364364624  :  [420, 193, 456, 268]
--------------------------------
person  :  32.444074749946594  :  [508, 51, 530, 98]
--------------------------------
tv  :  32.32133686542511  :  [80, 1, 221, 118]
--------------------------------
person  :  31.692799925804138  :  [114, 202, 143, 246]
--------------------------------
person  :  31.31427764892578  :  [676, 165, 700, 193]
--------------------------------
chair  :  30.299672484397888  :  [50, 243, 100, 334]
--------------------------------
```

*Ilustración 51: Ejecución partido de baloncesto*

En 2015 se anunció el desafío de reconocimiento visual a gran escala de ImageNet (ILSVRC) se han desarrollado muchas soluciones de detección de objetos de video basadas en aprendizaje profundo desde entonces. Este documento (Zhu, Wei, Li, Yuan, & Kehtarnavaz, 2020) proporciona una revisión de los métodos de detección de objetos de video que se han desarrollado en los últimos años. Esta revisión ha cubierto una descripción general de diferentes categorías de métodos basados en el aprendizaje profundo para la detección de objetos de video. El rendimiento de varios detectores con o sin posprocesamiento se resume en las tablas de datos proporcionadas en términos de precisión de detección y velocidad de cálculo.

# Bibliografía y Referencias


[1] Visión artificial (25/04/2021) Wikipedia, la enciclopedia libre.

https://es.wikipedia.org/wiki/Visi%C3%B3n_artificial

[2] Zbigatron (13/07/2018) Lawrence G. Roberts, The Early History of Computer Vision.

https://zbigatron.com/the-early-history-of-computer-vision/

[3] Operador Sobel (21/07/2021) Wikipedia, la enciclopedia libre.

https://es.wikipedia.org/wiki/Operador_Sobel

[4] Transformada de Hough (19/03/2020) Wikipedia, la enciclopedia libre.

https://es.wikipedia.org/wiki/Transformada_de_Hough

[5] Filtro de Kalman (07/06/2021) Wikipedia, la enciclopedia libre.

https://es.wikipedia.org/wiki/Filtro_de_Kalman

[6] Grupo de investigación EDMANS (2006) Técnicas y algoritmos básicos de visión artificial.

https://publicaciones.unirioja.es/catalogo/online/VisionArtificial.pdf

[7] Standford Vision Lab; Standford University and Princeton University (11/03/2021) ImageNet.

https://www.image-net.org/

[8] Moses Olafenwa and John Olafenwa Revision 89a1c799 (2021) Official English Documentation for ImageAI!

https://imageai.readthedocs.io/en/latest/

[9] Buhigas J (14/02/2018) Todo lo que necesitas saber sobre TensorFlow. En Puentes Digitales.

https://puentesdigitales.com/2018/02/14/todo-lo-que-necesitas-saber-sobre-tensorflow-la-plataforma-para-inteligencia-artificial-de-google/

[10] Deep Learning. Qué es y por qué es importante (01/05/2020) Sas.

https://www.sas.com/es_es/insights/analytics/deep-learning.html#:~:text=El%20deep%20learning%20es%20un,de%20im%C3%A1genes%20o%20hacer%20predicciones.

[11] Internet de las cosas "IOT" (10/08/2021) Wikipedia, la enciclopedia libre.

https://es.wikipedia.org/wiki/Internet_de_las_cosas

[12] Histograma (22/08/2021) Wikipedia, la enciclopedia libre.

https://es.wikipedia.org/wiki/Histograma

[13] Ruiz Arahal, M (2019) Apuntes y transparencias de clase.

[14] Moses Olafenwa (08/05/2021) GitHub ImageAI.

https://github.com/OlafenwaMoses/ImageAI

[15] Moses Olafenwa and John Olafenwa Revisión 89a1c799. GitHub ImageAI: Prediction Classes.

https://imageai.readthedocs.io/en/latest/prediction/index.html

[16] Vídeo (28/08/2021) Wikipedia, la enciclopedia libre.

https://es.wikipedia.org/wiki/Video

[17] Flujo Óptico (19/09/2019) Wikipedia, la enciclopedia libre.

https://es.wikipedia.org/wiki/Flujo_%C3%B3ptico

[18] Método Lucas-Kanade (01/05/2020) Wikipedia, la enciclopedia libre.

https://es.wikipedia.org/wiki/M%C3%A9todo_Lucas%E2%80%93Kanade

[19] Método Horn–Schunck (19/09/2019) Wikipedia, la enciclopedia libre.

https://es.wikipedia.org/wiki/M%C3%A9todo_de_Horn%E2%80%93Schunck

[20] Borda (09/08/2019) GitHub ImageAI: Video Object Detection, Tracking and Analysis.

https://github.com/OlafenwaMoses/ImageAI/blob/master/imageai/Detection/VIDEO.md#videodetection

[21] Na8 (05/02/2019) ¿Machine learning en la nube? Google Colaboratory con GPU!

https://www.aprendemachinelearning.com/machine-learning-en-la-nube-google-colaboratory-con-gpu/

[22] Moses Olafenwa and John Olafenwa Revisión 89a1c799. Video and Live-Feed Detection and Analysis.

https://imageai.readthedocs.io/en/latest/video/

[23] Anaconda documentation. TensorFlow. Anaconda.

https://docs.anaconda.com/anaconda/user-guide/tasks/tensorflow/

[24] Haidi Zhu, Haoran Wei, Baoqing Li, Xiaobing Yuan and Nasser Kehtarnavaz. (04/10/2020) A Review of Video Object Detection: Datasets, Metrics and Methods. MDPI applied sciences

# Anexos

En este apartado se incluirá para que quede constancia el código desarrollado en la plataforma Google Colaboratoy (Na8, 2019).

# Trabajo Fin de Grado: Detección de objetos en imágenes utilizando técnicas de aprendizaje profundo (Deep-Learning)

autor: Daniel Estévez Trigo.

# ImageAI: Object Detection

En este documento se detallará el código utilizado para la ejecución de la herramienta utilizada imageAI y la obtención de sus resultados.

Se comienza estableciendo la ruta al directorio donde se encuentras los archivos que se van a utilizar e instalando las dependencias necesarias. La herramienta sirve para detección de objetos, extracción de caracterísitcas, reconocimiento facial etc.

Comenzamos estableciendo nuestro drive como directorio principal en el que se encuentran las liberías, ficheros y modelos de prueba necesarios utilizados para la simulación.

```
from google.colab import drive
drive.mount('/content/drive/')
%cd /content/drive/MyDrive/TFG/ImageAI-master/
```

```
Drive already mounted at /content/drive/; to attempt to forcibly remount, call drive
/content/drive/MyDrive/TFG/ImageAI-master
```

```
from google.colab import drive
drive.mount("/content/drive/", force_remount=True)
%cd /content/drive/MyDrive/TFG/ImageAI-master/
```

```
Mounted at /content/drive/
/content/drive/MyDrive/TFG/ImageAI-master
```

```
pip install tensorflow==2.4.0
```

```
pip install keras==2.4.3 numpy==1.19.3 pillow==7.0.0 scipy==1.4.1 h5py==2.10.0 matplotlib=
```

```
pip install imageai --upgrade
```

```
pip install folium==0.2.1
```

```
pip install imgaug==0.2.5
```

## Modelo "Yolo.h5"

```
# USAR YoloV3

from imageai.Detection import ObjectDetection
import os

execution_path = os.getcwd()

detector = ObjectDetection()
detector.setModelTypeAsYOLOv3()
detector.setModelPath( os.path.join(execution_path , "yolo.h5"))
detector.loadModel()
detections = detector.detectObjectsFromImage(input_image=os.path.join(execution_path , "da

for eachObject in detections:
    print(eachObject["name"] , " : ", eachObject["percentage_probability"], " : ", eachObj
    print("--------------------------------")
```

```
person  :  92.86525845527649  :  [141, 41, 339, 250]
--------------------------------
person  :  99.4719922542572  :  [393, 267, 622, 560]
--------------------------------
person  :  95.09682059288025  :  [540, 288, 771, 564]
--------------------------------
person  :  98.23715686798096  :  [431, 22, 584, 296]
--------------------------------
person  :  89.85116481781006  :  [550, 15, 700, 322]
--------------------------------
person  :  96.48433327674866  :  [308, 41, 451, 309]
--------------------------------
person  :  96.3990867137909  :  [677, 48, 832, 339]
--------------------------------
person  :  99.34040904045105  :  [64, 43, 201, 521]
--------------------------------
person  :  98.67589473724365  :  [147, 176, 303, 523]
--------------------------------
person  :  90.58892130851746  :  [284, 215, 434, 519]
--------------------------------
person  :  64.0281081199646  :  [715, 294, 874, 546]
--------------------------------
```

Comenzamos importando la clase ObjectDetection perteneciente a ImageAI y obteniendo la ruta a nuestro directorio de archivos.

Definimos el detector creando una nueva instancia de la clase ObjetDetection y estableciendo el tipo de modelo como "yolo.h5"

Ejecutamos la función detectObjectsFromImage y analizamos la ruta a nuestra imagen, y a la nueva imagen que nos da la función.

La función devuelve una matriz de diccionarios. Cada diccionario corresponde al número de objetos detectados en la imagen. Cada diccionario tiene el nombre de las propiedades (nombre del objeto), percent_probability (porcentaje de probabilidad) y box_points (las coordenadas del cuadro delimitador del objeto).

```
# USAR YoloV3

from imageai.Detection import ObjectDetection
import os

execution_path = os.getcwd()

detector = ObjectDetection()
detector.setModelTypeAsYOLOv3()
detector.setModelPath( os.path.join(execution_path , "yolo.h5"))
detector.loadModel(detection_speed="fast")
detections = detector.detectObjectsFromImage(input_image=os.path.join(execution_path , "da

for eachObject in detections:
    print(eachObject["name"] , " : ", eachObject["percentage_probability"], " : ", eachObj
    print("--------------------------------")
```

```
WARNING:tensorflow:5 out of the last 5 calls to <function Model.make_predict_functior
person  :  67.08927750587463  :  [362, 49, 644, 314]
--------------------------------
person  :  92.5142228603363  :  [163, 36, 338, 280]
--------------------------------
person  :  60.25691032409668  :  [313, 51, 457, 297]
--------------------------------
person  :  80.21875023841858  :  [550, 33, 702, 317]
--------------------------------
person  :  95.47942280769348  :  [671, 47, 837, 499]
--------------------------------
person  :  99.78219866752625  :  [72, 55, 213, 510]
--------------------------------
person  :  99.07151460647583  :  [149, 178, 323, 519]
--------------------------------
person  :  95.87342143058777  :  [268, 218, 437, 524]
--------------------------------
person  :  94.14601922035217  :  [416, 256, 610, 544]
--------------------------------
person  :  89.23340439796448  :  [554, 272, 749, 543]
--------------------------------
person  :  88.05240988731384  :  [716, 302, 890, 553]
--------------------------------
sports ball  :  31.33937418460846  :  [412, 393, 445, 424]
--------------------------------
```

```
# USAR YoloV3
```

```
from imageai.Detection import ObjectDetection
import os

execution_path = os.getcwd()

detector = ObjectDetection()
detector.setModelTypeAsYOLOv3()
detector.setModelPath( os.path.join(execution_path , "yolo.h5"))
detector.loadModel(detection_speed="fastest")
detections = detector.detectObjectsFromImage(input_image=os.path.join(execution_path , "da

for eachObject in detections:
    print(eachObject["name"] , " : ", eachObject["percentage_probability"], " : ", eachObj
    print("--------------------------------")
```

```
WARNING:tensorflow:6 out of the last 6 calls to <function Model.make_predict_functior
person  :  72.59347438812256  :  [437, 77, 593, 415]
--------------------------------
person  :  60.84123253822327  :  [570, 83, 711, 432]
--------------------------------
person  :  86.8051290512085  :  [681, 118, 839, 467]
--------------------------------
person  :  90.99949598312378  :  [89, 99, 242, 527]
--------------------------------
person  :  64.87907767295837  :  [182, 184, 328, 559]
--------------------------------
person  :  78.74552011489868  :  [307, 200, 447, 576]
--------------------------------
person  :  71.78152203559875  :  [421, 195, 586, 588]
--------------------------------
person  :  57.82100558280945  :  [190, 46, 332, 239]
--------------------------------
person  :  44.47240233421326  :  [545, 26, 690, 312]
--------------------------------
person  :  83.25980305671692  :  [311, 60, 454, 308]
--------------------------------
person  :  67.78962016105652  :  [571, 285, 717, 515]
--------------------------------
person  :  60.20341515541077  :  [743, 346, 884, 558]
--------------------------------
```

```
# USAR YoloV3

from imageai.Detection import ObjectDetection
import os

execution_path = os.getcwd()

detector = ObjectDetection()
detector.setModelTypeAsYOLOv3()
detector.setModelPath( os.path.join(execution_path , "yolo.h5"))
detector.loadModel(detection_speed="flash")
detections = detector.detectObjectsFromImage(input_image=os.path.join(execution_path , "da

for eachObject in detections:
    print(eachObject["name"] , " : ", eachObject["percentage_probability"], " : ", eachObj
    print("                                    ")
```

```
    print( -------------------------------- )

     person  :  48.92940819263458  :  [421, 56, 580, 333]
     --------------------------------
     person  :  74.88034963607788  :  [554, 59, 711, 348]
     --------------------------------
     person  :  51.957279443740845  :  [695, 109, 847, 450]
     --------------------------------
     person  :  88.39104175567627  :  [66, 63, 202, 522]
     --------------------------------
     person  :  59.913796186447144  :  [108, 107, 258, 512]
     --------------------------------
     person  :  54.70535159111023  :  [175, 133, 326, 508]
     --------------------------------
     person  :  60.905832052230835  :  [278, 133, 426, 523]
     --------------------------------
     person  :  47.10249602794647  :  [407, 272, 587, 551]
     --------------------------------
     person  :  78.05079221725464  :  [562, 284, 722, 542]
     --------------------------------
     person  :  56.34065866470337  :  [705, 327, 870, 561]
     --------------------------------
```

```
# USAR YoloV3

from imageai.Detection import ObjectDetection
import os

execution_path = os.getcwd()

detector = ObjectDetection()
detector.setModelTypeAsYOLOv3()
detector.setModelPath( os.path.join(execution_path , "yolo.h5"))
detector.loadModel(detection_speed="fastest")
detections = detector.detectObjectsFromImage(input_image=os.path.join(execution_path , "da

for eachObject in detections:
    print(eachObject["name"] , " : ", eachObject["percentage_probability"], " : ", eachObj
    print("--------------------------------")
```

En el siguiente ejemplo vamos a probar con una imagen personalizada de la plantilla de la selección de futbol española para comprobar los parámetros que extrae la herramienta.

```
# USAR YoloV3

from imageai.Detection import ObjectDetection
import os

execution_path = os.getcwd()

detector = ObjectDetection()
detector.setModelTypeAsYOLOv3()
detector.setModelPath( os.path.join(execution_path , "yolo.h5"))
detector.loadModel()
detections = detector.detectObjectsFromImage(input_image=os.path.join(execution_path , "da
```

```
for eachObject in detections:
    print(eachObject["name"] , " : ", eachObject["percentage_probability"], " : ", eachObj
    print("--------------------------------")
```

A continuación, vamos a extraer cada objeto de la imagen de entrada y guardarlo de forma independiente. El código es practicamente idéntico al anterior, guardaremos cada objeto detectado como una imagen separada mediante un bucle for y el parámetro extract_detected_objects.

```
from imageai.Detection import ObjectDetection
import os

execution_path = os.getcwd()

detector = ObjectDetection()
detector.setModelTypeAsYOLOv3()
detector.setModelPath( os.path.join(execution_path , "yolo.h5"))
detector.loadModel()

detections, objects_path = detector.detectObjectsFromImage(input_image=os.path.join(execut

for eachObject, eachObjectPath in zip(detections, objects_path):
    print(eachObject["name"] , " : " , eachObject["percentage_probability"], " : ", eachOb
    print("Object's image saved in " + eachObjectPath)
    print("--------------------------------")
```

```
Object's image saved in /content/drive/My Drive/TFG/ImageAI-master/data-images/sel
--------------------------------
person  :  98.90502691268921  :  [263, 20, 302, 86]
Object's image saved in /content/drive/My Drive/TFG/ImageAI-master/data-images/sel
--------------------------------
person  :  73.20943474769592  :  [0, 6, 31, 97]
Object's image saved in /content/drive/My Drive/TFG/ImageAI-master/data-images/sel
--------------------------------
person  :  84.16476845741272  :  [99, 17, 141, 96]
Object's image saved in /content/drive/My Drive/TFG/ImageAI-master/data-images/sel
--------------------------------
person  :  69.96843218803406  :  [70, 48, 109, 112]
Object's image saved in /content/drive/My Drive/TFG/ImageAI-master/data-images/sel
--------------------------------
person  :  69.11114454269409  :  [110, 47, 151, 114]

Object's image saved in /content/drive/My Drive/TFG/ImageAI-master/data-images/sel
--------------------------------
person  :  98.35211038589478  :  [146, 49, 193, 120]
Object's image saved in /content/drive/My Drive/TFG/ImageAI-master/data-images/sel
--------------------------------
person  :  90.35298228263855  :  [189, 54, 233, 118]
Object's image saved in /content/drive/My Drive/TFG/ImageAI-master/data-images/sel
--------------------------------
person  :  96.85490131378174  :  [227, 53, 267, 122]
Object's image saved in /content/drive/My Drive/TFG/ImageAI-master/data-images/sel
--------------------------------
person  :  70.64650058746338  :  [262, 61, 300, 121]
Object's image saved in /content/drive/My Drive/TFG/ImageAI-master/data-images/sel
```

```
--------------------------------
person  :  97.45500087738037  :  [2, 44, 33, 157]
Object's image saved in /content/drive/My Drive/TFG/ImageAI-master/data-images/sel
--------------------------------
person  :  91.75204038619995  :  [40, 47, 73, 156]
Object's image saved in /content/drive/My Drive/TFG/ImageAI-master/data-images/sel
--------------------------------
person  :  99.50358271598816  :  [302, 55, 329, 190]
Object's image saved in /content/drive/My Drive/TFG/ImageAI-master/data-images/sel
--------------------------------
person  :  99.541574716568  :  [8, 73, 59, 182]
Object's image saved in /content/drive/My Drive/TFG/ImageAI-master/data-images/sel
--------------------------------
person  :  98.81204962730408  :  [64, 81, 105, 181]
Object's image saved in /content/drive/My Drive/TFG/ImageAI-master/data-images/sel
--------------------------------
person  :  98.4143078327179  :  [108, 78, 152, 184]
Object's image saved in /content/drive/My Drive/TFG/ImageAI-master/data-images/sel
--------------------------------
person  :  99.26004409790039  :  [153, 88, 190, 184]
Object's image saved in /content/drive/My Drive/TFG/ImageAI-master/data-images/sel
--------------------------------
person  :  98.94551038742065  :  [187, 88, 233, 187]
Object's image saved in /content/drive/My Drive/TFG/ImageAI-master/data-images/sel
--------------------------------
person  :  95.26483416557312  :  [232, 90, 269, 189]
Object's image saved in /content/drive/My Drive/TFG/ImageAI-master/data-images/sel
--------------------------------
person  :  97.87806868553162  :  [272, 90, 315, 189]
Object's image saved in /content/drive/My Drive/TFG/ImageAI-master/data-images/sel
```

```
from imageai.Detection import ObjectDetection
import os

execution_path = os.getcwd()

detector = ObjectDetection()
detector.setModelTypeAsYOLOv3()
detector.setModelPath( os.path.join(execution_path , "yolo.h5"))
detector.loadModel()

detections, objects_path = detector.detectObjectsFromImage(input_image=os.path.join(execut

for eachObject, eachObjectPath in zip(detections, objects_path):
    print(eachObject["name"] , " : " , eachObject["percentage_probability"], " : ", eachOb
    print("Object's image saved in " + eachObjectPath)
    print("--------------------------------")
```

Llamamos a detectObjectsFromImage (), analizamos en la ruta de la imagen de entrada, la parte de la imagen de salida y un parámetro adicional extract_detected_objects = True. Este parámetro establece que la función debe extraer cada objeto detectado de la imagen y guardarlo en una imagen separada que es FALSE por defecto. La función creará un nuevo directorio que es la ruta de la imagen de salida + "-objetos" y guarda todas las imágenes

extraídas en este nuevo directorio, siendo el nombre de cada imagen el nombre del objeto detectado + "-" + un número que corresponde al orden en el que se detectaron los objetos.

La función detectObjectsFromImage () devuelve el porcentaje de probabilidad para cada objeto detectado. La función tiene un parámetro mínimo_porcentaje_probabilidad, cuyo valor predeterminado es 50 (este valor puede variar entre 0 y 100). Para este ejemplo establecemos dicho valor en 30 lo que significa que para devolver un objeto detectado su porcentaje de probabilidad debe ser 30 o superior. Podemos ajustar la detección de objetos estableciendo un mínimo_porcentaje_probabilidad igual a un valor más pequeño para detectar una mayor

## Speed Detection

```
detector.loadModel(detection_speed="normal")
# detector.loadModel(detection_speed="fast")
# detector.loadModel(detection_speed="faster")
# detector.loadModel(detection_speed="fastest")
# detector.loadModel(detection_speed="flash")
```

ImageAI proporciona diferentes velocidades de detección. Nos permiten reducir el tiempo de detección a una tasa entre (20% - 80%) con cambios leves y resultados de detección precisos. Junto con la reducción del parámetro de probabilidad de porcentaje mínimo, las detecciones pueden igualar la velocidad normal reduciendo drásticamente el tiempo de detección. Las velocidades de detección disponibles son "normal" (predeterminada), "rápida", "más rápida", "más rápida" y "flash".

## Modelo "Resnet50_coco_best_v2.0.1.h5"

```
pip install keras_resnet
```

Tiene un tamaño de 145 mb. Ofrece un buen rendimiento y precisión con un tiempo de detección mayor.

```
from imageai.Detection import ObjectDetection
import os


execution_path = os.getcwd()

# USAR RetinaNet
detector = ObjectDetection()
detector.setModelTypeAsRetinaNet()
detector.setModelPath(os.path.join(execution_path, "resnet50_coco_best_v2.1.0.h5"))
detector.loadModel()
```

```
detections = detector.detectObjectsFromImage(input_image=os.path.join(execution_path , "da

for eachObject in detections:
    print(eachObject["name"] , " : ", eachObject["percentage_probability"], " : ", eachObj
    print("--------------------------------")
```

```
WARNING:tensorflow:No training configuration found in the save file, so the model was
WARNING:tensorflow:5 out of the last 5 calls to <function Model.make_predict_functior
person  :  93.54552030563354  :  [524, 681, 1064, 2061]
--------------------------------
person  :  82.988840341568  :  [2394, 735, 3039, 2092]
--------------------------------
person  :  82.98707604408264  :  [977, 706, 1515, 2066]
--------------------------------
person  :  81.79372549057007  :  [2110, 753, 2841, 2052]
--------------------------------
person  :  80.01710772514343  :  [1454, 722, 2019, 2038]
--------------------------------
person  :  74.62447881698608  :  [717, 682, 1312, 2093]
--------------------------------
person  :  74.18845891952515  :  [1898, 648, 2603, 2049]
--------------------------------
person  :  71.75090312957764  :  [590, 366, 1069, 934]
--------------------------------
person  :  71.5584397315979  :  [1018, 345, 1483, 909]
--------------------------------
person  :  55.32350540161133  :  [2646, 338, 3050, 1350]
--------------------------------
person  :  49.75452721118927  :  [1847, 335, 2320, 1305]
--------------------------------
person  :  44.994062185287476  :  [2235, 396, 2630, 1434]
--------------------------------
person  :  42.60857105255127  :  [347, 835, 484, 971]
--------------------------------
person  :  37.051770091056824  :  [2414, 749, 2740, 2014]
--------------------------------
sports ball  :  36.51437759399414  :  [2547, 1896, 2628, 1979]
--------------------------------
person  :  35.5748176574707  :  [3071, 731, 3138, 822]
--------------------------------
person  :  34.83889400959015  :  [1432, 367, 1851, 1460]
--------------------------------
person  :  34.498292207717896  :  [67, 836, 200, 970]
--------------------------------
person  :  33.059197664260864  :  [1877, 762, 2271, 2047]
--------------------------------
person  :  31.700238585472107  :  [1907, 358, 2598, 1169]
--------------------------------
person  :  30.340611934661865  :  [453, 487, 510, 614]
--------------------------------
```

```
from imageai.Detection import ObjectDetection
import os


execution_path = os.getcwd()

# USAR RetinaNet
```

```
# USAR RetinaNet
detector = ObjectDetection()
detector.setModelTypeAsRetinaNet()
detector.setModelPath(os.path.join(execution_path, "resnet50_coco_best_v2.1.0.h5"))
detector.loadModel(detection_speed="fast")

detections = detector.detectObjectsFromImage(input_image=os.path.join(execution_path , "da

for eachObject in detections:
    print(eachObject["name"] , " : ", eachObject["percentage_probability"], " : ", eachObj
    print("--------------------------------")
```

```
WARNING:tensorflow:No training configuration found in the save file, so the model was
WARNING:tensorflow:6 out of the last 6 calls to <function Model.make_predict_functior
person  :  93.54552030563354  :  [524, 681, 1064, 2061]
--------------------------------
person  :  82.988840341568  :  [2394, 735, 3039, 2092]
--------------------------------
person  :  82.98707604408264  :  [977, 706, 1515, 2066]
--------------------------------
person  :  81.79372549057007  :  [2110, 753, 2841, 2052]
--------------------------------
person  :  80.01710772514343  :  [1454, 722, 2019, 2038]
--------------------------------
person  :  74.62447881698608  :  [717, 682, 1312, 2093]
--------------------------------
person  :  74.18845891952515  :  [1898, 648, 2603, 2049]
--------------------------------
person  :  71.75090312957764  :  [590, 366, 1069, 934]
--------------------------------
person  :  71.5584397315979  :  [1018, 345, 1483, 909]
--------------------------------
person  :  55.32350540161133  :  [2646, 338, 3050, 1350]
--------------------------------
person  :  49.75452721118927  :  [1847, 335, 2320, 1305]
--------------------------------
person  :  44.994062185287476  :  [2235, 396, 2630, 1434]
--------------------------------
person  :  42.60857105255127  :  [347, 835, 484, 971]
--------------------------------
person  :  37.051770091056824  :  [2414, 749, 2740, 2014]
--------------------------------
sports ball  :  36.51437759399414  :  [2547, 1896, 2628, 1979]
--------------------------------
person  :  35.5748176574707  :  [3071, 731, 3138, 822]
--------------------------------
person  :  34.83889400959015  :  [1432, 367, 1851, 1460]
--------------------------------
person  :  34.498292207717896  :  [67, 836, 200, 970]
--------------------------------
person  :  33.059197664260864  :  [1877, 762, 2271, 2047]
--------------------------------
person  :  31.700238585472107  :  [1907, 358, 2598, 1169]
--------------------------------
person  :  30.340611934661865  :  [453, 487, 510, 614]
--------------------------------
```

```
from imageai.Detection import ObjectDetection
import os
```

```
import os


execution_path = os.getcwd()

# USAR RetinaNet
detector = ObjectDetection()
detector.setModelTypeAsRetinaNet()
detector.setModelPath(os.path.join(execution_path, "resnet50_coco_best_v2.1.0.h5"))
detector.loadModel(detection_speed="fastest")

detections = detector.detectObjectsFromImage(input_image=os.path.join(execution_path , "da

for eachObject in detections:
    print(eachObject["name"] , " : ", eachObject["percentage_probability"], " : ", eachObj
    print("--------------------------------")
```

```
WARNING:tensorflow:No training configuration found in the save file, so the model was
person  :  93.54552030563354  :  [524, 681, 1064, 2061]
--------------------------------
person  :  82.988840341568  :  [2394, 735, 3039, 2092]
--------------------------------
person  :  82.98707604408264  :  [977, 706, 1515, 2066]
--------------------------------
person  :  81.79372549057007  :  [2110, 753, 2841, 2052]
--------------------------------
person  :  80.01710772514343  :  [1454, 722, 2019, 2038]
--------------------------------
person  :  74.62447881698608  :  [717, 682, 1312, 2093]
--------------------------------
person  :  74.18845891952515  :  [1898, 648, 2603, 2049]
--------------------------------
person  :  71.75090312957764  :  [590, 366, 1069, 934]
--------------------------------
person  :  71.5584397315979  :  [1018, 345, 1483, 909]
--------------------------------
person  :  55.32350540161133  :  [2646, 338, 3050, 1350]
--------------------------------
person  :  49.75452721118927  :  [1847, 335, 2320, 1305]
--------------------------------
person  :  44.994062185287476  :  [2235, 396, 2630, 1434]
--------------------------------
person  :  42.60857105255127  :  [347, 835, 484, 971]
--------------------------------
person  :  37.051770091056824  :  [2414, 749, 2740, 2014]
--------------------------------
sports ball  :  36.51437759399414  :  [2547, 1896, 2628, 1979]
--------------------------------
person  :  35.5748176574707  :  [3071, 731, 3138, 822]
--------------------------------
person  :  34.83889400959015  :  [1432, 367, 1851, 1460]
--------------------------------
person  :  34.498292207717896  :  [67, 836, 200, 970]
--------------------------------
person  :  33.059197664260864  :  [1877, 762, 2271, 2047]
--------------------------------
person  :  31.700238585472107  :  [1907, 358, 2598, 1169]
--------------------------------
person  :  30.340611934661865  :  [453, 487, 510, 614]
--------------------------------
```

```
from imageai.Detection import ObjectDetection
import os


execution_path = os.getcwd()

# USAR RetinaNet
detector = ObjectDetection()
detector.setModelTypeAsRetinaNet()
detector.setModelPath(os.path.join(execution_path, "resnet50_coco_best_v2.1.0.h5"))
detector.loadModel(detection_speed="flash")

detections = detector.detectObjectsFromImage(input_image=os.path.join(execution_path , "da

for eachObject in detections:
    print(eachObject["name"] , " : ", eachObject["percentage_probability"], " : ", eachObj
    print("--------------------------------")
```

```
WARNING:tensorflow:No training configuration found in the save file, so the model was
person  :  93.54552030563354  :  [524, 681, 1064, 2061]
--------------------------------
person  :  82.988840341568  :  [2394, 735, 3039, 2092]
--------------------------------
person  :  82.98707604408264  :  [977, 706, 1515, 2066]
--------------------------------
person  :  81.79372549057007  :  [2110, 753, 2841, 2052]
--------------------------------
person  :  80.01710772514343  :  [1454, 722, 2019, 2038]
--------------------------------
person  :  74.62447881698608  :  [717, 682, 1312, 2093]
--------------------------------
person  :  74.18845891952515  :  [1898, 648, 2603, 2049]
--------------------------------
person  :  71.75090312957764  :  [590, 366, 1069, 934]
--------------------------------
person  :  71.5584397315979  :  [1018, 345, 1483, 909]
--------------------------------
person  :  55.32350540161133  :  [2646, 338, 3050, 1350]
--------------------------------
person  :  49.75452721118927  :  [1847, 335, 2320, 1305]
--------------------------------
person  :  44.994062185287476  :  [2235, 396, 2630, 1434]
--------------------------------
person  :  42.60857105255127  :  [347, 835, 484, 971]
--------------------------------
person  :  37.051770091056824  :  [2414, 749, 2740, 2014]
--------------------------------
sports ball  :  36.51437759399414  :  [2547, 1896, 2628, 1979]
--------------------------------
person  :  35.5748176574707  :  [3071, 731, 3138, 822]
--------------------------------
person  :  34.83889400959015  :  [1432, 367, 1851, 1460]
--------------------------------
person  :  34.498292207717896  :  [67, 836, 200, 970]
--------------------------------
person  :  33.059197664260864  :  [1877, 762, 2271, 2047]
```

```
--------------------------------
person  :  31.700238585472107  :  [1907, 358, 2598, 1169]
--------------------------------
person  :  30.340611934661865  :  [453, 487, 510, 614]
--------------------------------
```

```
from imageai.Detection import ObjectDetection
import os

execution_path = os.getcwd()

detector = ObjectDetection()
detector.setModelTypeAsRetinaNet()
detector.setModelPath( os.path.join(execution_path , "resnet50_coco_best_v2.1.0.h5"))
detector.loadModel()

detections, objects_path = detector.detectObjectsFromImage(input_image=os.path.join(execut

for eachObject, eachObjectPath in zip(detections, objects_path):
    print(eachObject["name"] , " : " , eachObject["percentage_probability"], " : ", eachOb
    print("Object's image saved in " + eachObjectPath)
    print("--------------------------------")
```

## Modelo "Yolo-tiny.h5"

Tiene un tamaño de 34mb, pesa poco en comparación a los modelos anteriores porque está optimizado para ser rápido y ofrecer un rendimento decente. Tiene un tiempo de detección rápido.

```
from imageai.Detection import ObjectDetection
import os

execution_path = os.getcwd()

# USAR Yolo-tiny
detector = ObjectDetection()
detector.setModelTypeAsTinyYOLOv3()
detector.setModelPath( os.path.join(execution_path , "yolo-tiny.h5"))
detector.loadModel()

# detector.loadModel(detection_speed="fast")
detections = detector.detectObjectsFromImage(input_image=os.path.join(execution_path , "da

for eachObject in detections:
    print(eachObject["name"] , " : ", eachObject["percentage_probability"], " : ", eachObj
    print("--------------------------------")
```

```
person  :  47.201499342918396  :  [159, 31, 360, 262]
--------------------------------
person  :  25.047898292541504  :  [303, 29, 459, 333]
```

```
--------------------------------
person  :  18.169406056404114  :  [408, 26, 608, 339]
--------------------------------
person  :  45.52554488182068  :  [59, 15, 194, 541]
--------------------------------
person  :  91.73627495765686  :  [132, 135, 313, 559]
--------------------------------
person  :  82.24657773971558  :  [261, 178, 417, 543]
--------------------------------
person  :  24.085770547389984  :  [404, 226, 619, 516]
--------------------------------
person  :  41.29771590232849  :  [553, 247, 751, 545]
--------------------------------
person  :  59.94440317153931  :  [703, 284, 886, 554]
--------------------------------
person  :  16.708384454250336  :  [647, 327, 793, 546]
--------------------------------
sports ball  :  57.300060987472534  :  [410, 372, 449, 450]
--------------------------------
```

```
from imageai.Detection import ObjectDetection
import os

execution_path = os.getcwd()

# USAR Yolo-tiny
detector = ObjectDetection()
detector.setModelTypeAsTinyYOLOv3()
detector.setModelPath( os.path.join(execution_path , "yolo-tiny.h5"))
detector.loadModel(detection_speed="fast")

detections = detector.detectObjectsFromImage(input_image=os.path.join(execution_path , "da

for eachObject in detections:
    print(eachObject["name"] , " : ", eachObject["percentage_probability"], " : ", eachObj
    print("--------------------------------")
```

```
WARNING:tensorflow:5 out of the last 5 calls to <function Model.make_predict_functior
person  :  65.77417850494385  :  [167, 34, 373, 263]
--------------------------------
person  :  80.64867854118347  :  [419, 25, 612, 310]
--------------------------------
person  :  67.52105355262756  :  [551, 20, 720, 340]
--------------------------------
person  :  49.710243940353394  :  [291, 28, 459, 348]
--------------------------------
person  :  33.94358158111572  :  [644, 21, 845, 528]
--------------------------------
person  :  94.62798833847046  :  [55, 23, 221, 548]
--------------------------------
person  :  93.41753125190735  :  [139, 152, 320, 542]
--------------------------------
person  :  74.16782975196838  :  [235, 181, 443, 546]
--------------------------------
person  :  74.15052056312561  :  [420, 231, 640, 559]
--------------------------------
person  :  67.9512619972229  :  [532, 264, 752, 552]
--------------------------------
```

```
person  :  69.05569434165955  :  [694, 300, 885, 561]
--------------------------------
person  :  27.401739358901978  :  [348, 0, 428, 330]
--------------------------------
sports ball  :  32.40821361541748  :  [408, 357, 448, 464]
--------------------------------
```

# ImageAI : Image Prediction

ImageAI proporciona 4 algoritmos y tipos de modelos diferentes para realizar predicciones de imágenes. Los 4 algoritmos proporcionados para la predicción de imágenes incluyen MobileNetV2, ResNet50, InceptionV3 y DenseNet121. Cada uno de estos algoritmos tiene archivos de modelo individuales que utilizaremos según el algoritmo.

# ResNet50, resnet50_imagenet_tf.2.0.h5

```
from imageai.Classification import ImageClassification
import os

execution_path = os.getcwd()

prediction = ImageClassification()
prediction.setModelTypeAsResNet50()
prediction.setModelPath(os.path.join(execution_path, "resnet50_imagenet_tf.2.0.h5"))
prediction.loadModel()

predictions, probabilities = prediction.classifyImage(os.path.join(execution_path, "data-i
for eachPrediction, eachProbability in zip(predictions, probabilities):
    print(eachPrediction , " : " , eachProbability)
```

```
monitor  :  19.894567131996155
prayer_rug  :  10.945137590169907
theater_curtain  :  9.362854808568954
scoreboard  :  7.561834156513214
mosque  :  6.818679720163345
```

```
from imageai.Classification import ImageClassification
import os

execution_path = os.getcwd()

prediction = ImageClassification()
prediction.setModelTypeAsResNet50()
prediction.setModelPath(os.path.join(execution_path, "resnet50_imagenet_tf.2.0.h5"))
prediction.loadModel()

predictions, probabilities = prediction.classifyImage(os.path.join(execution_path, "data-i
for eachPrediction, eachProbability in zip(predictions, probabilities):
```

```
    print(eachPrediction , " : " , eachProbability)
```

```
maze  :  99.20275807380676
tennis_ball  :  0.086994533194229
racket  :  0.050797645235434175
balloon  :  0.0499393732752651
scoreboard  :  0.03912507090717554
```

## DenseNet121, DenseNet-BC-121-32.h5

```
from imageai.Classification import ImageClassification
import os

execution_path = os.getcwd()

prediction = ImageClassification()
prediction.setModelTypeAsDenseNet121()
prediction.setModelPath(os.path.join(execution_path, "DenseNet-BC-121-32.h5"))
prediction.loadModel()

predictions, probabilities = prediction.classifyImage(os.path.join(execution_path, "data-i
for eachPrediction, eachProbability in zip(predictions, probabilities):
    print(eachPrediction , " : " , eachProbability)
```

```
aircraft_carrier  :  39.39239680767059
dam  :  10.03769189119339
prayer_rug  :  8.571392297744751
palace  :  4.591831192374229
dome  :  3.243429586291313
```

```
from imageai.Classification import ImageClassification
import os

execution_path = os.getcwd()

prediction = ImageClassification()
prediction.setModelTypeAsDenseNet121()
prediction.setModelPath(os.path.join(execution_path, "DenseNet-BC-121-32.h5"))
prediction.loadModel()

predictions, probabilities = prediction.classifyImage(os.path.join(execution_path, "data-i
for eachPrediction, eachProbability in zip(predictions, probabilities):
    print(eachPrediction , " : " , eachProbability)
```

```
maze  :  88.11689019203186
tennis_ball  :  3.771398216485977
scoreboard  :  2.0240701735019684
racket  :  1.9683513790369034
sundial  :  0.920239370316267
```

## MobileNetV2, mobilenet_v2.h5

```
from imageai.Classification import ImageClassification
import os

execution_path = os.getcwd()

prediction = ImageClassification()
prediction.setModelTypeAsMobileNetV2()
prediction.setModelPath(os.path.join(execution_path, "mobilenet_v2.h5"))
prediction.loadModel()

predictions, probabilities = prediction.classifyImage(os.path.join(execution_path, "data-i
for eachPrediction, eachProbability in zip(predictions, probabilities):
    print(eachPrediction , " : " , eachProbability)
```

```
palace  :  9.516901522874832
steel_arch_bridge  :  8.947718143463135
crane  :  7.617664337158203
pier  :  5.1558718085289
scoreboard  :  4.099031537771225
```

```
from imageai.Classification import ImageClassification
import os

execution_path = os.getcwd()

prediction = ImageClassification()
prediction.setModelTypeAsMobileNetV2()
prediction.setModelPath(os.path.join(execution_path, "mobilenet_v2.h5"))
prediction.loadModel()

predictions, probabilities = prediction.classifyImage(os.path.join(execution_path, "data-i
for eachPrediction, eachProbability in zip(predictions, probabilities):
    print(eachPrediction , " : " , eachProbability)
```

```
maze  :  17.85295158624649
velvet  :  10.146285593509674
theater_curtain  :  4.89390641450882
prayer_rug  :  3.768305480480194
quilt  :  3.539939597249031
```

## InceptionV3, inception_v3_weights_tf_dim_ordering_tf_kernels.h5

```
from imageai.Classification import ImageClassification
import os

execution_path = os.getcwd()

prediction = ImageClassification()
prediction.setModelTypeAsInceptionV3()
prediction.setModelPath(os.path.join(execution_path, "inception_v3_weights_tf_dim_ordering
prediction.loadModel()
```

```
predictions, probabilities = prediction.classifyImage(os.path.join(execution_path, "data-i
for eachPrediction, eachProbability in zip(predictions, probabilities):
    print(eachPrediction , " : " , eachProbability)
```

```
palace  :  88.45898509025574
pier  :  2.5890175253152847
patio  :  2.1082576364278793
monastery  :  1.5544397756457329
stage  :  0.8047371171414852
```

```
from imageai.Classification import ImageClassification
import os

execution_path = os.getcwd()

prediction = ImageClassification()
prediction.setModelTypeAsInceptionV3()
prediction.setModelPath(os.path.join(execution_path, "inception_v3_weights_tf_dim_ordering
prediction.loadModel()

predictions, probabilities = prediction.classifyImage(os.path.join(execution_path, "data-i
for eachPrediction, eachProbability in zip(predictions, probabilities):
    print(eachPrediction , " : " , eachProbability)
```

```
racket  :  74.63094592094421
basketball  :  9.298282861709595
bow  :  5.14446459710598
tennis_ball  :  5.029075965285301
ping-pong_ball  :  1.8615514039993286
```

# Speed Detection

ImageAI nos ofrece como en el caso de clasificación de imágenes diferentes velocidades de lectura y procesado, siendo 'normal' la predefinidad y standart y otras velocidades como fast, faster o fastest que le deberemos de pasar como argumento a la llamada prediction.loadModel(classification_speed) y puede reducir la velocidad de detención entre un 20% - 60%.

Lo óptimo es aplicar las diferentes velocidades de procesado con los modelos de DenseNet o InceptionV3 ya que son los que tienen mejor precisión.

```
from imageai.Classification import ImageClassification
import os

execution_path = os.getcwd()

prediction = ImageClassification()
prediction.setModelTypeAsResNet50()
prediction.setModelPath(os.path.join(execution_path, "resnet50_imagenet_tf.2.0.h5"))
prediction.loadModel(classification_speed='faster')
```

```
predictions, probabilities = prediction.classifyImage(os.path.join(execution_path, "data-i
for eachPrediction, eachProbability in zip(predictions, probabilities):
    print(eachPrediction , " : " , eachProbability)
```

```
racer  :  98.94137382507324
tow_truck  :  0.7607404608279467
sports_car  :  0.1630593091249466
car_wheel  :  0.07155194180086255
limousine  :  0.05747430259361863
```

```
from imageai.Classification import ImageClassification
import os

execution_path = os.getcwd()

prediction = ImageClassification()
prediction.setModelTypeAsDenseNet121()
prediction.setModelPath(os.path.join(execution_path, "DenseNet-BC-121-32.h5"))
prediction.loadModel(classification_speed='faster')

predictions, probabilities = prediction.classifyImage(os.path.join(execution_path, "data-i
for eachPrediction, eachProbability in zip(predictions, probabilities):
    print(eachPrediction , " : " , eachProbability)
```

```
racer  :  95.31360864639282
sports_car  :  4.68636192381382
convertible  :  1.6960991899850342e-05
limousine  :  1.3699837708713858e-06
car_wheel  :  9.223232844135509e-07
```

```
from imageai.Classification import ImageClassification
import os

execution_path = os.getcwd()

prediction = ImageClassification()
prediction.setModelTypeAsInceptionV3()
prediction.setModelPath(os.path.join(execution_path, "inception_v3_weights_tf_dim_ordering
prediction.loadModel(classification_speed='faster')

predictions, probabilities = prediction.classifyImage(os.path.join(execution_path, "data-i
for eachPrediction, eachProbability in zip(predictions, probabilities):
    print(eachPrediction , " : " , eachProbability)
```

```
convertible  :  99.8393714427948
cab  :  0.1386831165291369
grille  :  0.012983562191948295
sports_car  :  0.007201782864285633
beach_wagon  :  0.0009785354450286832
```

## Own example (Fútbol, tenis, baloncesto predicción)

Ahora vamos a probar como hemos hecho antes con la imagen de la plantilla de la selección.

```
from imageai.Classification import ImageClassification
import os

execution_path = os.getcwd()

prediction = ImageClassification()
prediction.setModelTypeAsResNet50()
prediction.setModelPath(os.path.join(execution_path, "resnet50_imagenet_tf.2.0.h5"))
prediction.loadModel()

predictions, probabilities = prediction.classifyImage(os.path.join(execution_path, "data-i
for eachPrediction, eachProbability in zip(predictions, probabilities):
    print(eachPrediction , " : " , eachProbability)
```

```
swimming_trunks  :  65.2655839920044
bikini  :  16.491419076919556
torch  :  12.885771691799164
sorrel  :  1.129803340882063
maillot  :  0.9406358934938908
```

```
from imageai.Classification import ImageClassification
import os

execution_path = os.getcwd()

prediction = ImageClassification()
prediction.setModelTypeAsInceptionV3()
prediction.setModelPath(os.path.join(execution_path, "inception_v3_weights_tf_dim_ordering
prediction.loadModel()

predictions, probabilities = prediction.classifyImage(os.path.join(execution_path, "data-i
for eachPrediction, eachProbability in zip(predictions, probabilities):
    print(eachPrediction , " : " , eachProbability)
```

```
WARNING:tensorflow:5 out of the last 5 calls to <function Model.make_predict_functior
passenger_car  :  48.15164506435394
trolleybus  :  43.741753697395325
minibus  :  4.695677012205124
recreational_vehicle  :  1.5537434257566929
fire_engine  :  0.42165075428783894
```

```
from imageai.Classification import ImageClassification
import os

execution_path = os.getcwd()

prediction = ImageClassification()
prediction.setModelTypeAsDenseNet121()
prediction.setModelPath(os.path.join(execution_path, "DenseNet-BC-121-32.h5"))
prediction.loadModel()
```

```
predictions, probabilities = prediction.classifyImage(os.path.join(execution_path, "data-i
for eachPrediction, eachProbability in zip(predictions, probabilities):
    print(eachPrediction , " : " , eachProbability)
```

```
passenger_car  :  39.39520716667175
school_bus  :  24.116992950439453
trolleybus  :  20.47085165977478
minibus  :  4.694885015487671
trailer_truck  :  2.7369460090994835
```

# ImageAI : Video Object Detection, Tracking and Analysis

Durante toda la sección se va evitar el resultado de la consola, ya que se tratan de ejecuciones muy detalladas y ocupan una gran cantidad de texto.

ImageAI proporciona métodos convenientes, flexibles y potentes para realizar la detección de objetos en videos. La clase de detección de objetos de video proporcionada solo es compatible con RetinaNet, YOLOv3 y TinyYOLOv3. Esta versión de ImageAI proporciona funciones de detección de objetos de video de calidad comercial, que incluyen, entre otras, entradas de dispositivo / cámara IP, por cuadro, por segundo, por minuto y análisis de video completo para almacenar en bases de datos y / o visualizaciones en tiempo real.

*La detección de objetos de video es una tarea de computación intensiva, se recomienda realizar el experimento en una computadora con una GPU NVIDIA y la versión de GPU de Tensorflow instalada. Usamos las GPU de Google Colab.*

```
pip install keras_resnet
```

```
Requirement already satisfied: keras_resnet in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: keras>=2.2.4 in /usr/local/lib/python3.7/dist-packages
```

# RetinaNet

El tiempo de ejecución del vídeo traffic MODELO RESNET en velocidad fast y 10fps es de: 2h 55m 39s con un total de 2115 frames

```
from imageai.Detection import VideoObjectDetection
import os

execution_path = os.getcwd()

detector = VideoObjectDetection()
detector.setModelTypeAsRetinaNet()
```

```
detector.setModelPath( os.path.join(execution_path , "resnet50_coco_best_v2.1.0.h5"))
detector.loadModel()

custom_objects = detector.CustomObjects(person=True) #Tan solo vamos a detectar jugadores

video_path = detector.detectCustomObjectsFromVideo(
                custom_objects=custom_objects,
                input_file_path=os.path.join(execution_path, "data-videos/Goliniestacorto.
                output_file_path=os.path.join(execution_path, "goliniestacustom_resnet"),
                frames_per_second=20, log_progress=True)
print(video_path)



from imageai.Detection import VideoObjectDetection
import os

execution_path = os.getcwd()

detector = VideoObjectDetection()
detector.setModelTypeAsRetinaNet()
detector.setModelPath( os.path.join(execution_path , "resnet50_coco_best_v2.1.0.h5"))
detector.loadModel()

video_path = detector.detectObjectsFromVideo(input_file_path=os.path.join(execution_path,
                                output_file_path=os.path.join(execution_path, "goliniestan
                                , frames_per_second=20, log_progress=True)
print(video_path)
```

```
WARNING:tensorflow:No training configuration found in the save file, so the model w
WARNING:tensorflow:No training configuration found in the save file, so the model w
Processing Frame :  1
Processing Frame :  2
Processing Frame :  3
Processing Frame :  4
Processing Frame :  5
Processing Frame :  6
Processing Frame :  7
Processing Frame :  8
Processing Frame :  9
Processing Frame :  10
Processing Frame :  11
Processing Frame :  12
Processing Frame :  13
Processing Frame :  14
Processing Frame :  15
Processing Frame :  16
Processing Frame :  17
Processing Frame :  18
Processing Frame :  19
Processing Frame :  20
Processing Frame :  21
Processing Frame :  22
Processing Frame :  23
Processing Frame :  24
Processing Frame :  25
Processing Frame :  26
Processing Frame :  27
```

```
Processing Frame :  28
Processing Frame :  29
Processing Frame :  30
Processing Frame :  31
Processing Frame :  32
Processing Frame :  33
Processing Frame :  34
Processing Frame :  35
Processing Frame :  36
Processing Frame :  37
Processing Frame :  38
Processing Frame :  39
Processing Frame :  40
Processing Frame :  41
Processing Frame :  42
Processing Frame :  43
Processing Frame :  44
Processing Frame :  45
Processing Frame :  46
Processing Frame :  47
Processing Frame :  48
Processing Frame :  49
Processing Frame :  50
Processing Frame :  51
Processing Frame :  52
Processing Frame :  53
Processing Frame :  54
Processing Frame :  55
Processing Frame :  56
```

El tiempo de ejecución del vídeo MODELO RESNET goliniesta en velocidad normal y 20fps es de: 2h 15m 39s con un total de 1022 frames

```
from imageai.Detection import VideoObjectDetection
import os

execution_path = os.getcwd()

detector = VideoObjectDetection()
detector.setModelTypeAsRetinaNet()
detector.setModelPath( os.path.join(execution_path , "resnet50_coco_best_v2.1.0.h5"))
detector.loadModel(detection_speed="fast")

video_path = detector.detectObjectsFromVideo(input_file_path=os.path.join(execution_path,
                                output_file_path=os.path.join(execution_path, "goliniestaf
                                , frames_per_second=20, log_progress=True)
print(video_path)
```

```
WARNING:tensorflow:No training configuration found in the save file, so the model w
Processing Frame :  1
Processing Frame :  2
Processing Frame :  3
Processing Frame :  4
Processing Frame :  5
Processing Frame :  6
Processing Frame :  7
Processing Frame :  8
```

```
Processing Frame :  9
Processing Frame :  10
Processing Frame :  11
Processing Frame :  12
Processing Frame :  13
Processing Frame :  14
Processing Frame :  15
Processing Frame :  16
Processing Frame :  17
Processing Frame :  18
Processing Frame :  19
Processing Frame :  20
Processing Frame :  21
Processing Frame :  22
Processing Frame :  23
Processing Frame :  24
Processing Frame :  25
Processing Frame :  26
Processing Frame :  27
Processing Frame :  28
Processing Frame :  29
Processing Frame :  30
Processing Frame :  31
Processing Frame :  32
Processing Frame :  33
Processing Frame :  34
Processing Frame :  35
Processing Frame :  36
Processing Frame :  37
Processing Frame :  38
Processing Frame :  39
Processing Frame :  40
Processing Frame :  41
Processing Frame :  42
Processing Frame :  43
Processing Frame :  44
Processing Frame :  45
Processing Frame :  46
Processing Frame :  47
Processing Frame :  48
Processing Frame :  49
Processing Frame :  50
Processing Frame :  51
Processing Frame :  52
Processing Frame :  53
Processing Frame :  54
Processing Frame :  55
Processing Frame :  56
Processing Frame :  57
```

El tiempo de ejecución del vídeo goliniesta MODELO RESNET en velocidad fast y 20fps es de: 1h 50m 19s con un total de 1022 frames

```
from imageai.Detection import VideoObjectDetection
import os

execution_path = os.getcwd()
```

```
detector = VideoObjectDetection()
detector.setModelTypeAsRetinaNet()
detector.setModelPath( os.path.join(execution_path , "resnet50_coco_best_v2.1.0.h5"))
detector.loadModel(detection_speed="fastest")

video_path = detector.detectObjectsFromVideo(input_file_path=os.path.join(execution_path,
                                output_file_path=os.path.join(execution_path, "goliniestaf
                                , frames_per_second=20, log_progress=True)
print(video_path)
```

```
WARNING:tensorflow:No training configuration found in the save file, so the model w
Processing Frame :  1
Processing Frame :  2
Processing Frame :  3
Processing Frame :  4
Processing Frame :  5
Processing Frame :  6
Processing Frame :  7
Processing Frame :  8
Processing Frame :  9
Processing Frame :  10
Processing Frame :  11
Processing Frame :  12
Processing Frame :  13
Processing Frame :  14
Processing Frame :  15
Processing Frame :  16
Processing Frame :  17
Processing Frame :  18
Processing Frame :  19
Processing Frame :  20
Processing Frame :  21
Processing Frame :  22
Processing Frame :  23
Processing Frame :  24
Processing Frame :  25
Processing Frame :  26
Processing Frame :  27
Processing Frame :  28
Processing Frame :  29
Processing Frame :  30
Processing Frame :  31
Processing Frame :  32
Processing Frame :  33
Processing Frame :  34
Processing Frame :  35
Processing Frame :  36
Processing Frame :  37
Processing Frame :  38
Processing Frame :  39
Processing Frame :  40
Processing Frame :  42
Processing Frame :  43
Processing Frame :  44
Processing Frame :  45
Processing Frame :  46
Processing Frame :  47
Processing Frame :  48
Processing Frame :  49
Processing Frame :  50
```

```
Processing Frame :  51
Processing Frame :  52
Processing Frame :  53
Processing Frame :  54
Processing Frame :  55
Processing Frame :  56
Processing Frame :  57
Processing Frame :  58
```

El tiempo de ejecución del vídeo goliniesta MODELO RESNET en velocidad fastest y 20fps es de: 1h 24m 15s con un total de 1022 frames

```
from imageai.Detection import VideoObjectDetection
import os

execution_path = os.getcwd()

detector = VideoObjectDetection()
detector.setModelTypeAsRetinaNet()
detector.setModelPath( os.path.join(execution_path , "resnet50_coco_best_v2.1.0.h5"))
detector.loadModel(detection_speed="flash")

video_path = detector.detectObjectsFromVideo(input_file_path=os.path.join(execution_path,
                                output_file_path=os.path.join(execution_path, "goliniestaf
                                , frames_per_second=20, log_progress=True)
print(video_path)
```

```
WARNING:tensorflow:No training configuration found in the save file, so the model w
Processing Frame :  1
Processing Frame :  2
Processing Frame :  3
Processing Frame :  4
Processing Frame :  5
Processing Frame :  6
Processing Frame :  7
Processing Frame :  8
Processing Frame :  9
Processing Frame :  10
Processing Frame :  11
Processing Frame :  12
Processing Frame :  13
Processing Frame :  14
Processing Frame :  15
Processing Frame :  16
Processing Frame :  17
Processing Frame :  18
Processing Frame :  19
Processing Frame :  20
Processing Frame :  21
Processing Frame :  22
Processing Frame :  23
Processing Frame :  24
Processing Frame :  25
Processing Frame :  26
Processing Frame :  27
Processing Frame :  28
```

```
Processing Frame :  29
Processing Frame :  30
Processing Frame :  31
Processing Frame :  32
Processing Frame :  33
Processing Frame :  34
Processing Frame :  35
Processing Frame :  36
Processing Frame :  37
Processing Frame :  38
Processing Frame :  39
Processing Frame :  40
Processing Frame :  41
Processing Frame :  42
Processing Frame :  43
Processing Frame :  44
Processing Frame :  45
Processing Frame :  46
Processing Frame :  47
Processing Frame :  48
Processing Frame :  49
Processing Frame :  50
Processing Frame :  51
Processing Frame :  52
Processing Frame :  53
Processing Frame :  54
Processing Frame :  55
Processing Frame :  56
Processing Frame :  57
```

El tiempo de ejecución del vídeo goliniesta MODELO RESNET en velocidad flash y 20fps es de: 1h 03m 54s con un total de 1022 frames

## Yolo video

```
from imageai.Detection import VideoObjectDetection
import os

execution_path = os.getcwd()

detector = VideoObjectDetection()
detector.setModelTypeAsYOLOv3()
detector.setModelPath( os.path.join(execution_path , "yolo.h5"))
detector.loadModel()

video_path = detector.detectObjectsFromVideo(input_file_path=os.path.join(execution_path,
                                output_file_path=os.path.join(execution_path, "goliniestan
                                , frames_per_second=20, log_progress=True)
```

```
Processing Frame :  1
Processing Frame :  2
Processing Frame :  3
Processing Frame :  4
Processing Frame :  5
```

```
Processing Frame :  6
Processing Frame :  7
Processing Frame :  8
Processing Frame :  9
Processing Frame :  10
Processing Frame :  11
Processing Frame :  12
Processing Frame :  13
Processing Frame :  14
Processing Frame :  15
Processing Frame :  16
Processing Frame :  17
Processing Frame :  18
Processing Frame :  19
Processing Frame :  20
Processing Frame :  21
Processing Frame :  22
Processing Frame :  23
Processing Frame :  24
Processing Frame :  25
Processing Frame :  26
Processing Frame :  27
Processing Frame :  28
Processing Frame :  29
Processing Frame :  30
Processing Frame :  31
Processing Frame :  32
Processing Frame :  33
Processing Frame :  34
Processing Frame :  35
Processing Frame :  36
Processing Frame :  37
Processing Frame :  38
Processing Frame :  39
Processing Frame :  40
Processing Frame :  41
Processing Frame :  42
Processing Frame :  43
Processing Frame :  44
Processing Frame :  45
Processing Frame :  46
Processing Frame :  47
Processing Frame :  48
Processing Frame :  49
Processing Frame :  50
Processing Frame :  51
Processing Frame :  52
Processing Frame :  53
Processing Frame :  54
Processing Frame :  55
Processing Frame :  56
Processing Frame :  57
Processing Frame :  58
Processing Frame :  59
```

El tiempo de ejecución del vídeo goliniesta MODELO YOLO en velocidad normal y 20fps es de: 2h 3m 44s con un total de 1022 frames

```
from imageai.Detection import VideoObjectDetection
import os
```

```
import os

execution_path = os.getcwd()

detector = VideoObjectDetection()
detector.setModelTypeAsYOLOv3()
detector.setModelPath( os.path.join(execution_path , "yolo.h5"))
detector.loadModel(detection_speed="fast")

video_path = detector.detectObjectsFromVideo(input_file_path=os.path.join(execution_path,
                                output_file_path=os.path.join(execution_path, "goliniestaf
                                , frames_per_second=20, log_progress=True)
```

```
Processing Frame :  1
Processing Frame :  2
Processing Frame :  3
Processing Frame :  4
Processing Frame :  5
Processing Frame :  6
Processing Frame :  7
Processing Frame :  8
Processing Frame :  9
Processing Frame :  10
Processing Frame :  11
Processing Frame :  12
Processing Frame :  13
Processing Frame :  14
Processing Frame :  15
Processing Frame :  16
Processing Frame :  17
Processing Frame :  18
Processing Frame :  19
Processing Frame :  20
Processing Frame :  21
Processing Frame :  22
Processing Frame :  23
Processing Frame :  24
Processing Frame :  25
Processing Frame :  26
Processing Frame :  27
Processing Frame :  28
Processing Frame :  29
Processing Frame :  30
Processing Frame :  31
Processing Frame :  32
Processing Frame :  33
Processing Frame :  34
Processing Frame :  35
Processing Frame :  36
Processing Frame :  37
Processing Frame :  38
Processing Frame :  39
Processing Frame :  40
Processing Frame :  41
Processing Frame :  42
Processing Frame :  43
Processing Frame :  44
Processing Frame :  45
Processing Frame :  46
Processing Frame :  47
```

```
Processing Frame :  48
Processing Frame :  49
Processing Frame :  50
Processing Frame :  51
Processing Frame :  52
Processing Frame :  53
Processing Frame :  54
Processing Frame :  55
Processing Frame :  56
Processing Frame :  57
Processing Frame :  58
Processing Frame :  59
```

El tiempo de ejecución del vídeo goliniesta MODELO YOLO en velocidad fast y 20fps es de: 1h 7m 30s con un total de 1022 frames

```
from imageai.Detection import VideoObjectDetection
import os

execution_path = os.getcwd()

detector = VideoObjectDetection()
detector.setModelTypeAsYOLOv3()
detector.setModelPath( os.path.join(execution_path , "yolo.h5"))
detector.loadModel(detection_speed="fastest")

video_path = detector.detectObjectsFromVideo(input_file_path=os.path.join(execution_path,
                                output_file_path=os.path.join(execution_path, "goliniestaf
                                , frames_per_second=20, log_progress=True)
```

```
Processing Frame :  1
Processing Frame :  2
Processing Frame :  3
Processing Frame :  4
Processing Frame :  5
Processing Frame :  6
Processing Frame :  7
Processing Frame :  8
Processing Frame :  9
Processing Frame :  10
Processing Frame :  11
Processing Frame :  12
Processing Frame :  13
Processing Frame :  14
Processing Frame :  15
Processing Frame :  16
Processing Frame :  17
Processing Frame :  18
Processing Frame :  19
Processing Frame :  20
Processing Frame :  21
Processing Frame :  22
Processing Frame :  23
Processing Frame :  24
Processing Frame :  25
Processing Frame :  26
Processing Frame :  27
Processing Frame :  28
```

```
Processing Frame :  29
Processing Frame :  30
Processing Frame :  31
Processing Frame :  32
Processing Frame :  33
Processing Frame :  34
Processing Frame :  35
Processing Frame :  36
Processing Frame :  37
Processing Frame :  38
Processing Frame :  39
Processing Frame :  40
Processing Frame :  41
Processing Frame :  42
Processing Frame :  43
Processing Frame :  44
Processing Frame :  45
Processing Frame :  46
Processing Frame :  47
Processing Frame :  48
Processing Frame :  49
Processing Frame :  50
Processing Frame :  51
Processing Frame :  52
Processing Frame :  53
Processing Frame :  54
Processing Frame :  55
Processing Frame :  56
Processing Frame :  57
Processing Frame :  58
Processing Frame :  59
```

El tiempo de ejecución del vídeo goliniesta MODELO YOLO en velocidad fastest y 20fps es de: 11m 18s con un total de 1022 frames

```
from imageai.Detection import VideoObjectDetection
import os

execution_path = os.getcwd()

detector = VideoObjectDetection()
detector.setModelTypeAsYOLOv3()
detector.setModelPath( os.path.join(execution_path , "yolo.h5"))
detector.loadModel(detection_speed="flash")

video_path = detector.detectObjectsFromVideo(input_file_path=os.path.join(execution_path,
                                output_file_path=os.path.join(execution_path, "goliniestaf
                                , frames_per_second=20, log_progress=True)
```

```
Processing Frame :  1
Processing Frame :  2
Processing Frame :  3
Processing Frame :  4
Processing Frame :  5
Processing Frame :  6
Processing Frame :  7
Processing Frame :  8
```

```
Processing Frame :  9
Processing Frame :  10
Processing Frame :  11
Processing Frame :  12
Processing Frame :  13
Processing Frame :  14
Processing Frame :  15
Processing Frame :  16
Processing Frame :  17
Processing Frame :  18
Processing Frame :  19
Processing Frame :  20
Processing Frame :  21
Processing Frame :  22
Processing Frame :  23
Processing Frame :  24
Processing Frame :  25
Processing Frame :  26
Processing Frame :  27
Processing Frame :  28
Processing Frame :  29
Processing Frame :  30
Processing Frame :  31
Processing Frame :  32
Processing Frame :  33
Processing Frame :  34
Processing Frame :  35
Processing Frame :  36
Processing Frame :  37
Processing Frame :  38
Processing Frame :  39
Processing Frame :  40
Processing Frame :  41
Processing Frame :  42
Processing Frame :  43
Processing Frame :  44
Processing Frame :  45
Processing Frame :  46
Processing Frame :  47
Processing Frame :  48
Processing Frame :  49
Processing Frame :  50
Processing Frame :  51
Processing Frame :  52
Processing Frame :  53
Processing Frame :  54
Processing Frame :  55
Processing Frame :  56
Processing Frame :  57
Processing Frame :  58
Processing Frame :  59
```

El tiempo de ejecución del vídeo goliniesta MODELO YOLO en velocidad flash y 20fps es de: 7m 21s con un total de 1022 frames

## Yolo-Tiny video

Aunque no da errores el video no se muestra correctamente y esto puede ser porque yolo-tiny parece que no usa tensorflow

```
from imageai.Detection import VideoObjectDetection
import os

execution_path = os.getcwd()

detector = VideoObjectDetection()
detector.setModelTypeAsTinyYOLOv3()
detector.setModelPath( os.path.join(execution_path , "yolo-tiny.h5"))
detector.loadModel()

video_path = detector.detectObjectsFromVideo(input_file_path=os.path.join(execution_path,
                                output_file_path=os.path.join(execution_path, "traffic_det
                                , frames_per_second=20, log_progress=True)
print(video_path)
```

```
     Processing Frame :  1
     Processing Frame :  2
     Processing Frame :  3
     Processing Frame :  4
     Processing Frame :  5
     Processing Frame :  6
     Processing Frame :  7
     Processing Frame :  8
     Processing Frame :  9
     Processing Frame :  10
     Processing Frame :  11
     Processing Frame :  12
     Processing Frame :  13
     Processing Frame :  14
     Processing Frame :  15
     Processing Frame :  16
     Processing Frame :  17
     Processing Frame :  18
     Processing Frame :  19
     Processing Frame :  20
     Processing Frame :  21
     Processing Frame :  22
     Processing Frame :  23
     Processing Frame :  24
     Processing Frame :  25
     Processing Frame :  26
     Processing Frame :  27
     Processing Frame :  28
     Processing Frame :  29
     Processing Frame :  30
     Processing Frame :  31
     Processing Frame :  32
     Processing Frame :  33
     Processing Frame :  34
     Processing Frame :  35
     Processing Frame :  36
     Processing Frame :  37
     Processing Frame :  38
     Processing Frame :  39
```

```
Processing Frame :  40
Processing Frame :  41
Processing Frame :  42
Processing Frame :  43
Processing Frame :  44
Processing Frame :  45
Processing Frame :  46
Processing Frame :  47
Processing Frame :  48
Processing Frame :  49
Processing Frame :  50
Processing Frame :  51
Processing Frame :  52
Processing Frame :  53
Processing Frame :  54
Processing Frame :  55
Processing Frame :  56
Processing Frame :  57
Processing Frame :  58
Processing Frame :  59
```

```
from imageai.Detection import VideoObjectDetection
import os

execution_path = os.getcwd()

detector = VideoObjectDetection()
detector.setModelTypeAsTinyYOLOv3()
detector.setModelPath( os.path.join(execution_path , "yolo-tiny.h5"))
detector.loadModel()

video_path = detector.detectObjectsFromVideo(input_file_path=os.path.join(execution_path,
                                output_file_path=os.path.join(execution_path, "goliniestan
                                , frames_per_second=20, log_progress=True)
print(video_path)
```

```
Processing Frame :  1
Processing Frame :  2
Processing Frame :  3
Processing Frame :  4
Processing Frame :  5
Processing Frame :  6
Processing Frame :  7
Processing Frame :  8
Processing Frame :  9
Processing Frame :  10
Processing Frame :  11
Processing Frame :  12
Processing Frame :  13
Processing Frame :  14
Processing Frame :  15
Processing Frame :  16
Processing Frame :  17
Processing Frame :  18
Processing Frame :  19
Processing Frame :  20
Processing Frame :  21
Processing Frame :  22
Processing Frame :  23
```

```
Processing Frame :  24
Processing Frame :  25
Processing Frame :  26
Processing Frame :  27
Processing Frame :  28
Processing Frame :  29
Processing Frame :  30
Processing Frame :  31
Processing Frame :  32
Processing Frame :  33
Processing Frame :  34
Processing Frame :  35
Processing Frame :  36
Processing Frame :  37
Processing Frame :  38
Processing Frame :  39
Processing Frame :  40
Processing Frame :  41
Processing Frame :  42
Processing Frame :  43
Processing Frame :  44
Processing Frame :  45
Processing Frame :  46
Processing Frame :  47
Processing Frame :  48
Processing Frame :  49
Processing Frame :  50
Processing Frame :  51
Processing Frame :  52
Processing Frame :  53
Processing Frame :  54
Processing Frame :  55
Processing Frame :  56
Processing Frame :  57
Processing Frame :  58
Processing Frame :  59
```

El tiempo de ejecución del vídeo goliniesta MODELO TINY en velocidad normal y 20fps es de: 1h 27m 06s con un total de 1022 frames (tiny)

```
from imageai.Detection import VideoObjectDetection
import os

execution_path = os.getcwd()

detector = VideoObjectDetection()
detector.setModelTypeAsTinyYOLOv3()
detector.setModelPath( os.path.join(execution_path , "yolo-tiny.h5"))
detector.loadModel(detection_speed="fast")

video_path = detector.detectObjectsFromVideo(input_file_path=os.path.join(execution_path,
                                output_file_path=os.path.join(execution_path, "goliniestaf
                                , frames_per_second=20, log_progress=True)
print(video_path)
```

```
Processing Frame :  1
Processing Frame :  2
```

```
Processing Frame :  3
Processing Frame :  4
Processing Frame :  5
Processing Frame :  6
Processing Frame :  7
Processing Frame :  8
Processing Frame :  9
Processing Frame :  10
Processing Frame :  11
Processing Frame :  12
Processing Frame :  13
Processing Frame :  14
Processing Frame :  15
Processing Frame :  16
Processing Frame :  17
Processing Frame :  18
Processing Frame :  19
Processing Frame :  20
Processing Frame :  21
Processing Frame :  22
Processing Frame :  23
Processing Frame :  24
Processing Frame :  25
Processing Frame :  26
Processing Frame :  27
Processing Frame :  28
Processing Frame :  29
Processing Frame :  30
Processing Frame :  31
Processing Frame :  32
Processing Frame :  33
Processing Frame :  34
Processing Frame :  35
Processing Frame :  36
Processing Frame :  37
Processing Frame :  38
Processing Frame :  39
Processing Frame :  40
Processing Frame :  41
Processing Frame :  42
Processing Frame :  43
Processing Frame :  44
Processing Frame :  45
Processing Frame :  46
Processing Frame :  47
Processing Frame :  48
Processing Frame :  49
Processing Frame :  50
Processing Frame :  51
Processing Frame :  52
Processing Frame :  53
Processing Frame :  54
Processing Frame :  55
Processing Frame :  56
Processing Frame :  57
Processing Frame :  58
Processing Frame :  59
```

El tiempo de ejecución del vídeo goliniesta MODELO TINY en velocidad fast y 20fps es de: 42m 32s con un total de 1022 frames (tiny)

```
from imageai.Detection import VideoObjectDetection
import os

execution_path = os.getcwd()

detector = VideoObjectDetection()
detector.setModelTypeAsTinyYOLOv3()
detector.setModelPath( os.path.join(execution_path , "yolo-tiny.h5"))
detector.loadModel(detection_speed="fastest")

video_path = detector.detectObjectsFromVideo(input_file_path=os.path.join(execution_path,
                                output_file_path=os.path.join(execution_path, "goliniestaf
                                , frames_per_second=20, log_progress=True)
print(video_path)
```

```
     Processing Frame :  1
     Processing Frame :  2
     Processing Frame :  3
     Processing Frame :  4
     Processing Frame :  5
     Processing Frame :  6
     Processing Frame :  7
     Processing Frame :  8
     Processing Frame :  9
     Processing Frame :  10
     Processing Frame :  11
     Processing Frame :  12
     Processing Frame :  13
     Processing Frame :  14
     Processing Frame :  15
     Processing Frame :  16
     Processing Frame :  17
     Processing Frame :  18
     Processing Frame :  19
     Processing Frame :  20
     Processing Frame :  21
     Processing Frame :  22
     Processing Frame :  23
     Processing Frame :  24
     Processing Frame :  25
     Processing Frame :  26
     Processing Frame :  27
     Processing Frame :  28
     Processing Frame :  29
     Processing Frame :  30
     Processing Frame :  31
     Processing Frame :  32
     Processing Frame :  33
     Processing Frame :  34
     Processing Frame :  35
     Processing Frame :  36
     Processing Frame :  37
     Processing Frame :  38
     Processing Frame :  39
     Processing Frame :  40
     Processing Frame :  41
     Processing Frame :  42
     Processing Frame :  43
```

```
Processing Frame :  44
Processing Frame :  45
Processing Frame :  46
Processing Frame :  47
Processing Frame :  48
Processing Frame :  49
Processing Frame :  50
Processing Frame :  51
Processing Frame :  52
Processing Frame :  53
Processing Frame :  54
Processing Frame :  55
Processing Frame :  56
Processing Frame :  57
Processing Frame :  58
Processing Frame :  59
```

El tiempo de ejecución del vídeo goliniesta MODELO TINY en velocidad fastest y 20fps es de: 13m 36s con un total de 1022 frames (tiny)

```
from imageai.Detection import VideoObjectDetection
import os

execution_path = os.getcwd()

detector = VideoObjectDetection()
detector.setModelTypeAsTinyYOLOv3()
detector.setModelPath( os.path.join(execution_path , "yolo-tiny.h5"))
detector.loadModel(detection_speed="flash")

video_path = detector.detectObjectsFromVideo(input_file_path=os.path.join(execution_path,
                                output_file_path=os.path.join(execution_path, "goliniestaf
                                , frames_per_second=20, log_progress=True)
print(video_path)
```

```
Processing Frame :  1
Processing Frame :  2
Processing Frame :  3
Processing Frame :  4
Processing Frame :  5
Processing Frame :  6
Processing Frame :  7
Processing Frame :  8
Processing Frame :  9
Processing Frame :  10
Processing Frame :  11
Processing Frame :  12
Processing Frame :  13
Processing Frame :  14
Processing Frame :  15
Processing Frame :  16
Processing Frame :  17
Processing Frame :  18
Processing Frame :  19
Processing Frame :  20
Processing Frame :  21
Processing Frame :  22
```

```
Processing Frame :  23
Processing Frame :  24
Processing Frame :  25
Processing Frame :  26
Processing Frame :  27
Processing Frame :  28
Processing Frame :  29
Processing Frame :  30
Processing Frame :  31
Processing Frame :  32
Processing Frame :  33
Processing Frame :  34
Processing Frame :  35
Processing Frame :  36
Processing Frame :  37
Processing Frame :  38
Processing Frame :  39
Processing Frame :  40
Processing Frame :  41
Processing Frame :  42
Processing Frame :  43
Processing Frame :  44
Processing Frame :  45
Processing Frame :  46
Processing Frame :  47
Processing Frame :  48
Processing Frame :  49
Processing Frame :  50
Processing Frame :  51
Processing Frame :  52
Processing Frame :  53
Processing Frame :  54
Processing Frame :  55
Processing Frame :  56
Processing Frame :  57
Processing Frame :  58
Processing Frame :  59
```

El tiempo de ejecución del vídeo goliniesta MODELO TINY en velocidad flash y 20fps es de: 3m 01s con un total de 1022 frames (tiny)

Importamos la clase VideoObjectDetection de la librería en la primera línea, importamos el sistema operativo en la segunda línea y la ruta a la carpeta donde se ejecuta nuestro archivo. Creamos una nueva instancia de la clase VideoObjectDetection, configuramos el tipo de modelo en RetinaNet en la quinta línea, configuramos la ruta del modelo al archivo del modelo RetinaNet que descargamos. Ejecutamos la función detectObjectsFromVideo (). la ruta al nuevo video (sin la extensión, guarda un video .avi por defecto), el número de cuadros por segundo (fps) deseado y la opción de registrar el progreso de la detección en la consola. La función devuelve la ruta al video guardado que contiene cuadros y probabilidades porcentuales representadas en el video.

## Video gol final corto

```
from imageai.Detection import VideoObjectDetection
```

```
import os

execution_path = os.getcwd()

detector = VideoObjectDetection()
detector.setModelTypeAsTinyYOLOv3()
detector.setModelPath( os.path.join(execution_path , "yolo-tiny.h5"))
detector.loadModel()

video_path = detector.detectObjectsFromVideo(input_file_path=os.path.join(execution_path,
                                output_file_path=os.path.join(execution_path, "goliniesta-
                                , frames_per_second=10, log_progress=True)
print(video_path)
```

```
     Processing Frame :  1
     Processing Frame :  2
     Processing Frame :  3
     Processing Frame :  4
     Processing Frame :  5
     Processing Frame :  6
     Processing Frame :  7
     Processing Frame :  8
     Processing Frame :  9
     Processing Frame :  10
     Processing Frame :  11
     Processing Frame :  12
     Processing Frame :  13
     Processing Frame :  14
     Processing Frame :  15
     Processing Frame :  16
     Processing Frame :  17
     Processing Frame :  18
     Processing Frame :  19
     Processing Frame :  20
     Processing Frame :  21
     Processing Frame :  22
     Processing Frame :  23
     Processing Frame :  24
     Processing Frame :  25
     Processing Frame :  26
     Processing Frame :  27
     Processing Frame :  28
     Processing Frame :  29
     Processing Frame :  30
     Processing Frame :  31
     Processing Frame :  32
     Processing Frame :  33
     Processing Frame :  34
     Processing Frame :  35
     Processing Frame :  36
     Processing Frame :  37
     Processing Frame :  38
     Processing Frame :  39
     Processing Frame :  40
     Processing Frame :  41
     Processing Frame :  42
     Processing Frame :  43
     Processing Frame :  44
     Processing Frame :  45
```

```
Processing Frame :  46
Processing Frame :  47
Processing Frame :  48
Processing Frame :  49
Processing Frame :  50
Processing Frame :  51
Processing Frame :  52
Processing Frame :  53
Processing Frame :  54
Processing Frame :  55
Processing Frame :  56
Processing Frame :  57
Processing Frame :  58
Processing Frame :  59
```

# Camera / Live Stream Video Detection

## Yolo-tiny

```
from imageai.Detection import VideoObjectDetection
import os
import cv2

execution_path = os.getcwd()
camera = cv2.VideoCapture(0)

detector = VideoObjectDetection()
detector.setModelTypeAsTinyYOLOv3()
detector.setModelPath( os.path.join(execution_path , "yolo-tiny.h5"))
detector.loadModel()


video_path = detector.detectObjectsFromVideo(
                camera_input=camera,
                output_file_path=os.path.join(execution_path, "camera_detected_video_prueb
                frames_per_second=20, log_progress=True, minimum_percentage_probability=40
```

## RetinaNet

```
from imageai.Detection import VideoObjectDetection
import os
import cv2

execution_path = os.getcwd()
camera = cv2.VideoCapture(0)

detector = VideoObjectDetection()
detector.setModelTypeAsRetinaNet()
detector.setModelPath(os.path.join(execution_path , "resnet50_coco_best_v2.1.0.h5"))
detector.loadModel()
```

```
video_path = detector.detectObjectsFromVideo(camera_input=camera,
                                             output_file_path=os.path.join(execution_path,
                                             frames_per_second=20,
                                             log_progress=True,
                                             minimum_percentage_probability=40,
                                             detection_timeout=120)
```

```
WARNING:tensorflow:No training configuration found in the save file, so the model was
```

En este nuevo código definimos una instancia de OpenCV VideoCapture y cargamos la cámara del dispositivo predeterminada. Analizamos la cámara que definimos en el parámetro camera_input que reemplaza el input_file_path que se usa para el archivo de video.

ImageAI proporciona análisis de video de calidad comercial para detección de objetos de video para entradas de archivos de video y entradas de cámara. Permite obtener información detallada. Se puede visualizar en tiempo real y almacenarse en una base de datos NoSQL para su revisión o análisis en el futuro.

detectObjectsFromVideo () y detectCustomObjectsFromVideo () se ejecutan para cada fotograma, segundo y / o minuto del video detectado. Las funciones reciben datos analíticos sin procesar completos sobre el índice del cuadro / segundo / minuto, los objetos detectados (nombre, porcentaje_probabilidad y puntos_cuadro), número de instancias de cada objeto único detectado y número promedio de ocurrencia de cada uno.

Para obtener el análisis de video: Especificar una función, indicar los parámetros y analizar el nombre de la función en los parámetros per_frame_function, per_second_function, per_minute_function y video_complete_function en la función de detección.

```
def forFrame(frame_number, output_array, output_count):
    print("FOR FRAME " , frame_number)
    print("Output for each object : ", output_array)
    print("Output count for unique objects : ", output_count)
    print("------------END OF A FRAME --------------")

def forSeconds(second_number, output_arrays, count_arrays, average_output_count):
    print("SECOND : ", second_number)
    print("Array for the outputs of each frame ", output_arrays)
    print("Array for output count for unique objects in each frame : ", count_arrays)
    print("Output average count for unique objects in the last second: ", average_output_c
    print("------------END OF A SECOND --------------")

def forMinute(minute_number, output_arrays, count_arrays, average_output_count):
    print("MINUTE : ", minute_number)
    print("Array for the outputs of each frame ", output_arrays)
    print("Array for output count for unique objects in each frame : ", count_arrays)
    print("Output average count for unique objects in the last minute: ", average_output_c
    print("------------END OF A MINUTE --------------")
```

# Experimentos

```
video_detector = VideoObjectDetection()
video_detector.setModelTypeAsYOLOv3()
video_detector.setModelPath(os.path.join(execution_path, "yolo.h5"))
video_detector.loadModel()

video_detector.detectObjectsFromVideo(
    input_file_path=os.path.join(execution_path, "data-videos/Goliniestacorto.mp4"),
    output_file_path=os.path.join(execution_path, "Gol_iniesta_yolo"),
    frames_per_second=10,
    per_second_function=forSeconds,
    per_frame_function=forFrame,
    per_minute_function=forMinute,
    minimum_percentage_probability=30
)
```

Modelo: YOLO. FPS: 10. VELOCIDAD: normal

```
video_detector = VideoObjectDetection()
video_detector.setModelTypeAsYOLOv3()
video_detector.setModelPath(os.path.join(execution_path, "yolo.h5"))
video_detector.loadModel(detection_speed="flash")

video_detector.detectObjectsFromVideo(
    input_file_path=os.path.join(execution_path, "data-videos/Goliniestacorto.mp4"),
    output_file_path=os.path.join(execution_path, "Gol_iniesta_yolo2"),
    frames_per_second=20,
    per_second_function=forSeconds,
    per_frame_function=forFrame,
    per_minute_function=forMinute,
    minimum_percentage_probability=30
)
```

Modelo: YOLO. FPS: 20. VELOCIDAD: flash

```
video_detector = VideoObjectDetection()
video_detector.setModelTypeAsTinyYOLOv3()
video_detector.setModelPath( os.path.join(execution_path , "yolo-tiny.h5"))
video_detector.loadModel()

video_detector.detectObjectsFromVideo(
    input_file_path=os.path.join(execution_path, "data-videos/Goliniestacorto.mp4"),
    output_file_path=os.path.join(execution_path, "Gol_iniesta_yolo2"),
    frames_per_second=10,
    per_second_function=forSeconds,
    per_frame_function=forFrame,
```

```
    per_minute_function=forMinute,
    minimum_percentage_probability=30
)
```

Modelo: YOLOTINY. FPS: 10. VELOCIDAD: normal

```
video_detector = VideoObjectDetection()
video_detector.setModelTypeAsTinyYOLOv3()
video_detector.setModelPath( os.path.join(execution_path , "yolo-tiny.h5"))
video_detector.loadModel(detection_speed="flash")

video_detector.detectObjectsFromVideo(
    input_file_path=os.path.join(execution_path, "data-videos/Goliniestacorto.mp4"),
    output_file_path=os.path.join(execution_path, "Gol_iniesta_yolo2"),
    frames_per_second=20,
    per_second_function=forSeconds,
    per_frame_function=forFrame,
    per_minute_function=forMinute,
    minimum_percentage_probability=30
)
```

Modelo: YOLOTINY. FPS: 20. VELOCIDAD: flash

```
video_detector = VideoObjectDetection()
video_detector.setModelTypeAsRetinaNet()
video_detector.setModelPath(os.path.join(execution_path , "resnet50_coco_best_v2.1.0.h5"))
video_detector.loadModel()

video_detector.detectObjectsFromVideo(
    input_file_path=os.path.join(execution_path, "data-videos/Goliniestacorto.mp4"),
    output_file_path=os.path.join(execution_path, "Gol_iniesta_yolo2"),
    frames_per_second=10,
    per_second_function=forSeconds,
    per_frame_function=forFrame,
    per_minute_function=forMinute,
    minimum_percentage_probability=30
)
```

Modelo: RESNET. FPS: 10. VELOCIDAD: normal

```
video_detector = VideoObjectDetection()
video_detector.setModelTypeAsRetinaNet()
video_detector.setModelPath(os.path.join(execution_path , "resnet50_coco_best_v2.1.0.h5"))
video_detector.loadModel(detection_speed="flash")

video_detector.detectObjectsFromVideo(
    input_file_path=os.path.join(execution_path, "data-videos/Goliniestacorto.mp4"),
    output_file_path=os.path.join(execution_path, "Gol_iniesta_yolo2"),
    frames_per_second=20,
    per_second_function=forSeconds,
```

```
    per_frame_function=forFrame,
    per_minute_function=forMinute,
    minimum_percentage_probability=30
)
```

Modelo: RESNET. FPS: 20. VELOCIDAD: flash

```
# def forFull(output_arrays, count_arrays, average_output_count):

# video_detector.detectObjectsFromVideo(
#    input_file_path=os.path.join(execution_path, "data-videos/Goliniestacorto.mp4"),
#    output_file_path=os.path.join(execution_path, "gol_iniesta_forFull"),
#    frames_per_second=10,
#    video_complete_function=forFull,
#    minimum_percentage_probability=30
#)
```

En el resultado anterior, el video se procesó y se guardó en 10 cuadros por segundo (FPS).

Segundo índice: Es el número de posición del segundo dentro del video (por ejemplo, 1 para el primer segundo y 20 para el vigésimo segundo).

Matriz de salida: Es una matriz de matrices, con cada matriz contenida y su posición (índice de matriz + 1) correspondiente al fotograma equivalente en el último segundo del video. Cada matriz contenida contiene diccionarios. Cada diccionario corresponde a cada objeto detectado en la imagen y contiene los valores de "nombre", "porcentaje_probabilidad" y "puntos_cuadro" (x1, y1, x2, y2) del objeto.

Contar matrices: esta es una matriz de diccionarios. Cada diccionario y su posición (índice de matriz + 1) corresponde al fotograma equivalente en el último segundo del video. Cada diccionario tiene el nombre de cada objeto único detectado como sus claves y el número de instancias de los objetos detectados como valores.

Recuento de salida promedio: este es un diccionario que tiene el nombre de cada objeto único detectado en el último segundo como sus claves y el número promedio de instancias de los objetos detectados en el número de fotogramas como valores.

Los 4 parámetros que se devuelven por cada segundo del video procesado son los mismos parámetros que se devolverán por cada minuto. La diferencia es que el índice devuelto corresponde al índice de minutos, output_arrays es una matriz que contiene el número de FPS * 60 número de matrices, y count_arrays es una matriz que contiene el número de FPS * 60 número de diccionarios y el average_output_count es un diccionario que cubre todos los objetos detectados en todos los fotogramas contenidos en el último minuto.

Los resultados dnos permiten obtener un análisis completo de todo el video procesado. Necesitamos definir una función como la función forSecond o forMinute y establecer el parámetro video_complete_function en su función .detectObjectsFromVideo () o .detectCustomObjectsFromVideo (). Se devolverán los mismos valores para per_second-

function y per_minute_function. La diferencia es que no se devolverá ningún índice y se devolverán los otros 3 valores, y los 3 valores cubrirán todos los fotogramas del vídeo.

```
def forFrame(frame_number, output_array, output_count, detected_frame):
    print("FOR FRAME " , frame_number)
    print("Output for each object : ", output_array)
    print("Output count for unique objects : ", output_count)
    print("Returned Object is : ", type(detected_frame))
    print("------------END OF A FRAME --------------")

video_detector.detectObjectsFromVideo(
    input_file_path=os.path.join(execution_path, "data-videos/8K_Football_corto.mp4"),
    output_file_path=os.path.join(execution_path, "8K_Football_prueba.mp4"),
    frames_per_second=10,
    per_frame_function=forFrame,
    minimum_percentage_probability=30,
    return_detected_frame=True
)
```

# conclusiones

```
# USAR YoloV3

from imageai.Detection import ObjectDetection
import os

execution_path = os.getcwd()

detector = ObjectDetection()
detector.setModelTypeAsYOLOv3()
detector.setModelPath( os.path.join(execution_path , "yolo.h5"))
detector.loadModel(detection_speed="fastest")
detections = detector.detectObjectsFromImage(input_image=os.path.join(execution_path , "da

for eachObject in detections:
    print(eachObject["name"] , " : ", eachObject["percentage_probability"], " : ", eachObj
    print("--------------------------------")
```

```
 person  :  30.33740222454071  :  [279, 26, 301, 52]
 --------------------------------
 person  :  91.17082357406616  :  [440, 102, 486, 166]
 --------------------------------
 person  :  76.85571312904358  :  [44, 110, 90, 180]
 --------------------------------
 person  :  79.49832677841187  :  [61, 109, 110, 183]
 --------------------------------
 person  :  82.55444765090942  :  [191, 118, 228, 177]
 --------------------------------
 skis  :  32.61066675186157  :  [191, 118, 228, 177]
 --------------------------------
 skis  :  38.12129199504852  :  [434, 154, 492, 170]
```

```
--------------------------------
skis  :  32.38912224769592  :  [45, 178, 87, 187]
--------------------------------
skis  :  67.96051859855652  :  [65, 179, 116, 191]
--------------------------------
person  :  94.94777321815491  :  [205, 145, 279, 243]
--------------------------------
person  :  92.56008863449097  :  [282, 149, 347, 249]
--------------------------------
person  :  80.04907369613647  :  [339, 169, 376, 238]
--------------------------------
skis  :  48.59285056591034  :  [215, 229, 274, 247]
--------------------------------
skis  :  47.45813608169556  :  [291, 230, 346, 251]
--------------------------------
person  :  97.74788618087769  :  [428, 193, 505, 306]
--------------------------------
person  :  88.31343650817871  :  [516, 192, 589, 287]
--------------------------------
person  :  98.11171889305115  :  [114, 203, 195, 349]
--------------------------------
skateboard  :  48.893240094184875  :  [517, 272, 585, 291]
--------------------------------
skateboard  :  30.968406796455383  :  [103, 314, 178, 338]
--------------------------------
```

```
from imageai.Detection import ObjectDetection
import os


execution_path = os.getcwd()

# USAR RetinaNet
detector = ObjectDetection()
detector.setModelTypeAsRetinaNet()
detector.setModelPath(os.path.join(execution_path, "resnet50_coco_best_v2.1.0.h5"))
detector.loadModel()

detections = detector.detectObjectsFromImage(input_image=os.path.join(execution_path , "da

for eachObject in detections:
    print(eachObject["name"] , " : ", eachObject["percentage_probability"], " : ", eachObj
    print("--------------------------------")
```

```
WARNING:tensorflow:No training configuration found in the save file, so the model was
person  :  98.29033613204956  :  [280, 208, 369, 355]
--------------------------------
person  :  95.95049619674683  :  [281, 42, 349, 155]
--------------------------------
person  :  95.63030004501343  :  [559, 49, 612, 181]
--------------------------------
tennis racket  :  95.58138251304626  :  [332, 112, 375, 157]
--------------------------------
person  :  94.04250383377075  :  [48, 165, 118, 304]
--------------------------------
tennis racket  :  76.65385603904724  :  [540, 108, 577, 128]
--------------------------------
tennis racket  :  75.7364273071289  :  [92, 196, 123, 221]
```

```
--------------------------------
tennis racket  :  51.13970637321472  :  [576, 121, 601, 143]
--------------------------------
```

```
from imageai.Detection import ObjectDetection
import os


execution_path = os.getcwd()

# USAR RetinaNet
detector = ObjectDetection()
detector.setModelTypeAsRetinaNet()
detector.setModelPath(os.path.join(execution_path, "resnet50_coco_best_v2.1.0.h5"))
detector.loadModel()

detections = detector.detectObjectsFromImage(input_image=os.path.join(execution_path , "da

for eachObject in detections:
    print(eachObject["name"] , " : ", eachObject["percentage_probability"], " : ", eachObj
    print("--------------------------------")
```

```
WARNING:tensorflow:No training configuration found in the save file, so the model was
person  :  81.78449273109436  :  [297, 186, 414, 374]
--------------------------------
person  :  79.53939437866211  :  [416, 146, 562, 374]
--------------------------------
person  :  79.42053079605103  :  [208, 154, 278, 366]
--------------------------------
person  :  75.62363743782043  :  [647, 194, 715, 345]
--------------------------------
person  :  68.49392056465149  :  [697, 195, 748, 329]
--------------------------------
person  :  61.84029579162598  :  [215, 84, 315, 365]
--------------------------------
person  :  59.89253520965576  :  [53, 241, 102, 333]
--------------------------------
person  :  59.02734994888306  :  [412, 189, 490, 359]
--------------------------------
person  :  54.23574447631836  :  [265, 66, 336, 272]
--------------------------------
person  :  52.64163613319397  :  [247, 82, 406, 371]
--------------------------------
person  :  46.17352485656738  :  [0, 255, 40, 374]
--------------------------------
person  :  44.7573721408844  :  [0, 184, 53, 374]
--------------------------------
sports ball  :  41.8388694524765  :  [292, 64, 315, 92]
--------------------------------
person  :  41.260191798210144  :  [390, 196, 420, 266]
--------------------------------
person  :  39.45923149585724  :  [371, 224, 397, 268]
--------------------------------
person  :  36.35486364364624  :  [420, 193, 456, 268]
--------------------------------
person  :  32.444074749946594  :  [508, 51, 530, 98]
--------------------------------
```

```
tv  :  32.32133686542511  :  [80, 1, 221, 118]
--------------------------------
person  :  31.692799925804138  :  [114, 202, 143, 246]
--------------------------------
person  :  31.31427764892578  :  [676, 165, 700, 193]
--------------------------------
chair  :  30.299672484397888  :  [50, 243, 100, 334]
--------------------------------
```

2 min 8 s completado a las 12:44