

Universidad de Alcalá  
Escuela Politécnica Superior

GRADO EN INGENIERÍA INFORMÁTICA



ESCUELA POLITECNICA  
**Autor:** Mateusz Klimas  
**Tutor/es:** José Amelio Medina Merodio

2019



Universidad  
de Alcalá

**Escuela Politécnica Superior**

**Grado en Ingeniería Informática**

Trabajo Fin de Grado

**REALIDAD AUMENTADA APLICADA A LA  
BIOLOGÍA MARINA**

**Mateusz Klimas**

Alcalá de Henares, Junio 2019



**UNIVERSIDAD DE ALCALÁ**

**Escuela Politécnica Superior**

**Grado en Ingeniería Informática**

**Trabajo Fin de Grado**

**REALIDAD AUMENTADA APLICADA A LA  
BIOLOGÍA MARINA**

**Autor:** Mateusz Klimas

**Director:** José Amelio Medina Merodio

**TRIBUNAL:**

**Presidente:**

**Vocal 1º:**

**Vocal 2º:**

**CALIFICACIÓN:**

\_\_\_\_\_

**FECHA:**



Dedicado a:

Mis padres, que siempre han sido un apoyo fundamental a la hora de afrontar cualquier dificultad que pudiese aparecer a lo largo del camino.

A todas las personas del mundo del submarinismo, que a través de su pasión ayudan a dar a conocer el mundo de la biología marina.

A toda persona que aporta su grano de arena para seguir manteniendo limpios los mares y océanos, ayudando a conservar la vida marina.



Agradecimiento especial a la Universidad de Alcalá de Henares, por darme la gran oportunidad de estudiar el Grado de Ingeniería Informática, culminando esta etapa con el presente Trabajo de Fin de Grado.

También quiero agradecer todo el apoyo brindado por mi familia a lo largo del transcurso de mi época estudiantil, donde siempre he recibido la motivación y fuerza necesarias para afrontar cualquier tipo de situación.

Por último, me gustaría dar las gracias a José Amelio Medina Merodio, que se ofreció a ser mi tutor para este proyecto y me abrió las puertas a un nuevo campo del mundo de las ciencias de la computación: el Machine Learning.



## INDICE

RESUMEN .....	1
ABSTRACT .....	3
RESUMEN EXTENDIDO.....	5
INTRODUCCIÓN .....	13
JUSTIFICACIÓN Y MOTIVACIÓN.....	13
OBJETIVOS DEL PROYECTO.....	14
ESTRUCTURA DE LA MEMORIA.....	14
1.1. REALIDAD AUMENTADA.....	19
1.1.1. INTRODUCCIÓN .....	19
1.1.2. DIFERENCIAS ENTRE REALIDAD AUMENTADA Y REALIDAD VIRTUAL	19
1.1.3. TIPOS DE REALIDAD AUMENTADA .....	21
1.1.4. DISPOSITIVOS PARA LA REALIDAD AUMENTADA.....	24
1.1.5. CAMPOS DE APLICACIÓN.....	24
1.1.6. APLICACIONES REALES .....	25
1.1.7. POSIBLES PELIGROS .....	26
1.1.8. USO EN APLICACIÓN PROPUESTA .....	27
1.2. MACHINE LEARNING: CLASIFICACIÓN DE IMÁGENES .....	29
1.2.1. INTRODUCCIÓN .....	29
1.2.1.1. APRENDIZAJE SUPERVISADO.....	29
1.2.1.2. APRENDIZAJE NO SUPERVISADO.....	30
1.2.1.3. APRENDIZAJE POR REFORZAMIENTO.....	30
1.2.2. APLICACIONES DEL MACHINE LEARNING .....	30
1.2.3. REDES NEURONALES ARTIFICIALES (ANN) .....	31
1.2.4. REDES NEURONALES CONVOLUCIONALES (CNN) .....	34
1.2.5. COMPONENTES DE UNA CNN .....	35
1.2.6. MODELOS MÁS FAMOSOS .....	37
1.3. DETECCIÓN DE OBJETOS .....	41
1.3.1. INTRODUCCIÓN A LOS ALGORITMOS DE DETECCIÓN DE OBJETOS .....	41
1.3.2. ALGORITMOS DE DETECCIÓN DE OBJETOS .....	41
1.3.2.1. R-CNN .....	41
1.3.2.2. FAST R-CNN.....	42
1.3.2.3. FASTER R-CNN.....	43
1.3.2.4. YOLO .....	43
1.3.3. ¿QUÉ ES DARKNET? .....	43
1.3.4. YOU ONLY LOOK ONCE (YOLO) .....	43
1.3.5. YOLO V2 .....	44
1.3.6. YOLO V3 .....	46
2.1. INTRODUCCIÓN .....	51
2.2. APLICACIONES EXISTENTES .....	53
2.3. REQUISITOS.....	55
2.3.1. REQUISITOS DE USUARIO .....	55
2.3.2. REQUISITOS DE SISTEMA .....	55
2.4. BOCETOS DEL SISTEMA.....	59
2.4.1. BOCETOS DE LOS BLOQUES DEL SISTEMA .....	59
2.4.2. BOCETOS DE LAS SALIDAS DEL SISTEMA .....	60
2.5. ARQUITECTURA DEL SISTEMA.....	61
2.5.1. DESCRIPCIÓN DE LA ARQUITECTURA .....	61
2.5.1.1. Suministrador de vídeo.....	62

2.5.1.2. Algoritmo de detección (YOLO v3).....	62
2.5.1.3. Red neuronal de clasificación (CNN).....	63
2.5.1.4. Procesamiento de la información para su presentación.....	66
2.5.2. METODOLOGÍA SOFTWARE .....	66
2.5.3. CASOS DE USO .....	73
2.5.4. DIAGRAMA DE CLASES .....	74
2.6. IMPLEMENTACIÓN Y FUNCIONAMIENTO DEL SISTEMA .....	77
2.6.1. SPRINT 1 .....	77
2.6.2. SPRINT 2 .....	78
2.6.3. SPRINT 3 .....	79
2.6.4. SPRINT 4 .....	82
2.6.5. SPRINT 5 .....	85
2.6.6. SPRINT 6 .....	86
CONCLUSIONES Y FUTURAS LÍNEAS .....	89
3.1. MANUAL DE INSTALACIÓN .....	93
3.1.1. INSTALACIÓN CUDA .....	93
3.1.2. INSTALACIÓN OPENCV EN SISTEMA OPERATIVO .....	93
3.1.3. INSTALACIÓN KERAS .....	95
3.2. MANUAL DE USUARIO .....	97
PRESUPUESTO DEL PROYECTO.....	101
BIBLIOGRAFIA .....	105

## RESUMEN

El globo terrestre se encuentra cubierto de agua, en su gran parte por océanos. Se estima que solo ha sido explorado un 5% de estos, lo cuál supone que existe un gran número de especies marinas sin descubrir todavía. Además, el conocimiento hoy en día sobre la biología marina es muy básico para la mayor parte de la población. Salvo el conocimiento de especies muy famosas como puede ser el pez payaso, el resto cae en el olvido, y estas generalmente se suelen denominar peces, en vez de ser referidas por su especie. Por estas razones, el objetivo de este trabajo ha sido desarrollar una aplicación que ayude al estudio de poblaciones marinas y su muestreo, así como su uso como herramienta didáctica. Para ello, se ha desarrollado una aplicación que conjuga la visión artificial y la realidad aumentada sobre vídeos. A través de esta aplicación se permite detectar cualquier pez que aparezca en el vídeo, y posteriormente clasificarlo proporcionando información relativa a este.

*Palabras clave:* biología, biología marina, visión artificial, realidad aumentada, machine learning



**ABSTRACT**

The Earth globe it is found covered in water, big part of it by oceans. It is estimated that only about 5% of them has been explored, which supposes that exists a huge number of underwater species still without being discovered. Moreover, nowadays the knowledge about the marine biology it is very basic in the great part of the population. Except the knowledge of the most famous species such as the Clown Fish, the rest end up being forgotten, and these in the general cases are usually referred as fishes, instead of referring to them by their species. For these reasons, the target of this work has been to develop and application that helps in the study of marine populations and its sampling, as well as its use as an educational tool. In this way, an application has been developed that combines the artificial vision and augmented reality over videos. Through this application the intention is to detect any fish that appears in a video, and later classify them to be able to provide further information about it.

*Keywords:* biology, marine biology, artificial vision, augmented reality, machine learning



## RESUMEN EXTENDIDO

### INTRODUCCIÓN

Vivimos en un mundo en el que la tecnología esta viviendo una gran revolución. Si miramos un poco hacia atrás, no más de 50 años, podemos ver el gran “boom” que ha tenido el sector tecnológico. Todo este avance obliga a las personas a estar al corriente a diario, o de lo contrario dejaras de competir en el mercado laboral, el cuál exige cada vez más competencias y conocimientos en este campo.

Dentro del sector de la tecnología existen gran multitud de campos que cada día van ramificándose en muchos más. Hoy en día se habla mucho sobre el tema de la inteligencia artificial. ¿Qué entendemos realmente por inteligencia artificial? Según la RAE [1], la definición para la inteligencia artificial es:

*“Disciplina científica que se ocupa de crear programas informáticos que ejecutan operaciones comparables a las que realiza la mente humana, como el aprendizaje o el razonamiento lógico.”*

Es decir, el fin de la IA es crear máquinas que sean capaces de pensar por si mismas y las cuales puedan ir aprendiendo a través de la experiencia. Se están haciendo muchos esfuerzos en este campo, pero todavía no se ha llegado a diseñar ninguna máquina que sea capaz de hacer esto en su totalidad (que tengan libre albedrío). Donde se han logrado avances en cambio es en el campo del Machine Learning, que puede considerarse como una derivación de la Inteligencia Artificial. Este campo es el encargado de estudiar algoritmos y modelos estadísticos para realizar ciertas tareas sin la necesidad de dar ninguna instrucción en concreto. Este avance viene ligado a la generación masiva de datos que se da hoy en día, ya que para que se puedan crear modelos de Machine Learning son necesarios datos de los cuales aprender.

El Machine Learning ofrece diversas soluciones en diferentes campos, como puede ser la detección de caras, reconocimiento de voz o clasificación de imágenes. Sobre todo, la parte de visión artificial se está empleando mucho para la creación de modelos para coches autónomos o detección de tumores en tomografías.

La intención de este trabajo es aplicar el campo de la visión artificial a la biología marina. ¿El porqué? La principal causa es el estudio de poblaciones marinas. La superficie de la tierra esta cubierta por los océanos aproximadamente en un 71%. Gran parte de estos océanos se encuentran sin explorar, y probablemente existan muchas especies que aun no se sepa de su existencia. Actualmente se sabe que existen alrededor de 33.000 especies de peces, de las cuales unas 20.000 corresponden a especies marinas [2]. Este es un gran número, y a veces puede resultar complicado identificar ciertas especies ya que no se está en contacto de la misma forma que con la vida terrestre.

De esta forma se pretende desarrollar una primera parte que se capaz de generalizar para la detección de cualquier pez, y posteriormente se desarrollará otra parte que será la encargada de clasificar las especies de un entorno cerrado como ejemplo. Como no existen datasets con fotografías hará falta crear uno para poder llevar a caso este desarrollo. Al final del todo veremos a los resultados que se ha llegado, que mejoras se pueden hacer, y como seguir con este proyecto para poder seguir mejorándolo.

Para terminar esta breve introducción, me gustaría hacer un llamamiento al cuidado del mar para que se pueda seguir conservando todas las especies de peces y otro tipo de seres vivos que existen actualmente. Estamos en una situación en la que se vierte una cantidad inimaginable de desechos al mar, lo cuál produce que poco a poco se vaya reduciendo la biodiversidad de los mares y océanos debido a la alta contaminación que esto produce. Si esta situación no se

solventa pronto, este trabajo podría carecer de sentido en un futuro, debida a la posible desaparición de grandes ecosistemas submarinos.

## ESTADO DEL ARTE

En la actualidad existen múltiples métodos para la detección y clasificación de imágenes. Hay que tener bien clara la diferencia entre estas dos tareas: en la primera se quiere localizar el/los objeto/s en una imagen, es decir, se quiere obtener las coordenadas en las que se encuentra dicho/s objetos/s, mientras que en la clasificación se quiere saber la probabilidad que tiene un objeto de pertenecer a una determinada clase.

Para la tarea de la detección existen numerosos modelos de detección que han ido surgiendo a lo largo del tiempo: desde R-CNN, Fast R-CNN y Faster R-CNN, a algoritmos capaces de realizar detecciones en tiempo real, como pueden ser todas las versiones de Yolo. Para este último se ha conseguido ir mejorándolo hasta ser capaz de clasificar y detectar en tiempo real objetos tanto grandes como pequeños que aparecen en una imagen. La tercera versión de Yolo esta entrenada para detectar 80 objetos usando el dataset de COCO [4], aunque la segunda versión es capaz de detectar hasta 9000 diferentes categorías de objetos.

**Tabla 1.**

Resultados de los diferentes modelos de detección con una IoU de 0.5.

Model	Train	Test	mAP	FLOPS	FPS
SSD300	COCO trainval	test-dev	41.2	-	46
SSD500	COCO trainval	test-dev	46.5	-	19
YOLOv2 608x608	COCO trainval	test-dev	48.1	62.94 Bn	40
Tiny YOLO	COCO trainval	test-dev	23.7	5.41 Bn	244
SSD321	COCO trainval	test-dev	45.4	-	16
DSSD321	COCO trainval	test-dev	46.1	-	12
R-FCN	COCO trainval	test-dev	51.9	-	12
SSD513	COCO trainval	test-dev	50.4	-	8
DSSD513	COCO trainval	test-dev	53.3	-	6
FPN FRCN	COCO trainval	test-dev	59.1	-	6
Retinanet-50-500	COCO trainval	test-dev	50.9	-	14
Retinanet-101-500	COCO trainval	test-dev	53.1	-	11
Retinanet-101-800	COCO trainval	test-dev	57.5	-	5
YOLOv3-320	COCO trainval	test-dev	51.5	38.97 Bn	45
YOLOv3-416	COCO trainval	test-dev	55.3	65.86 Bn	35
YOLOv3-608	COCO trainval	test-dev	57.9	140.69 Bn	20
YOLOv3-tiny	COCO trainval	test-dev	33.1	5.56 Bn	220
YOLOv3-spp	COCO trainval	test-dev	60.6	141.45 Bn	20

En cuanto a la clasificación de imágenes también existen diversos modelos que son capaces de realizar esta tarea, entre ellos AlexNet, VGG16 o GoogleNet. Estos modelos están entrenados sobre el dataset de la competición ILSVRC [3], el cuál contiene imágenes de 1000 clases diferentes. Como es obvio, para clases conocidas funcionan con unos resultados muy buenos, pero si se introducen clases desconocidas entonces nos dará clasificaciones que se parezcan en la medida de lo posible a la nueva clase pasada como entrada.

En el panorama actual no existe ningún tipo de aplicación o sistema dedicado a la detección y clasificaciones de peces. La única forma de poder obtener información sobre las diversas especies que uno pueda encontrarse debajo del agua es a través de guías submarinas o libros relacionados con la biología submarina. De ahí la motivación de desarrollar un aplicación que sea capaz de ayudar en esta tarea, pudiendo tener diversas aplicaciones futuras.

## METODOLOGÍA

A la hora de planificar el desarrollo de la aplicación, se plantearon varias metodologías. En primer lugar, se descarto usar metodologías tradicionales, como puede ser la de cascada o incremental. El motivo principal de esta decisión viene de la existencia de metodologías ágiles que permiten planificar proyectos con alto grado de incertidumbre.

Si empleamos una metodología tradicional, como puede ser la de cascada, el proyecto que se planifica se define desde el principio con planes detallados, costes, tiempos, etc. Cualquier cambio que se quisiese hacer durante el desarrollo del proyecto afectaría a toda esta planificación. Estos problemas se solventan con las metodologías ágiles.

Las metodologías ágiles están diseñadas para adaptarse a los cambios que puedan surgir a lo largo del desarrollo de un producto. La idea es dividir el desarrollo en la implementación de funciones pequeñas y básicas, a las cuales se les asocia un tiempo para su realización. La idea es ir detectando los errores para poder solventarlos cuanto antes. Este tipo de metodologías vienen muy bien para proyectos con alto nivel de incertidumbre y cambiantes. Para ello, la metodología escogida para el desarrollo de este proyecto ha sido Scrum.

Scrum es una metodología ágil desarrollada para trabajar con un equipo en intervalos de tiempo denominados Sprints. La idea de Scrum es mantener en constante contacto tanto al equipo de desarrollo como al cliente. Para ello, existen tres roles: Scrum Master, Scrum Product Owner y Scrum Team. Al ser el único componente de este proyecto, estos roles se obvian. Ahora queda por ver los eventos alrededor de los cuáles se desarrolla la actividad de Scrum:

- **Sprint planning:** evento que se realiza antes de iniciar un sprint. En este se realiza todo el plan para ver que tareas se van a realizar durante el sprint y quienes las van a realizar.
- **Daily Scrum Meeting:** en esta parte se revisa diariamente que se ha hecho, si ha surgido algún problema, si se ha conseguido solventarlo, y que se va a hacer para el siguiente día. Cualquier tipo de complicación que pueda surgir se comenta para intentar ser paliada.
- **Sprint Review:** este evento se realiza al finalizar un sprint. Entre las actividades que se realizan, las que más destacan son: la actualización de las tareas completadas, comentarios sobre como se ha desarrollado el sprint, o que complicaciones han podido surgir
- **Sprint Retrospective:** ocurre después del Sprint Review. El objetivo de este evento es hacer un análisis del sprint que se ha realizado, y ver que cosas pueden ser mejoradas a través de la experiencia ya vivida.

Con todo esto en mente, en primer lugar se ha confeccionado la lista de tareas que hay que realizar a lo largo del proyecto (Product Backlog), asignándoles diferentes prioridades, puntos de historia y estableciendo las dependencias de otras tareas. Teniendo la lista de tareas hecha, se ha descrito cada una de estas para que no haya ninguna duda a la hora de empezar a implementarlas.

Ya solo quedaba ponerse a trabajar. Para cada sprint se ha ido revisando el Product Backlog, del cuál se han ido escogiendo las tareas de tal forma que la suma de los puntos de historia no fuesen muy alta, o de lo contrario se aplicaría una carga de tarea excesiva, lo cuál podría conducir a no cumplir con los tiempos establecidos para cada sprint. Según la complejidad de la tarea, se ha asignado más o menos tiempo a cada sprint.

## RESULTADOS

Con todos los sprints concluidos y con el sistema listo para funcionar, vamos a ver cuales son los resultados que se han obtenido. En primer lugar, se ha desarrollado el detector general de peces

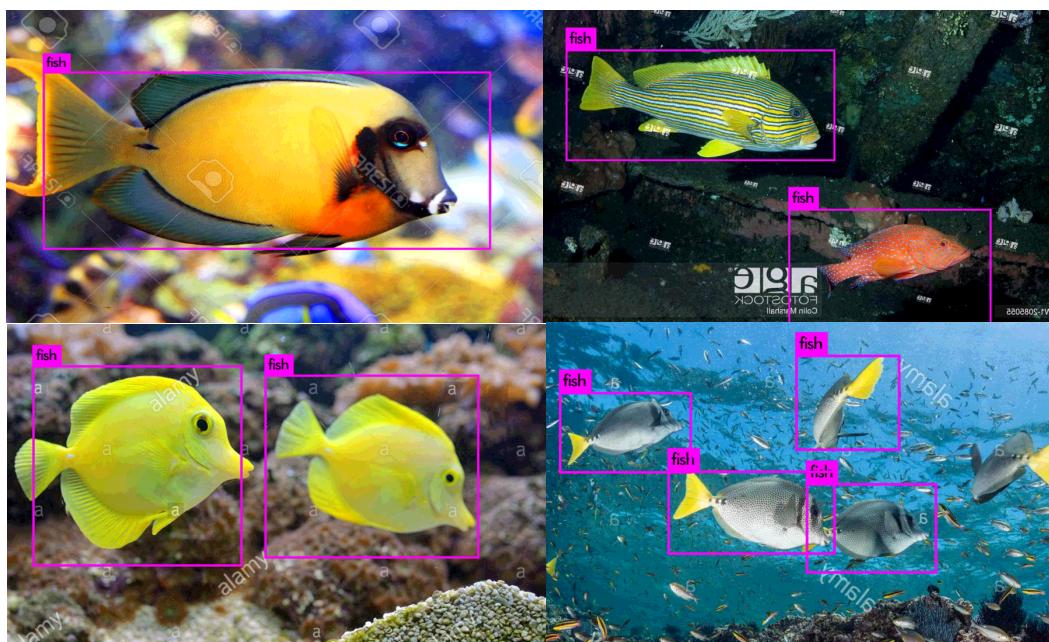
el detector general de peces. Para ello ha hecho falta recolectar mas de 8.000 fotografías, las cuales se etiqueto marcando cualquier pez que apareciese con ellas. Al querer detectar todo tipo de peces, se ha recolectado fotografías de todo tipo de especies de peces. En la Tabla 2 podemos ver la tabla que se ha obtenido con las medias de precisión media para diferentes Intersection over Union (IoU). Con IoU nos referimos a la medida que nos permite evaluar como de similar es nuestra predicción al objeto real de una imagen. A esta área en el que se localiza el objeto se le denomina ground truth.

**Tabla 2.**

Resultados del modelo de detección general.

IoU	mAP (media precisión media) (%)	IoU media (%)
0.3	93.96	78.33
0.4	93.19	78.27
0.5	91.35	78.06
0.6	86.18	77.17
0.7	66.44	70.16
0.8	25.92	44.64

En general los resultados son bastante buenos para los datos que se han obtenido, pero todavía hay margen de mejora. Esta mejora se encuentra en los datos necesarios para obtener mejores detecciones cuando los peces se localizan lejos o cuando hay una oclusión de estos. Para ello haría falta más imágenes en las que se den estos casos para poder reentrenar el modelo. También se podría obtener cierta mejora si se obtuviese en general más imágenes de diferentes especies, para que así el modelo pueda generalizar mejor.

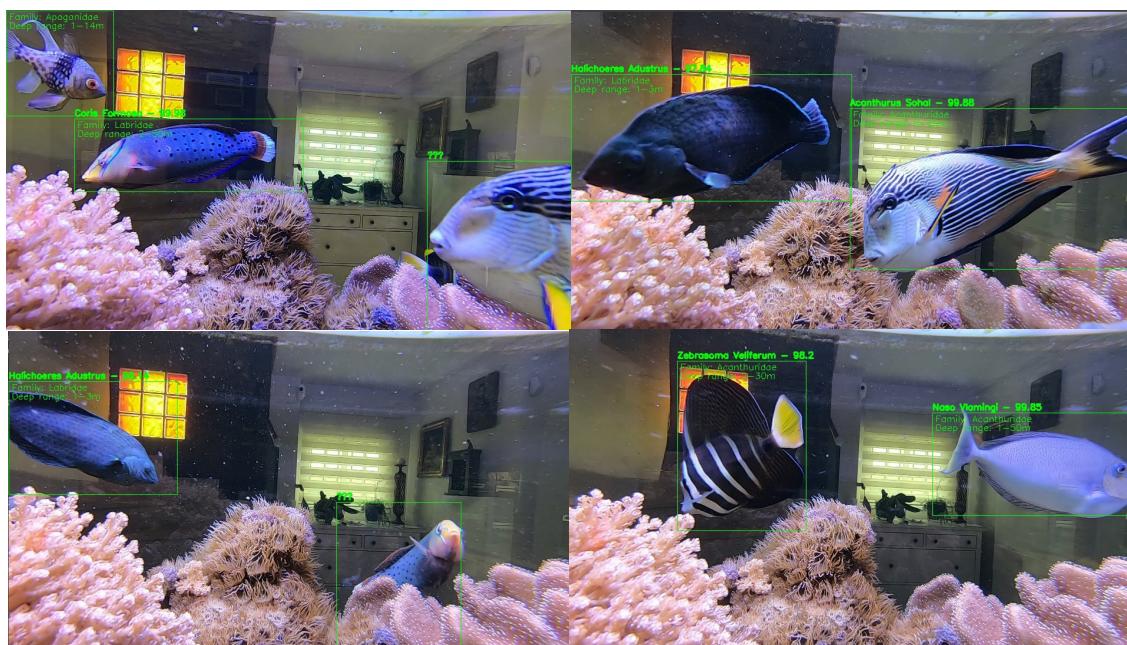


**Figura 1.** Imágenes con la salida del sistema de detecciones.

En cuanto al modelo de clasificación, se ha decidido usar un modelo conocido como base del clasificador, para así evitar la necesidad de recolectar un gran número de imágenes. Al emplear un modelo base ya entrenado sobre un gran número de datos, este ya tiene aprendidas muchas características que emplean para realizar las clasificaciones. De esta forma se ha cambiado la capa final del modelo original que se encarga de realizar la distribución probabilística para las clases, y se añaden un par de capas para cambiar el número de neuronas de salida, que estará

ligado al número de clases que se quieren clasificar. Con el rediseño del modelo, se reentrena con las imágenes recolectadas para las diferentes clases de peces, y se obtuvo una precisión del 98.51% y una pérdida del 0.068.

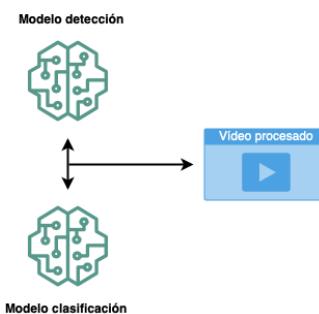
Finalmente, con los dos modelos listos, se puso estos conjuntamente en funcionamiento, para que mientras que uno va realizando las detecciones, otro se encarga de clasificar estas detecciones. Al emplear estos dos modelos conjuntamente, se reduce bastante el tiempo de proceso, y se analizan unos 4 fotogramas por segundo de media. A continuación, en la discusión se comentará que posibles mejoras se pueden realizar de cara al futuro para poder llegar a realizar esta detección y clasificación en tiempo real.



**Figura 2.** Ejemplos de la salida final de la aplicación.

## DISCUSIÓN

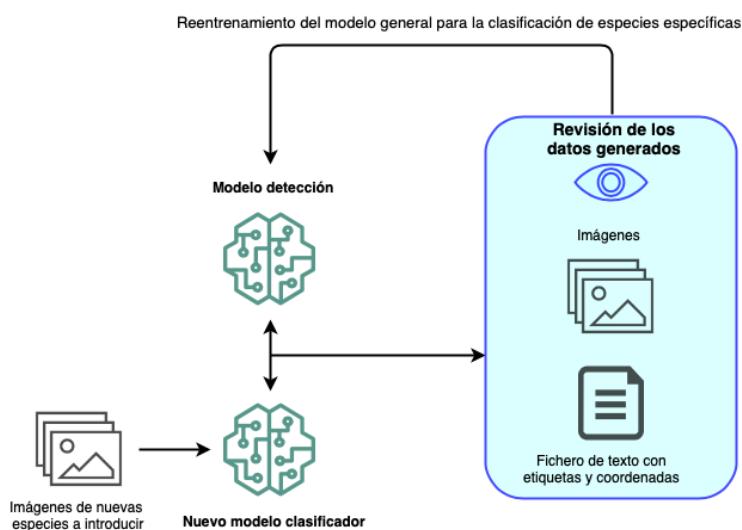
Tras el desarrollo del sistema, se ha visto que una desventaja que tiene esta aplicación es la velocidad de procesamiento del vídeo que se le pasa como entrada. Esto se debe a dos razones: se ha aumentado la resolución de entrada de los vídeos, y el modelo de clasificación conectado al modelo de detección produce retrasos, debido a que se emplean dos redes neuronales para la tarea.



**Figura 3.** Arquitectura simplificada del modelo actual.

La intención ha sido el diseño de un modelo que fuera capaz de generalizar las detecciones para cualquier especie de pez, y a si se ha hecho. Con el modelo general se ha creado el modelo clasificador como ejemplo sobre un entorno cerrado en el que se conoce el número de especies. En caso de que no importe el tiempo de procesado entonces esta aproximación está bien, en cambio si se desea obtener un procesamiento en tiempo real habría que cambiar un poco el escenario.

El planteamiento es seguir empleando el modelo de detecciones general, e ir reentrenando el modelo de clasificación para las especies nuevas que se quiera introducir. Para la introducción de especies nuevas bastaría con obtener unas 100-200 imágenes, las cuales posteriormente se ampliarían a 1000 mediante técnicas de data augmentation. Así obtendríamos un modelo que clasificaría nuevas especies pero que sigue funcionando con velocidades de procesamiento lentas. Para lograr una aplicación que funcione en tiempo real habría que modificar todo el sistema para que nos permita ir extrayendo los fotogramas con las detecciones y sus clasificaciones, es decir, se iría guardando estos fotogramas con un fichero de texto que indicase la clase a la que pertenece y las coordenadas en las que se encuentra cada objeto detectado. De esta forma, al necesitar muchas más imágenes para entrenar el modelo de detección general para que nos diga también las clases de las especies, estas se podrán obtener mediante grabaciones específicas a las especies que se quieran introducir al modelo. Estas grabaciones serán procesadas y se obtendrán las imágenes con las etiquetas. Estas etiquetas generadas por el modelo habrá que revisarlas manualmente para garantizar que son correctas y que no falte ninguna por poner. Así pues, se dispondrá de muchos más datos que garantizarán que el modelo de detecciones también pueda clasificar especies, en vez de generalizar para todas ellas. En la Figura 4 se puede ver como quedaría este proceso.



**Figura 4.** Resumen del proceso para la creación de modelos específicos de procesamiento en tiempo real.

## CONCLUSIONES Y FUTURAS LÍNEAS

Con el proyecto concluido, se ha cumplido con los objetivos que se ha propuesto menos con uno que se comentó en el anteproyecto, que era el de crear una herramienta para extraer detecciones y emplear la Vision API de Google para crear nuevos datasets. El motivo de esto es que se descubrió que realmente las clasificaciones que daba como respuesta esta API no era lo deseado tras varias pruebas.

A parte de este cambio, el resto de los objetivos que se propuso en el anteproyecto han sido cumplidos con éxito. Es obvio que todavía hay margen de mejora en los resultados obtenidos, pero estos dependen en gran parte de los datos de los que se disponga. Es por ello por lo que

una de las futuras líneas que se proponen es la ampliación del número de estos datos, introduciendo más fotografías de diversas especies en diferentes entornos, así mismo como en diferentes posiciones y con diferentes tamaños. Además, si se incorporan imágenes con cierto grado de oclusión también mejoraría mucho los resultados cuando los peces se encuentren en cierta medida obstaculizados por algún otro objeto.

Otra futura línea de investigación es la implantación de esta aplicación en unas gafas de buceo, para así crear un dispositivo de realidad aumentada que proporcione información relativa al entorno en el que se encuentre un submarinista buceando. Según la zona en la que se realice esta actividad, se elegirán diferentes modelos acordes a la localización, para así poder proporcionar información relativa a esa zona en concreto.



## INTRODUCCIÓN

Vivimos en un mundo en el que la tecnología está viviendo una gran revolución. Si miramos un poco hacia atrás, no más de 50 años, podemos ver el gran “boom” que ha tenido el sector tecnológico. Todo este avance obliga a las personas a estar al corriente a diario, o de lo contrario dejaras de competir en el mercado laboral, el cuál exige cada vez más competencias y conocimientos en este campo.

Dentro del sector de la tecnología existen gran multitud de campos que cada día van ramificándose en muchos más. Hoy en día se habla mucho sobre el tema de la inteligencia artificial. ¿Qué entendemos realmente por inteligencia artificial? Según la RAE [1], la definición para la inteligencia artificial es:

*“Disciplina científica que se ocupa de crear programas informáticos que ejecutan operaciones comparables a las que realiza la mente humana, como el aprendizaje o el razonamiento lógico.”*

Es decir, el fin de la IA es crear máquinas que sean capaces de pensar por si mismas y las cuales puedan ir aprendiendo a través de la experiencia. Se están haciendo muchos esfuerzos en este campo, pero todavía no se ha llegado a diseñar ninguna máquina que sea capaz de hacer esto en su totalidad (que tengan libre albedrío). Donde se han logrado avances en cambio es en el campo del Machine Learning, que puede considerarse como una derivación de la Inteligencia Artificial. Este campo es el encargado de estudiar algoritmos y modelos estadísticos para realizar ciertas tareas sin la necesidad de dar ninguna instrucción en concreto. Este avance viene ligado a la generación masiva de datos que se da hoy en día, ya que para que se puedan crear modelos de Machine Learning son necesarios datos de los cuales aprender.

El Machine Learning ofrece diversas soluciones en diferentes campos, como puede ser la detección de caras, reconocimiento de voz o clasificación de imágenes. Sobre todo, la parte de visión artificial se está empleando mucho para la creación de modelos para coches autónomos o detección de tumores en tomografías.

En esta sección introductoria vamos a ver de forma muy breve cuáles han sido las motivaciones para el desarrollo de la aplicación propuesta, los objetivos que se establecieron desde el principio y como se ha estructurado todo el desarrollo.

## JUSTIFICACIÓN Y MOTIVACIÓN

El planteamiento de este proyecto fue motivado por muchas razones. Yéndonos a un caso personal como el nuestro, nos dedicamos mucho al submarinismo. Cuando uno se encuentra buceando en el mar, se va topando con muchas especies a lo largo del recorrido durante el tiempo que dura una inmersión. Gran parte de estas especies pueden resultar desconocidas para muchas personas, ya que como es obvio, nosotros desarrollamos nuestra vida en la superficie, por lo que tenemos más contacto con las especies terrestres que las submarinas. De esta forma uno se refiere a gran parte de los seres vivos que habitan el mundo submarino como peces, pero estos peces realmente están divididos en un gran número de especies. Poniendo un ejemplo ridículo pero que es muy similar a lo que ocurre, es que llamásemos a todas las especies terrestres perros en el caso de que viviésemos en el mundo acuático, ya que al fin y al cabo gran parte de los animales terrestres andan a cuatro patas y tienen una forma anatómica similar a la de un perro. A través de este proyecto se quiere intentar dar a conocer más este mundo subacuático y ayudar a comprender la vida que se desarrolla en este entorno a través del uso de la realidad aumentada.

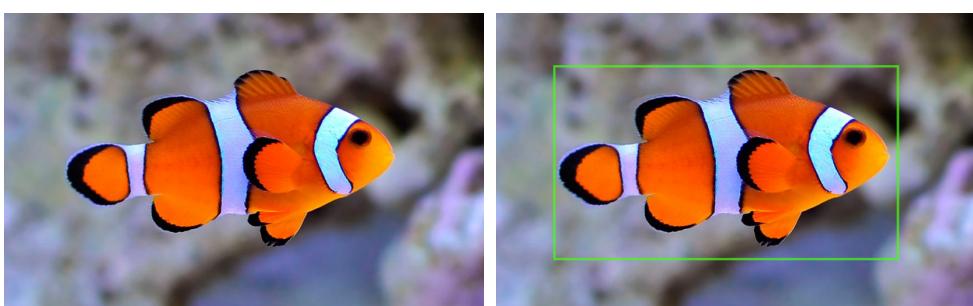
Por otro lado, vivimos en un mundo en el que se estima que hay alrededor de 8-9 millones de especies, de las cuales se conoce una parte muy pequeña. Al ritmo que se van clasificando

nuevas especies llevará un largo periodo de tiempo el poder catalogar todas. De esta manera se pretende diseñar una herramienta que pueda contribuir a la aceleración de este proceso.

En último lugar, los campos de relacionados con la visión artificial están en pleno desarrollo, donde cada día aparecen nuevas tecnologías y métodos que mejoran los resultados previamente obtenidos. Por eso se ha determinado que puede ser de gran interés aplicarlo a un entorno que no está muy en contacto con este tipo de tecnologías.

## OBJETIVOS DEL PROYECTO

El objetivo principal de este proyecto es el de diseñar e implementar un modelo que sea capaz de realizar detecciones de cualquier especie de pez sobre vídeos. Para ello, cualquier tipo de pez que aparezca en un determinado fotograma tiene que ser detectado, proporcionando las coordenadas de su localización y pintando una caja alrededor de este. En la Figura 5 se puede observar como tendría que quedar una imagen después de realizar una detección.



**Figura 5.** A la izquierda un fotograma sin procesar, a la derecha un fotograma procesado con la detección pintada.

Para alcanzar este objetivo principal se han definido los siguientes sub-objetivos que se presentan a continuación

Primero se ha realizado un análisis de las distintas herramientas existentes y del estado actual. Este estudio estará enfocado en analizar los diferentes algoritmos/modelos que existen, para ver cuáles se adaptan mejor a la aplicación propuesta.

En segundo lugar, se ha diseñará un clasificador que sea capaz de clasificar las especies. Este clasificador tiene que ser capaz de determinar qué especie le es pasada a través de una foto cualquiera, y dar información relativa a esta.

Por último, el último objetivo consiste en validar el sistema propuesto en un entorno real cerrado donde se encuentran una serie de especies definidas. Este entorno cerrado es un acuario, y se quiere determinar qué tipo de especies se van detectando, para poder ir mostrando información acorde a la especie que sea.

## ESTRUCTURA DE LA MEMORIA

La redacción de la memoria sigue un orden lógico, donde en primer lugar se ven de forma teórica los campos que se cubren en este proyecto, y luego se pasa a la parte práctica y las conclusiones. A continuación, se va a describir la estructura de la memoria para saber de forma resumida qué se cubre en cada parte.

### Marco teórico

En primer lugar, se tratan los temas que engloban este proyecto de forma teórica. Los temas tratados en este apartado son:

- **Realidad aumentada:** se ve en profundidad en que consiste la realidad aumentada, la diferencia entre realidad aumentada y realidad virtual, los diferentes tipos de realidad aumentada y campos de aplicación de esta con diversos ejemplos.
- **Machine Learning – Clasificación de imágenes:** se comienza hablando sobre información básica relacionada con este campo, pasando por las redes neuronales artificiales y terminando viendo las redes neuronales convolucionales, revisando varios modelos existentes.
- **Detección de objetos:** por último, se trata el tema de la detección de objetos, explicando diferentes algoritmos existentes y poniéndolos en comparativa para la elección de uno.

### Marco práctico

Con los conceptos teóricos cubiertos, se pasa a explicar todo el proceso que se ha seguido a la hora de desarrollar el proyecto. Los apartados que componen esta parte son:

- **Introducción:** se explica de forma breve la estructura del desarrollo.
- **Aplicaciones existentes:** análisis del panorama actual en cuanto a aplicaciones de carácter similar a la propuesta.
- **Requisitos:** todos los requisitos que deben ser cumplidos por el sistema son descritos en este apartado, tanto requisitos del usuario como del sistema.
- **Bocetos del sistema:** se representa de forma esquemática a través de imágenes el funcionamiento de la aplicación.
- **Arquitectura del sistema:** esta parte se divide en los siguientes sub-apartados:
  - **Descripción de la arquitectura:** todos los bloques que componen la aplicación son descritos y explicados.
  - **Metodología software:** explicación de la metodología empleada y la planificación.
  - **Casos de uso:** descripción del caso de uso de la aplicación.
  - **Diagrama de clases:** visualización del diagrama de clases de la aplicación donde se explican todos los métodos.
- **Implementación y funcionamiento del sistema:** en este apartado se explica todo el proceso que se ha seguido para desarrollar la aplicación. Esta explicación está dividida en los sprints que se han ido realizando a lo largo de la implementación del sistema.
- **Manual de usuario:** explicación de todo lo que es necesario tener instalado para poder poner en funcionamiento el sistema, además de explicar como usarlo.
- **Presupuesto:** desglose de los costes que conlleva desarrollar el proyecto.

### Conclusiones y futuras líneas

En esta última parte se resume todo lo logrado, viendo las complicaciones que han ido surgiendo y como se han solventado. Además de eso, se proponen futuras líneas de investigación para como mejorar esta aplicación y conseguir llevarla a distintas aplicaciones.



# DESARROLLO TEORICO



## 1.1. REALIDAD AUMENTADA

### 1.1.1. INTRODUCCIÓN

Hoy en día se está observando una gran evolución en la forma en la que se investiga y enseña gracias a la aparición de nuevas tecnologías. Una de estas tecnologías que está cogiendo mucho impulso es la realidad aumentada, ¿pero qué entendemos exactamente por realidad aumentada?

Según dice el libro “Virtual, Augmented and Mixed Reality”, la realidad virtual consiste en sobreponer información virtual en un espacio real [1]. ¿Pero cómo es posible llevar esta información inteligible al mundo real? Básicamente, a través de un medio que sea capaz de mostrar el mundo real como puede ser la pantalla de un Smartphone, se capturan imágenes/vídeos o cualquier tipo de señal del mundo real, y se les aplica las transformaciones/operaciones convenientes para mostrar el contenido oportuno. Resumiendo, la realidad aumentada nos permite mezclar objetos generados por computadores con la realidad.



**Figura 6.** Augment [6], aplicación que permite simular objetos virtuales en un entorno real.

El origen del término “realidad aumentada” viene de aumentar, que significa dar mayor extensión, número o materia a algo. De esta forma, se añaden capas de información virtual generada a través de dispositivos con suficiente capacidad de computo, ofreciendo al usuario una experiencia más enriquecedora del mundo físico.

Una cosa muy importante que no hay que confundir es la realidad aumentada con la realidad virtual. Son dos conceptos que pueden parecer iguales pero que no tienen nada que ver. Para aclarar esto, vamos a verlo en profundidad.

### 1.1.2. DIFERENCIAS ENTRE REALIDAD AUMENTADA Y REALIDAD VIRTUAL

Otro concepto que está emergiendo con mucha fuerza (y que no hay que confundir con la realidad aumentada) es la realidad virtual. La realidad virtual es el término empleado para describir mundos tridimensionales que no son reales. Estos mundos virtuales habilitan la posibilidad de ser explorados por los usuarios, además de permitir cierta interacción con ellos.

Para que un usuario pueda visualizar estos mundos virtuales, necesita ciertos dispositivos que le permitan representarlos de alguna forma. Algunos ejemplos de estos dispositivos son las gafas

de realidad virtual, unos auriculares de sonido, o unos mandos que le permitan moverse y interactuar con el entorno.



**Figura 7.** Usuario equipado con unas gafas de realidad virtual, cascos y arma equipada con sensores.

Gracias a la realidad virtual se pueden simular muchos entornos para diferentes aplicaciones, como puede ser la simulación de una operación policial, o la visualización de lugares a los que un usuario de forma normal no puede acceder, como puede ser una ciudad lejana.

De esta forma, lo que se hace a través de la realidad virtual es simular entornos que no son reales, y esta es la principal característica que lo hace completamente diferente de la realidad aumentada. En la realidad aumentada se interactúa con el mundo real, y en la realidad virtual con un mundo ficticio. Veamos unos ejemplos de realidad virtual y realidad aumentada relacionados con el mundo del videojuego para ver como estos dos términos difieren.

En primer lugar, tenemos un juego que tuvo una gran popularidad hace unos años, y que todavía sigue siendo muy conocido en el mundo. Este juego es Pokemon Go [8], diseñado por la compañía Niantic y lanzado en el año 2016. El principal objetivo de este juego es ir capturando Pokemons (una especie de criaturas fantásticas) alrededor de nuestro entorno. Cualquier tipo de Pokemon puede aparecer al lado nuestra en cada momento, y también puede ser que aparezca por las inmediaciones, por lo que el usuario tendrá que explorar sus alrededores para encontrarlos. Toda esta actividad se realiza en el mundo real, es decir, el usuario tiene que ir moviéndose por las calles, campo o cualquier sitio físico real para poder capturar estas criaturas. Como es obvio, en el mundo real no aparece ninguna de estas criaturas, pero el usuario puede visualizarlas a través de su dispositivo móvil. Cuando le aparezca un Pokemon, el usuario tiene la opción de activar su cámara frontal para poder visualizar dicho Pokemon en su entorno real. De esta forma, lo que se está haciendo es añadir Pokemons generados por la aplicación al mundo real. También hay puntos geográficos donde los usuarios pueden reunirse y pelear en entornos virtuales, pero que pueden visualizar gracias a sus Smartphones.



**Figura 8.** Ejemplo de como se vería un Pokemon simulado a través de un Smartphone.

Ahora, pasando a la realidad virtual tenemos Beat Saber [9], un videojuego de ritmo de realidad virtual desarrollado por la compañía Beat Games. El objetivo de este juego es ir cortando bloques que se acercan hacia el usuario al ritmo de una música. ¿Cómo logra el usuario ver estos bloques y cortarlos? Esto se logra gracias a unas gafas de realidad virtual, unos sensores de posición y unos controladores de realidad virtual. Estos mandos simulan los sables laser que usará el jugador para cortar los bloques que se dirijan hacia él. La visualización del entorno se logra gracias a las gafas de realidad virtual. Así pues, poniendo en conjunto todos estos accesorios, se logra simular un entorno tridimensional virtual con el cuál el usuario puede interactuar.

Con estos dos ejemplos, nos es posible entender la principal diferencia entre estos dos campos: en la realidad virtual toda la acción se desarrolla en el mundo real con ayuda de simulaciones que se adjuntan como capas de información sobre nuestro mundo físico, mientras que en la realidad virtual todo ocurre en entornos virtuales simulados, sin tener ninguna interacción con el mundo real.

### 1.1.3. TIPOS DE REALIDAD AUMENTADA

Con los conceptos aclarados de que es la realidad aumentada, vamos a pasar a ver como es posible llevar a cabo este proceso de adjuntar información a un entorno real. Es cierto que para poder interactuar con el mundo físico es necesario algún tipo de pre-procesado de las características del mundo real para luego poder aplicar la información deseada. En la Tabla 3 podemos observar los diferentes tipos de realidad aumentada que existen.

**Tabla 3.**

Tipos de realidad aumentada.

Tipo	Características	Ejemplos
Marcadores	<ul style="list-style-type: none"> <li>• Se utilizan marcas en ciertos lugares o superficies para posicionar los objetos.</li> <li>• El mundo virtual es anclado al mundo real</li> <li>• Las marcas son distintivos que permiten poner los objetos virtuales de forma correcta.</li> </ul>	Libros con marcas para mostrar diferentes escenarios
Sin marcador	<ul style="list-style-type: none"> <li>• Puesta de objetos en el entorno real.</li> <li>• No requieren marcadores.</li> <li>• El usuario decide la localización de los objetos.</li> </ul>	Aplicaciones para decoración de interiores
Basado en localización	<ul style="list-style-type: none"> <li>• Aplica información sobre una localización.</li> <li>• Puede ser una imagen o un mapa.</li> </ul>	Pokemon GO
“Dynamic Augmentation”	<ul style="list-style-type: none"> <li>• Interacción de la realidad aumentada con la detección de objetos.</li> </ul>	Aplicación propuesta

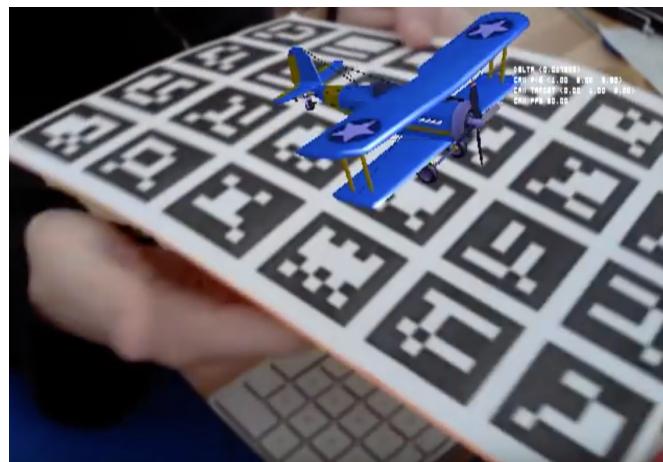
Vamos a pasar a ver un poco más en detalle estos tipos de realidad aumentada para entender mejor en qué se diferencias cada uno de ellos.

#### Realidad aumentada con marcadores

En este caso, para mostrar las capas de información deseadas se emplean unas marcas en planos como puntos de referencia. Estas marcas se pueden ser puestas sobre cualquier plano, ya sea un papel, una imagen o una superficie del suelo. Una vez puesta la marca, es posible visualizar la información virtual a través de dispositivos que tengan una cámara.

Para empezar, es necesario definir las marcas que harán que se dispare un evento, el cuál será el encargado de visualizar los objetos virtuales. Hoy en día se emplean diferentes tipos de marcadores, pero los más usado suelen ser los que tienen forma de rectángulo. Esto se debe principalmente a que a lo largo de los años se ha ido perfeccionando la tarea de reconocer rectángulos en las imágenes, a pesar de estar inclinados o con diferentes perspectivas. A través de la detección de rectángulos, es posible codificar información dentro de estos. Un ejemplo de esto es ArUco.

ArUco es una librería de realidad aumentada usada para la estimación de la localización de la posición una cámara. Lo que se hace es encontrar la correspondencia de los puntos del mundo real con las proyecciones que se desean hacer en el plano de dos dimensiones. Esto se logra gracias a los marcadores de ArUco. Estos marcadores son unos cuadrados con el borde negro y una matriz en su interior compuesta por cuadrados blancos y negros. Esta matriz determina el identificador de cada marcador. Gracias a que estos marcadores tienen el borde negro, es mucho más fácil la detección de estos cuadrados. Como es obvio, cuando se capturan estos marcadores a través de una cámara, estos lo más seguro es que estén rotados o inclinados. Es por ello muy importante que cuando se detecte un marcador ArUco sea posible determinar su posición original. Toda la colección de marcadores empleados en una aplicación se almacena en un diccionario.



**Figura 9.** Representación de una avioneta sobre un plano, empleando los marcadores ArUco para posicionarlo correctamente según la posición del plano.

Otro tipo de marcadores son los marcadores de imagen o marcadores NFT. Este tipo de marcadores no necesitan ningún borde para ser detectados. Se emplean una serie de características que definen una imagen y a través de las cuales se pueden detectar en una foto. Este tipo de marcadores son más complicados de desarrollar debido a que hay que saber exactamente qué características de la imagen se detectan mejor.

Una vez que se ha detectado una marca, se calcula la posición y orientación del marcador para así poder mostrar el contenido de forma apropiada.

#### Realidad aumentada sin marcadores

Otra forma de añadir información o contenido virtual a la realidad es sin utilizar ningún tipo de marcador. Para ello el usuario simplemente añade información a su entorno usando herramientas como puede ser el compás o giroscopio. Un ejemplo es la una aplicación para poner muebles virtuales en nuestro entorno. El usuario es capaz de poner cualquier objeto a su alrededor dejándolo fijo en esa posición. Hay que destacar que lo que se vaya añadiendo estará “flotando”, no está anclado a ningún punto como pasaba con los marcadores, aunque si es posible detectar superficies planas donde colocar lo que se desee.

#### Realidad aumentada basada en localización

En este tipo de realidad aumentada, la información que se añade al mundo real viene ligada a la posición donde se este. Para ello se emplea la localización geográfica, y según esta todos los elementos virtuales correspondientes se nos van mostrando alrededor nuestra. El ejemplo más fácil de entender es el de ir por una calle, sacar nuestro Smartphone mirar a través de él para que nos salgan los nombres de las calles, las rutas disponibles o cualquier información relacionada. No nos confundamos con lo que podría ser un mapa interactivo, el caso que se describe muestra toda esta información sobre la imagen que va captando nuestro dispositivo a través de su cámara. Hasta el momento, la aplicación más famosa que emplea este método es el juego para Smartphones, Pokemon Go, el cuál hemos mencionado anteriormente.

#### Dynamic Augmentation

Este es un tipo de realidad aumentada que se basa en la interacción con el mundo real empleando detección de objetos y su seguimiento. Este es el tipo de realidad que encaja con la aplicación que se va a desarrollar. Básicamente se detectan objetos de los cuales se puede ir mostrando información o ofrecer algún tipo de interacción.

#### 1.1.4. DISPOSITIVOS PARA LA REALIDAD AUMENTADA

Existe múltiples dispositivos que pueden ser usados para la realidad aumentada. El más común y frecuente son nuestros dispositivos móviles. Gracias a las cámaras de nuestros teléfonos inteligentes podemos interactuar con el mundo real añadiéndole cierto valor/información gracias a las aplicaciones que se van desarrollando diariamente por diferentes compañías.

Alejándonos de los Smartphones que son algo comunes, tenemos otros dispositivos diseñados específicamente para la tarea de la realidad aumentada, como pueden ser unas gafas de realidad aumentada. Estas gafas suelen ir equipadas con una cámara que capta la visión del mundo real, le aplican las operaciones oportunas, y la muestran sobre las gafas. Compañías como Google o Microsoft están trabajando en sus propias gafas de realidad aumentada.

Yéndonos a escalas más pequeñas, están las lentes de contacto usadas como dispositivo de realidad aumentada. Todavía no existe ningún producto definido, pero hay ciertas compañías como Samsung que están intentando desarrollar este tipo de dispositivos. De esta forma, las lentillas tendrían que ir equipadas con circuitos y accesorios que brinden ciertas funcionalidades.

Y si damos un paso más, llegamos a un campo que ya parece de ciencia ficción. Se trata de proyectar la información procesada directamente sobre la retina de una persona. Para ello, se capta el vídeo y se transforma a fotones con una intensidad determinada que se proyectan sobre la retina.

#### 1.1.5. CAMPOS DE APLICACIÓN

Con todo lo que hemos visto, se nos pueden venir muchas ideas a la cabeza sobre diversas aplicaciones que se le podría dar a la realidad virtual. Veamos algunos campos en los que puede ser de gran utilidad su utilización.

La educación es desde luego uno de los campos que pueden resultar más interesantes donde usar la realidad aumentada. Gracias a esta tecnología es posible recrear diferentes simulaciones de conceptos que se quieran enseñar a los alumnos con las cuales podrá interactuar. Además de ofrecer más información de lo que podría ofrecer un vídeo explicativo, resulta mucho más divertido y curiosos a la hora de aprender nuevos conceptos. Un ejemplo claro ejemplo de aplicación es a la hora de enseñar la anatomía humana. Usando métodos tradicionales solo se enseñaría diferentes fotos por capas de las distintas partes del cuerpo humano. Gracias a la realidad aumentada es posible reproducir diferentes partes del cuerpo sobre un plano con las que se puede interactuar.



**Figura 10.** Ejemplo de representación de un cerebro usando realidad aumentada con marcador.

Otro campo de aplicación es el de medicina. En primer lugar, las personas que estudian para llegar a ser cirujanos necesitan adquirir práctica en casos reales, ya que de otra forma nunca estarán listo para poder operar a alguien. A través de la realidad virtual se espera poder brindar

información y ayuda a estas personas en casos reales, para que puedan poner en práctica sus conocimientos en un entorno controlado. También existe la posibilidad de desarrollar tecnologías que sean capaces de detectar ciertos tejidos o formaciones malignas. Al final, la idea de la realidad aumentada aplicada en la medicina es brindar a los cirujanos de información útil mientras están operando, mejorando los resultados y reduciendo los fallos.

Se está invirtiendo mucho dinero en el ejército para desarrollar nuevas tecnologías que sean capaces de ayudar a los soldados durante las operaciones militares. Existen ciertas tareas que requieren el manejo de mucha información, lo cual puede ser abrumador, aparte de que puede resultar crítico en situaciones complicadas. Un ejemplo es el uso de interfaces que muestran información relativa a la altitud, velocidad o velocidad del viento mientras pilotan una aeronave. Si el piloto se encuentra en una situación comprometida, el consultar la instrumentación que tiene a bordo puede poner en peligro su integridad. En cambio, si dispone de un visor transparente donde se va mostrando esta información en tiempo real, el piloto evita estar mirando los instrumentos de abordo y puede centrarse en el manejo de la aeronave.

A parte de todos los campos mencionados, la realidad aumentada está teniendo un fuerte crecimiento en las aplicaciones de uso cotidiano. Desde aplicaciones que son usadas para mostrar figuras o animaciones en museos, hasta videojuegos en los que se interacciona con el mundo físico.

### 1.1.6. APLICACIONES REALES

#### Volvo

La empresa fabricante de vehículos Volvo y Microsoft están colaborando conjuntamente para revolucionar la industria del automóvil. Pretenden revolucionar este sector a través de la realidad aumentada, y de como se realizan todos los procesos. Mediante el uso de las Microsoft HoloLens se quiere cambiar la manera en la que diseñar y mostrar los diferentes vehículos que se vayan produciendo. La intención es proyectar las maquetas de estos vehículos en entornos reales, como puede ser la superficie de una mesa. Este proyecto puede cambiar la forma de trabajar en la industria del automóvil brindando infinitas posibilidades. Actividades como el diseño de nuevos coches o la exposición de modelos virtuales a clientes para mostrar diferentes características del vehículo son una de las muchas aplicaciones que se nos vienen a la mente.



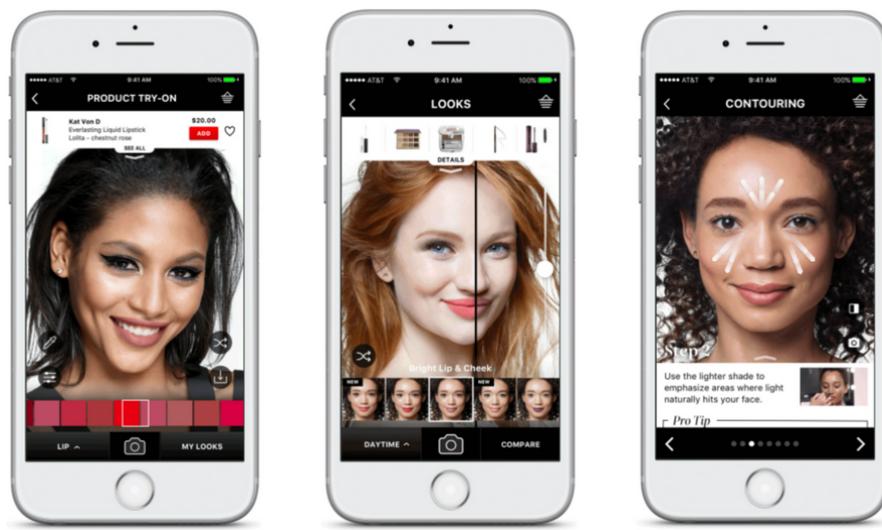
**Figura 11.** Recreación de como podría verse un coche a través de las HoloLens de Microsoft.

### Home Depot

Home Depot, la compañía americana de venta de suministros de construcción desarrolló una aplicación usando el kit de desarrollo de realidad aumentada de Apple. Esta aplicación permite colocar objetos de su catálogo, como pueden ser mesas, sillas o estanterías, en un entorno real, simulando como quedaría cualquiera de esos objetos sin la necesidad de traerlos físicamente. Con tan solo uno bajarse la aplicación puede ir probando como quedarían los diferentes muebles en su hogar.

### Sephora

Este caso es muy parecido al de Home Depot. En el mundo del maquillaje existe una infinidad de productos y colores/tonalidades que pueden adquirirse en el mercado hoy en día. Es por eso por lo que a veces muchos clientes duden a la hora de comprarse cierto producto debido a que no les quede bien y les sea imposible devolver el producto. Para ello, Sephora ha desarrollado una aplicación que permite probarse los diferentes productos empleando realidad aumentada.



**Figura 12.** Apariencia de la aplicación de Sephora para la prueba de distintos productos a través del uso de la realidad aumentada.

#### 1.1.7. POSIBLES PELIGROS

Son muchas las ventajas las que nos ofrece la realidad virtual, pero como todo, tiene sus lados negativos que hay que tener en cuenta. Hoy en día la protección de los datos y la privacidad es algo que hay que tener muy en cuenta. Los datos que se ofrecen a las grandes compañías al fin y al cabo es información que pueden usar para sus propios fines o bien venderla a terceros. Es por eso por lo que hay que tener especial cuidado con lo que se comparte en estas aplicaciones.

Pongamos un ejemplo con el juego de Pokemon GO. Para poder jugar a este juego es necesario tener la geolocalización activada siempre que este se use, incluso hay un modo en el que esta en segundo plano para realizar diversas tareas. De esta forma, la aplicación nos esta rastreando constantemente, pudiendo trazar rutas habituales del usuario en su día a día.

También es cierto que hoy en día la gente cada vez es más imprudente con la aparición de nuevas tecnologías, lo cuál hace que estas presten mayor atención a sus dispositivos que a su entorno. Este tipo de situaciones puede hacer que el usuario sufra despistes y pueda sufrir algún tipo de accidente mientras usa estas aplicaciones.

Por otro lado, hay otro tipo de peligro relacionado con la fiabilidad de la aplicación en cuanto a los resultados que esta proporciona. Si el campo donde se emplea cierta aplicación es intolerante a fallos, como una neurocirugía, cualquier tipo de información errónea que pueda ser proporcionada por los sistemas puede producir unos resultados catastróficos.

### 1.1.8. USO EN APLICACIÓN PROPUESTA

La aplicación que se ha propuesto para desarrollar esta comprendida dentro del campo de la realidad aumentada, siendo más concretos en Dynamic Augmentation. El principal objetivo es mostrar información relativa a las especies marinas que uno pueda encontrarse debajo del mar.

Para esta aplicación el tipo de capas de información que se va a aplicar al mundo físico es el marcado de los peces que vayan apareciendo por el vídeo que se tenga como entrada al sistema. Este marcado se hará de tal forma que enmarque completamente al pez detectado y que pueda ser apreciado fácilmente por el usuario. Además de destacar las detecciones, también se irá mostrando información relativa a cada detección, dependiendo totalmente de la especie que se encuentre.

La idea del desarrollo de esta aplicación es poder en el futuro trasladar esta aplicación a unas gafas de bucear, por ejemplo, en las que se vayan mostrando información relativa a la fauna marina que se está observando.

Otra posible aplicación es la de ayudar al estudio de las poblaciones marinas y muestreo en diversos estudios relacionados con la biología marina, ya que todo este campo está todavía en plena expansión.



## 1.2. MACHINE LEARNING: CLASIFICACIÓN DE IMÁGENES

### 1.2.1. INTRODUCCIÓN

Vivimos en un mundo en el que diariamente se produce una inmensa cantidad de datos, y cada día el tamaño de datos que se van generando va a más. ¿Pero para qué se quiere realmente estos datos? Los datos sin ningún tratamiento para darles algún significado carecen de valor. Ciencias como el Big Data o Machine Learning se encargan de darle este significado. Vamos a indagar en la ciencia del Machine Learning, que es el campo al que corresponde el trabajo desarrollado.

Básicamente, el Machine Learning se encarga de transformar la información (todos los datos) en conocimiento. Una descripción más detallada nos la da Daniel Fagella:

*“El Machine Learning es la ciencia que hace que los computadores aprendan y actúen como lo hacen las personas humanas, y mejoren su aprendizaje a lo largo del tiempo de forma autónoma, al proporcionales datos e información en forma de observaciones y interacciones con el mundo real.”*

Veamos algunas definiciones más que nos ayuden a entender lo que es realmente el Machine Learning.

1. “El Machine Learning, en su forma más básica, es la práctica de usar algoritmos para analizar datos, aprender de ellos, y entonces hacer una predicción determinada sobre algo del mundo.” – NVIDIA [15]
2. “Machine Learning es un método de análisis de datos que automatiza la construcción de modelos analíticos.” – SAS Institute [16]
3. “Machine Learning es una técnica de análisis de datos que enseña a los ordenadores a hacer lo que resulta natural para las personas y los animales: aprender de la experiencia.” – MathWorks [17]
4. “El Machine Learning usualmente se refiere a los cambios en un sistema que realiza tareas relacionadas con la inteligencia artificial. Dichas tareas” – Nils J. Nilsson [18]

Es decir, el objetivo del Machine Learning es conseguir diseñar máquinas capaces de aprender por sí solas, sin la necesidad de programarlas, y que sean capaces de predecir resultados a partir de la experiencia previa. Con experiencia previa nos referimos a datos históricos de una actividad en concreto. Dentro de la ciencia del Machine Learning existen tres ramas: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por reforzamiento.

#### 1.2.1.1. APRENDIZAJE SUPERVISADO

Para este tipo de Machine Learning se usan datos que han sido previamente etiquetados. Estos datos en forma de dataset se emplean para entrenar los sistemas. Para entrenar estos sistemas lo que se hace es pasarle una serie de características del dataset y la etiqueta correspondiente al resultado. De esta forma el algoritmo va aprendiendo a través de la experiencia proporcionada como datos. El objetivo final de estos sistemas es el poder pasarle unas características y que este sea capaz de predecir la etiqueta que le corresponda, sin tener la etiqueta final que le corresponde a esas características pasadas.

Este tipo de aprendizaje lo podemos comparar a como aprende un ser humano de pequeño. Imaginemos que un niño pequeño no sabe lo que es ni un elefante ni un caballo. A este se le proporcionan una serie de imágenes de caballos y elefantes (características de la etiqueta) y el nombre de cada uno (etiqueta de la correspondiente salida). A través de la visualización de estas

imágenes y sus nombres durante un periodo de tiempo, el niño es capaz de coger una nueva imagen y decir si es un caballo o un elefante sin tener el nombre de este. De la misma forma funciona un algoritmo supervisado, solo que los datos pasados como entrada al algoritmo y su salida tienen que ser correctamente transformados para que el modelo pueda interpretarlos correctamente.

Este modelo de aprendizaje es el que se va a emplear en la aplicación propuesta: a partir de unos datos sobre peces queremos obtener el tipo de pez que se pasa al sistema.

### 1.2.1.2. APRENDIZAJE NO SUPERVISADO

Al contrario de lo que ocurre en el aprendizaje supervisado, en el aprendizaje supervisado se tienen unos datos con unas características, pero estos no están etiquetados. El concepto de este tipo de aprendizaje es el poder agrupar los datos en categorías según las características que se elijan.

Llevando esta modalidad de aprendizaje a un ejemplo cotidiano es el de darle una serie de figuras a una persona con diferentes características (tamaño, forma, color, etc.) y decirle que las tiene que agruparlas utilizando una de estas características. Según los criterios establecidos para la agrupación, se obtendrán unas agrupaciones determinadas. Básicamente, lo que se hace es extraer características comunes de los datos, y asignar cada dato a un grupo según su valor.

### 1.2.1.3. APRENDIZAJE POR REFORZAMIENTO

Esta modalidad de aprendizaje está basado en la psicología conductista. Se tiene un agente localizado en un entorno, y el objetivo es lograr la toma de una serie de acciones que conduzcan a una recompensa. Es decir, se van tomando acciones y todas las que vayan produciendo resultados positivos en ese entorno se consideraran como recompensa, pero la idea es la obtención de una recompensa final o recompensa acumulada, la cuál llega a través de la toma de una serie de acciones correctas. De esta forma se logra que el agente vaya aprendiendo a tomar las acciones correctas que lleven a resolver un problema.

Imaginemos que a un niño se le deja en su habitación desordenada sin ningún tipo de material de entretenimiento. Al principio puede ser que el niño se entreteenga con algo (recompensa de una acción), pero a la larga puede que se acabe aburriendo. En un momento dado puede ser que se le ocurra ordenar la habitación, lo cuál hará que sus padres le den juguetes para que pueda entretenerte (recompensa acumulada/final). De esta forma, el niño asociará que si ordena su habitación entonces se le entregarán sus juguetes como recompensa. Todas las acciones de ordenar la habitación que va realizando son las acciones necesarias para llegar a esta recompensa. De la misma forma funciona el aprendizaje reforzado en sistemas en los que hay un problema y se tienen que seguir una serie de acciones hasta llegar a la solución.

## 1.2.2. APLICACIONES DEL MACHINE LEARNING

Con una idea más clara de lo que es el Machine Learning, vamos a ver algunas de las aplicaciones más comunes donde se emplea.

### Salud

Cuando vamos a un médico porque tenemos ciertos síntomas que pueden tener alguna relación con una enfermedad, el médico usa sus conocimientos al igual que su experiencia previa para darnos un diagnóstico. Es decir, si por ejemplo vamos al médico en época primaveral con dificultades respiratorias, lo más probable es que nos diagnostique con algún tipo de alergia.

Esta decisión lo más probable es que venga dada a partir de un historial en el que muchos pacientes que vienen en la misma época del año con síntomas parecidos.

Así, a través del uso del Machine Learning, si se usan datos de pacientes que han padecido cierta enfermedad, es posible detectar patrones que pueden indicar la posible aparición de una enfermedad en un paciente nuevo. Algunos casos en los que se emplea es para la detección de cáncer de mama, o la detección de tumores en las imágenes obtenidas de una tomografía.

#### Publicidad personalizada

Hoy en día, toda actividad que realicemos en Internet queda registrada. Si por ejemplo estamos explorando una tienda online por un producto en especial, lo más probable es que esa página este anotando toda esta actividad para poder ofrecer en el futuro recomendaciones similares al producto buscado. Compañías como Amazon emplean el Machine Learning. Estos van recaudando información de las búsquedas del usuario, y a través del historial del resto de usuarios es capaz de determinar que es lo próximo que le gustaría ver.

#### Seguridad informática

Gracias al Machine Learning es posible detectar ciertos tipos de ataques informáticos. Esto se debe a que muchos de estos ataques siguen una serie de patrones para poder llevar a cabo la intrusión. A través de estos patrones se entrena algoritmos capaces de detectar actividades sospechosas más rápido de lo que lo haría un sistema de seguridad convencional.

#### Detección de fraudes

Al igual que ocurre en la seguridad informática, cuando se intenta realizar fraudes financieros, se suele seguir una serie de pasos hasta completar estas actividades ilegales. De esta forma, compañías como PayPal implementan algoritmos capaces detectar acciones fraudulentas como puede ser el blanqueo de dinero. Esto lo consiguen a través de la comparación de millones de transacciones que se realizan a través de su plataforma.

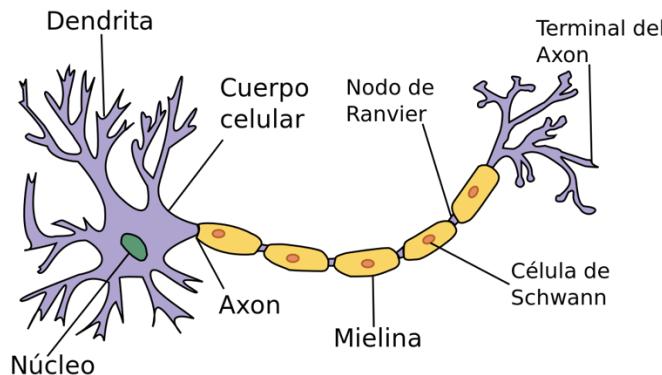
### 1.2.3. REDES NEURONALES ARTIFICIALES (ANN)

Hemos comenzado viendo lo que es el Machine Learning, y las diferentes modalidades que existen. Ahora vamos a indagar aun más en este campo, en concreto en la rama del Deep Learning. El Deep Learning consiste básicamente en aprender a través de ejemplos, es decir, estamos en un caso de aprendizaje supervisado. ¿Pero y como realmente se consigue crear estos algoritmos para predecir resultados?

En gran parte de los algoritmos de Deep Learning se emplean redes neuronales artificiales (ANN). Estas redes neuronales simulan el funcionamiento de un sistema biológico como es el cerebro, el cuál está compuesto por millones y millones de neuronas.

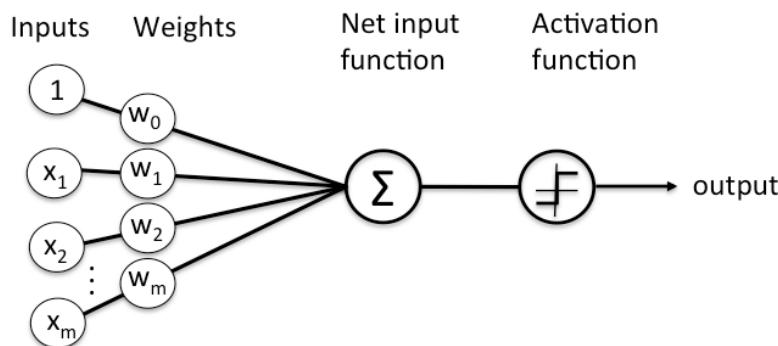
Cada neurona del cerebro está compuesta por las siguientes partes:

- **Dendritas:** son las extensiones de la neurona a través de las cuales recibe los impulsos eléctricos de otras neuronas y los transmite al cuerpo de la neurona.
- **Cuerpo celular o pericarion:** es la parte principal de la neurona. Aquí alberga se alojan todos los órganos de la neurona necesarios para su funcionamiento. Esta parte de la neurona permite el procesamiento de la información nerviosa que recibe de otras neuronas.
- **Axón:** es una prolongación del cuerpo de la neurona localizado en la parte opuesta a las dendritas. Se encarga de transmitir los impulsos eléctricos generados por la neurona al resto de neuronas.



**Figura 13.** Representación de una neurona biológica.

De la misma forma que funciona una neurona de un cerebro humano funcionan las neuronas de las redes neuronales artificiales. Cada neurona de una ANN tiene entradas hacia esta, que se podrían comparar con las dendritas de las neuronas biológicas. El cuerpo de la neurona de una ANN estaría compuesto por una función de activación, que produce una salida según las entradas que reciba. Por último, el axón de una neurona de una ANN sería la salida de la función de activación, que puede ser redirigida hacia otras neuronas.

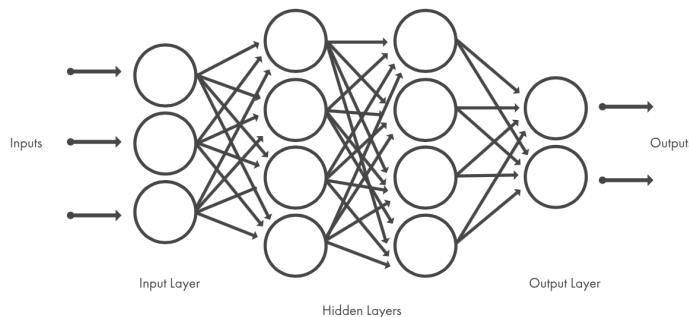


**Figura 14.** Representación de una neurona de una red neuronal artificial.

Al final, la arquitectura de las redes neuronales artificiales tiene la siguiente estructura:

- **Capa de entrada:** es la capa en la que se localizan las neuronas a las que se les pasa la entrada al modelo.
- **Capa oculta:** capa formada por neuronas que reciben entradas y proporcionan salidas según los pesos y las funciones de activación que se empleen.
- **Capa de salida:** capa que proporciona la salida del proceso de una entrada a través de la red neuronal.

Estas capas están interconectadas entre si mediante enlaces a través de los cuales van transmitiendo información. En la Figura 15 podemos observar como se vería la arquitectura de una red neuronal conformada por las distintas capas y como se entrelazan las neuronas entre ellas.



**Figura 15.** Arquitectura de una red neuronal.

Hemos visto las partes principales de una neurona, pero vamos a pasar a ver en más detalle qué elementos la componen para lograr el funcionamiento de un algoritmo. En primer lugar, cuando a una neurona se le pasa una entrada, a esta entrada se le aplican unos pesos que se establecen en la fase de entrenamiento. Con las entradas multiplicadas por los pesos se suman todos los resultados y se le añade un bias. El resultado es pasado a una función de activación que proporciona una salida según el tipo de función empleado. Algunas de las funciones de activación más empleadas son:

- **Función sigmoidea:** los valores introducidos son transformados a un rango  $\{0, 1\}$ . Tiene una convergencia muy lenta, es cóncava para valores mayores que 0, y convexa para valores menores que 0. Su ecuación es:

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

- **Rectified lineal unit (ReLU):** esta función es muy sencilla, los valores positivos activan la función, mientras que los negativos se anulan. Se emplea mucho para las redes neuronales convolucionales. Su ecuación es:

$$f(x) = \begin{cases} 0 & \text{para } x < 0 \\ x & \text{para } x \geq 0 \end{cases}$$

- **Leaky ReLU:** es muy similar a la función ReLU, solo que los valores negativos son multiplicados por un coeficiente rectificador  $c$ . Su ecuación es:

$$f(x) = \begin{cases} c \cdot x & \text{para } x < 0 \\ x & \text{para } x \geq 0 \end{cases}$$

- **Softmax:** esta función se usa cuando se quiere obtener la representación probabilística de una red neuronal. La suma de todas las probabilidades da 1. Generalmente se utiliza en la clasificación de imágenes en redes neuronales convolucionales. Su ecuación es:

$$f(x)_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$



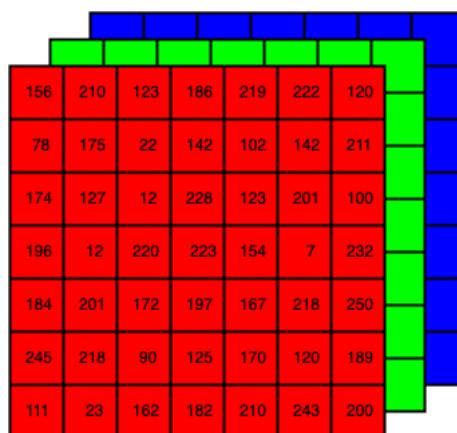
**Figura 16.** Representación de las funciones sigmoide, ReLU y Leaky ReLU, de izquierda a derecha.

Si la entrada a las neuronas activa las funciones, estas transmiten su salida a las siguientes neuronas de la red a las que este conectadas. Esta es la manera en la que todas las neuronas interactúan así. Según los pesos que tenga cada neurona se obtendrá una salida u otra. La arquitectura de una red neuronal vendrá dada según el caso que se este trabajando. No hay ninguna regla que establezca el número de neuronas para cada capa, es más una tarea de “afinado” de la red neuronal, en la que se van probando diferentes configuraciones hasta obtener los resultados deseados.

#### 1.2.4. REDES NEURONALES CONVOLUCIONALES (CNN)

Similar a las redes neuronales artificiales, están las redes neuronales convolucionales. Este tipo de redes neuronales trabajan con imágenes como entradas. Se usan principalmente para tareas de reconocimiento, clasificación y detección de diferentes objetos.

Lo primero que hay que tener en cuenta es como se representan las imágenes para que puedan ser introducidas a una CNN. Una imagen no es más que un array de 3 dimensiones, donde las dos primeras versiones almacenan valores entre 0 y 255, y la tercera dimensión son los tres canales de los colores rojo, verde y azul. Los valores almacenados en las dos primeras dimensiones representan la intensidad de cada color. El color final de un píxel se obtiene de la combinación de los valores de los tres canales de color.



**Figura 17.** Representación de una matriz RGB.

De esta forma, las imágenes no son más que números con los que una CNN puede trabajar. También es posible encontrarse con otro tipo de matrices, como puede ser una imagen que esté en escala de grises, cuya matriz sería de dos dimensiones, ya que solo representa un color (el negro) en sus distintas intensidades.

Una CNN coge una imagen en forma de matriz como entrada, la procesa, y devuelve un resultado relacionado a la clasificación de la imagen en una clase.

### 1.2.5. COMPONENTES DE UNA CNN

Para poder llegar a clasificar una imagen es necesario ir aplicando una serie operaciones que nos lleven hacia el resultado que indique la clase predicha. Para ello, vamos a ver en qué consisten las diferentes capas empleadas. Estas capas son:

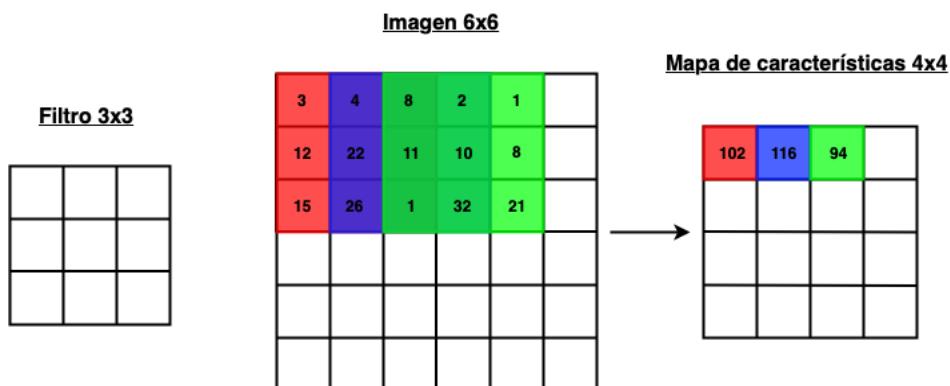
- Capa de convolución (Convolutional layer)
- Capa de rectificación (Rectification layer)
- Capa de agrupación (Pooling layer)
- Capa completamente conectada (Fully-connected layer)

Para que no haya ningún tipo de confusión, vamos a referirnos a estas capas por su nombre en inglés.

#### Convolutional layer

Como comentamos anteriormente, las imágenes se expresan como matrices de tres dimensiones. Estas imágenes en su formato original suelen ser demasiado grandes para extraer cualquier tipo de información. Es por ello por lo que lo primero que se hace es extraer las características de la imagen a través del uso de filtros.

Estos filtros tendrán un tamaño inferior en sus dos primeras dimensiones, pero la tercera permanecerá igual debido a que representa los tres canales de los colores. Cada filtro está diseñados para poder extraer alguna característica específica de la imagen. Diferentes filtros se van convolucionando sobre la imagen original generando mapas de características. En la Figura 18 podemos observar como se iría aplicando un filtro de 3x3 que va sumando el valor de todos los píxeles, con un desplazamiento de un píxel.



**Figura 18.** Aplicación de un filtro 3x3 que suma los valores de los píxeles de una imagen.

A la hora de diseñar un modelo que emplee capas de convolución, estas se emplean en diferentes niveles, para que así puedan ir aprendiendo características de diferentes niveles. Por ejemplo, las capas de convolución más próximas a la entrada serán capaces de detectar características sencillas, como pueden ser líneas o curvas, mientras que las capas que estén localizadas casi al final de la arquitectura de una CNN tendrán la capacidad de detectar características más avanzadas, como pueden ser figuras con una forma ya más específica.

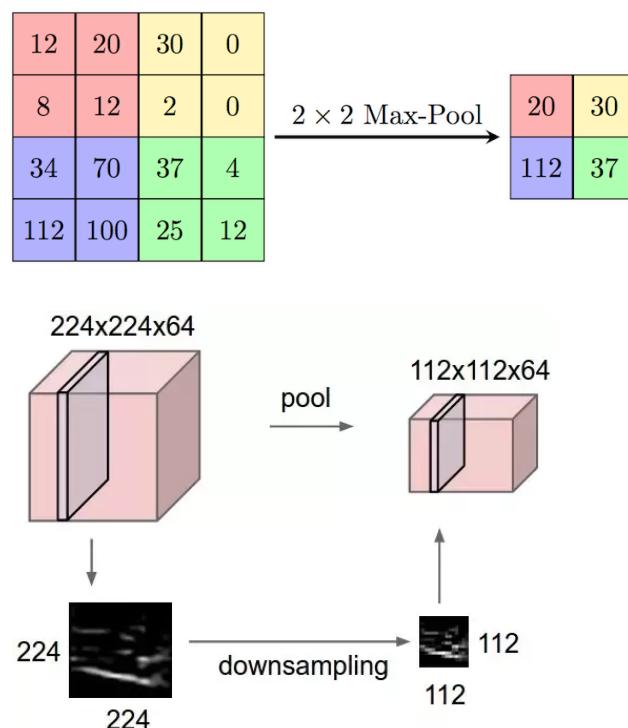
#### Rectification layer

Con los mapas de características obtenidos en la Convolutional layer, estos son pasados a esta capa para la creación de un mapa de activación, resultado de ir pasando los valores del mapa de características por una función de activación. La función de activación más común es la ReLU, aunque con anterioridad la función sigmoide era muy usada.

### Pooling layer

Esta capa es la responsable de reducir el espacio de las dimensiones (la dimensión de los tres canales de color no se reduce) de los mapas de características sin perder detalle de ellos. Uno de los motivos para querer reducir el espacio de las dimensiones es reducir la carga computacional requerida. Otro motivo por el que se emplea esta capa es proporcionar invarianza a cualquier traslación o desplazamiento que pueda ocurrir con las características. De esta forma, lo que se hace es reducir este espacio de las dimensiones conservando las características más importantes y eliminando los datos irrelevantes.

La operación para reducir el tamaño de la muestra se denomina Max-pooling. Para ello, se va aplicando un filtro, que suele ser de  $2 \times 2$ , sobre la imagen con un desplazamiento de 2 píxeles para evitar el solapamiento del filtro. En la Figura 19 se puede observar como se realiza esta operación sobre una matriz y como se vería el resultado.



**Figura 19.** Ejemplo de la operación Max-pooling sobre una matriz y un ejemplo real.

Las Pooling layer suelen ir siempre después de una Convolutional layer. Esta operación de pooling se aplica a todos los mapas de características generados por Convolutional layer, aunque antes de ello suelen aplicarse operaciones de rectificación empleando funciones de activación como puede ser la ReLU. Se generarán tantos mapas de características reducidos como los que proporciona la Convolutional layer que va justo detrás de esta capa.

### Fully-connected layer

Esta es la capa encargada de darle sentido a todas a las operaciones que se han ido realizando para extraer las características de la imagen. En esta capa todas las neuronas están conectadas a las funciones de activación de la capa anterior. De esta forma se obtiene un vector que es utilizado para realizar las clasificaciones. De forma resumida, lo que se pretende con esta capa es obtener una representación probabilística de pertenencia a las diferentes clases que hay definidas en el modelo.

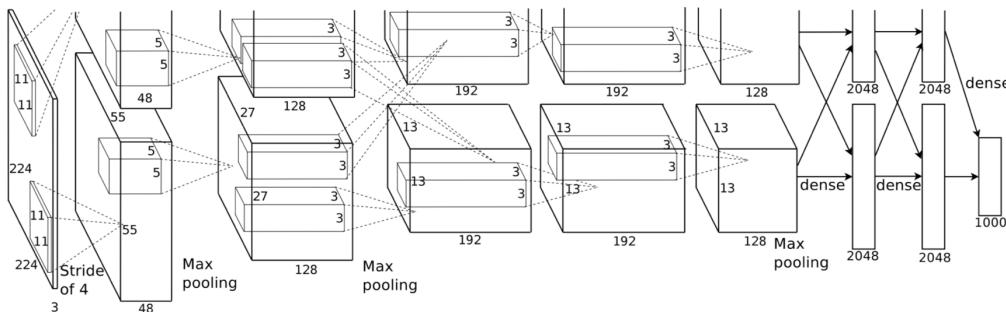
### 1.2.6. MODELOS MÁS FAMOSOS

Para finalizar, vamos a ver algunos de los modelos de redes neuronales convolucionales más conocidos.

#### AlexNet

Este modelo fue diseñado por Alex Krizhevsky [26], con el compitió en el desafío de ImageNet [3]. Para el entrenamiento de este modelo, Alex empleó el dataset de ImageNet preparado para el desafío ILSVRC, el cuál contiene alrededor 1000 imágenes para cada clase, sumando un total de 1.2 millones de imágenes de entrenamiento, 50.000 de validación y 150.000 para el test. Estas imágenes vienen etiquetadas con la clase a la que pertenecen.

La arquitectura del modelo esta compuesta de cinco Convolutional layers seguidas por tres Fully-connected layers. La salida de la última capa esta conectada a una función softmax que se encarga de distribuir las probabilidades sobre las 1000 clases.



**Figura 20.** Arquitectura del modelo AlexNet.

En cuanto a los resultados obtenidos, en la competición ILSVRC del año 2010, obtuvo el top-1 y top-5 en las tasas de error, con un 17.0% y 37.5% respectivamente.

#### VGG Net

VGGNet es el modelo diseñado por Karen Simonyan y Andrew Zisserman [27] que quedó subcampeón en la competición ILSVRC en en año 2014. La arquitectura de este modelo esta compuesta con 16 Convolutional layers, en las cuáles se aplican filtros de 3x3 con un desplazamiento de un píxel. La aplicación de filtros tan pequeños hace que aumente la profundidad del modelo si lo comparamos con el modelo AlexNet, el cual esta compuesto por 5 Convolutional layers. La decisión de emplear de filtros pequeños viene dada a que se observan mejores resultados respecto a los casos anteriores.

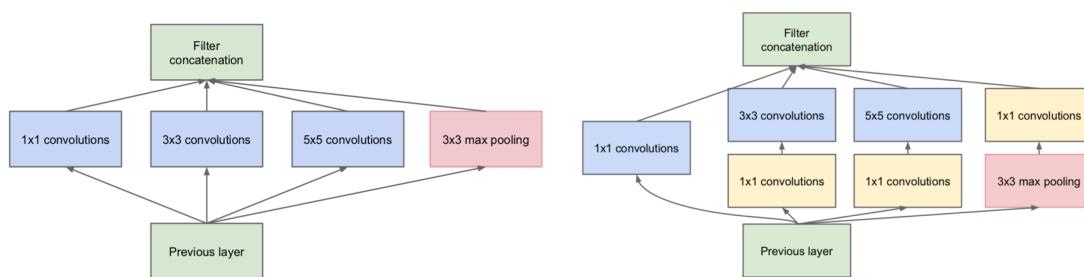
En cuanto a los resultados obtenidos, el modelo VGGNet quedó en segundo lugar en tareas de clasificación de imágenes con un error del 7.3%. Este modelo mejora mucho los resultados de las competiciones de años anteriores, pero en 2014 el modelo GoogleNet ganó la competición con un error del 6.7%.

Gracias a que este modelo ha aprendido a detectar las características que más se suelen emplear en problemas de visión artificial, se ha decidido usar VGG16 como modelo base de nuestro clasificador, ya que de esta forma evitamos tener que recopilar una gran cantidad de datos, además de necesitar una gran capacidad de computación durante un periodo largo de tiempo.

## GoogleNet

Visto el subcampeón de ILSVRC del año 2014, vamos a ver el modelo ganador. Este modelo fue diseñado por Google y consiguió el top-5 de tasa de error con un 6.67%. En cuanto a la arquitectura de este modelo, es muy diferente a la de los modelos que hemos visto hasta ahora. Se emplea convoluciones con filtros de tamaño 1x1 para reducir el tamaño del modelo, lo cuál reduce de forma muy notoria el número de operaciones necesarias cuando se trabaja con el modelo.

A diferencia de los modelos anteriores en los que el tamaño de cada Convolutional layer estaba fijado, en GoogleNet se emplean diferentes tamaños para la convolución la operación de Max-pooling. A través del uso de diferentes tamaños de las capas convolucionales es posible extraer diferentes características. Todos los mapas de características obtenidos se concatenan antes de ser pasados al siguiente módulo. La agrupación de todas estas operaciones se conoce como el módulo Inception.

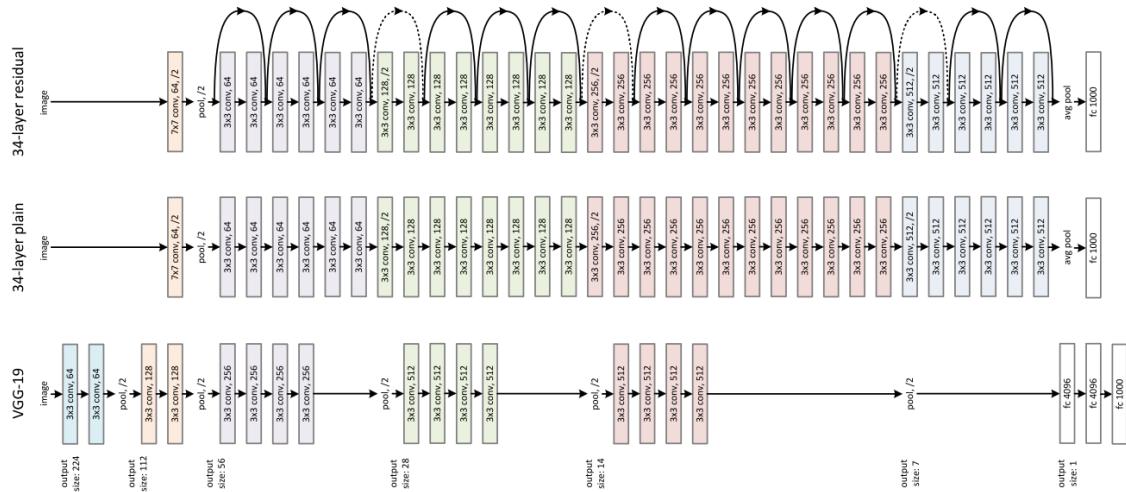


**Figura 21.** Representación del módulo Inception sin uso de filtros (a la izquierda) y empleando filtros 1x1 (a la derecha).

Otra diferencia respecto a los modelos anteriores es que las Fully-connected layers se localizaban al final del modelo. En el caso de GoogleNet la operación de *global average pooling* se realiza casi al final del modelo, promediando cada mapa de características de 7x7 a 1x1. Como resultado, el modelo cuenta con 22 capas en total.

## Res Net

Para finalizar, tenemos el modelo ResNet, también conocido como red neuronal residual, diseñado por Kaiming He. Si nos fijamos en los modelos que hemos ido viendo, podemos ver como cada vez iba aumentando el número de capas que estos tenían. Pero llega un punto que si se siguen añadiendo más entonces el gradiente se vuelve excesivamente pequeño, lo cuál deteriora el modelo. Para solucionar esto, los autores de este modelo decidieron añadir unas conexiones de salto que permiten saltarse una o más capas.



**Figura 22.** Comparativa entre los modelos VGG19, un modelo plano de 34 capas y un modelo residual de 34 capas.

Según el tamaño las dimensiones de las entradas y las salidas, se añade padding a estas si es necesario o se ajustan las dimensiones para hacerlas coincidir. En cuanto a los resultados obtenidos en el campeonato ILSVRC del año 2015, este modelo consiguió el primer puesto en la tarea de clasificación de imágenes, con un error del 3.57%.

**Tabla 4.**

Tasas de errores de los diferentes modelos de la competición ILSVRC.

method	top-5 err. ( <b>test</b> )
VGG [41] (ILSVRC'14)	7.32
GoogLeNet [44] (ILSVRC'14)	6.66
VGG [41] (v5)	6.8
PReLU-net [13]	4.94
BN-inception [16]	4.82
<b>ResNet (ILSVRC'15)</b>	<b>3.57</b>



## 1.3. DETECCIÓN DE OBJETOS

### 1.3.1. INTRODUCCIÓN A LOS ALGORITMOS DE DETECCIÓN DE OBJETOS

Hemos hecho un recorrido a lo largo del cual hemos comenzado viendo lo básico de lo que es la realidad aumentada, hasta lo que son las redes neuronales convolucionales y como funcionan. Se ha seguido este orden para llegar hasta este punto donde la intención es encontrar una manera para diseñar una aplicación de realidad aumentada que sea capaz de detectar peces en un vídeo y clasificarlos.

Hemos visto modelos de clasificación de imágenes, pero estos solo valen para decir a qué clase pertenece una entrada, no para localizar donde se encuentra un determinado objeto en una imagen. Es por ello por lo que se necesita de un algoritmo que sea capaz de detectarnos la localización de un determinado tipo de objeto, para que luego podamos clasificarlo.

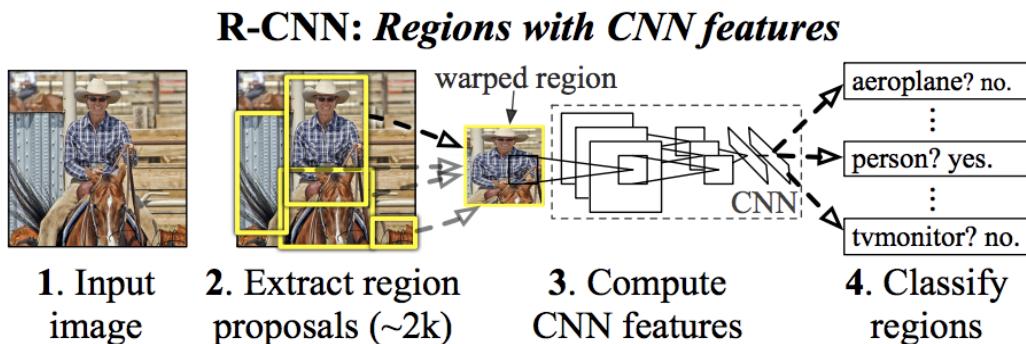
El objetivo es el poder dibujar una caja alrededor de los objetos que se van detectando, y extraerlos si es oportuno. Se nos pueden venir diferentes ideas de como implementar este caso. Una de ellas es la división de una imagen en sub-imágenes que se alimenten a una CNN y que cada sub-imagen nos diga si existe un objeto de los que se quiere detectar o no. El problema es que no todos los objetos que aparezcan en una imagen tendrán el mismo tamaño, o puede que al dividir la imagen un objeto resulte partido en dos. Con este problema se nos puede venir a la mente en dividir la imagen en un gran número de sub-imágenes con diferentes tamaños, pero esto empieza a inducir unas cargas computacionales que no son viables.

Por todas estas razones expuestas es necesario emplear otros métodos que sean capaces de realizar estas detecciones en un tiempo viable, aproximándose al tiempo real. Hoy en día existen múltiples algoritmos capaces de detectar objetos. Vamos a ver como han ido evolucionando estos hasta llegar al algoritmo empleado en esta aplicación, Yolo v3.

### 1.3.2. ALGORITMOS DE DETECCIÓN DE OBJETOS

#### 1.3.2.1. R-CNN

Ross Girshick es el autor de este algoritmo. Según explica el trabajo de Ross, el sistema de detección de objetos está dividido en tres módulos: el primero genera regiones de proposición independiente de categorías, el segundo es una gran CNN que extrae un vector de características de tamaño definido, y el tercero es un conjunto de máquinas de vectores de soporte lineales (SVM). En la Figura 23 se puede observar el proceso que sigue este sistema para llevara cabo las detecciones y sus clasificaciones.



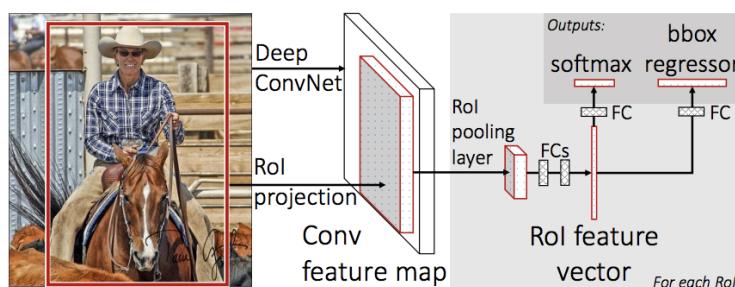
**Figura 23.** Esquema del sistema R-CNN.

Para llevar a cabo el proceso de detección, lo primero que se hace es extraer las regiones de propuesta, que suelen ser alrededor de 2000. Con las regiones propuestas, se alimenta la CNN, la cuál produce un vector de características de 4096 dimensiones. Este vector de características generado por la CNN se pasa a la SVM, la cuál se encarga de clasificar las detecciones.

El principal problema de R-CNN es que aparte de que tarde mucho tiempo en ser entrenado un modelo, las predicciones también tardan un periodo de tiempo demasiado grande (alrededor de los 50 segundos por imagen).

### 1.3.2.2. FAST R-CNN

El mismo autor que diseño R-CNN propuso un nuevo método para detectar y clasificar objetos. La aproximación es similar a la de R-CNN, solo que en vez de pasar las regiones de proposición a la CNN, se pasa la imagen para obtener el mapa de características. A través del mapa se obtienen las regiones de proposición, que son ajustadas a la dimensión correcta para ser pasadas por una RoI pooling layer. Esta última capa se encarga de realizar Max-pooling sobre entradas con tamaños no uniformes y producir pequeños mapas de características con unas dimensiones fijadas. Con estos mapas de características obtenidos de la RoI pooling layer se pasan Fully-connected layer para obtener un vector de características RoI. Por último, este vector se emplea en una capa softmax para clasificar las detecciones y obtener las posiciones de la caja que enmarca la detección.



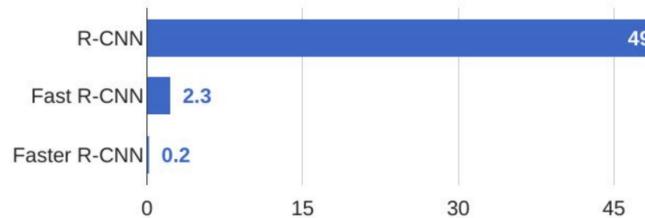
**Figura 24.** Esquema del sistema Fast R-CNN.

Esta vez se consiguen mejores tiempos de procesado de las imágenes (2 segundos aproximadamente) y en el entrenamiento. La principal causa de esto es que no se pasan las 2000 regiones de proposición.

### 1.3.2.3. FASTER R-CNN

Con el modelo Fast R-CNN se determinó que los algoritmos de proposición de regiones eran un cuello de botella, lo cual impedía obtener mejores resultados. Shaoqing Ren presentó una red de proposición de regiones (Region Proposal Network) que elimina este problema. De esta manera se consigue eliminar la parte encargada de presentar estas regiones propuestas y se deja que el propio sistema aprenda a predecir estas regiones.

Con este método ya es posible trabajar casi en tiempo real con imágenes, ya que las predicciones tardan alrededor de 0.2 segundos. En la Figura 25 podemos observar los diferentes tiempos de predicción de los modelos que hemos hablado hasta ahora.



**Figura 25.** Gráfica con los tiempos de predicción de los diferentes modelos en segundos.

### 1.3.2.4. YOLO

En último lugar tenemos Yolo, que es un algoritmo que funciona de forma totalmente diferente a los que hemos visto hasta ahora. Pero antes de ponernos hablar de este algoritmo vamos a hablar un poco de lo que es Darknet y que lo compone.

### 1.3.3. ¿QUÉ ES DARKNET?

Darknet es un framework escrito en C y CUDA. Entre las diferentes cosas que puede hacer destacan:

- Detección en tiempo real: ofrece un algoritmo capaz de detectar objetos en tiempo real.
- Clasificación de imágenes: ofrece clasificadores como ResNet y ResNeXt para ser usados.
- Redes neuronales recurrentes

Para su instalación tan solo requiere OpenCV y CUDA, aunque también hay librerías adicionales que pueden ser instaladas para mejorar ciertas tareas como puede ser el entrenamiento, a través de la librería cuDNN.

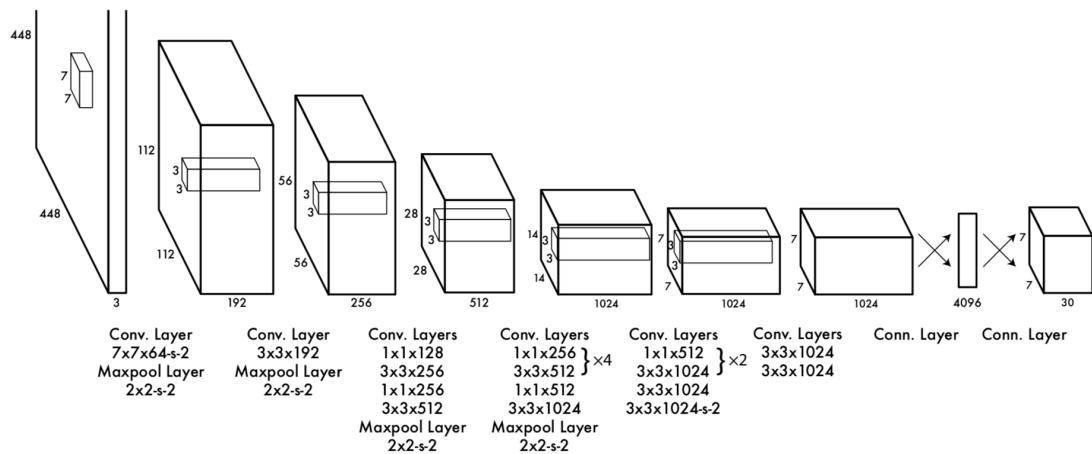
De todo esto, lo que nos interesa a nosotros es el algoritmo de detección en tiempo real Yolo, siendo más específicos la versión 3.

### 1.3.4. YOU ONLY LOOK ONCE (YOLO)

Como se comentó en los anteriores algoritmos de detección, estos hacían uso de las denominadas regiones de proposición, para lo que hacia falta siempre dos redes: una para calcular la posición de la caja que enmarca el objeto, y otra para clasificar el objeto detectado. Para el algoritmo propuesto por Joseph Redmon solo se emplea una sola red neuronal.

El funcionamiento del algoritmo Yolo es muy simple: una sola red neuronal convolucional predice la caja del objeto y la probabilidad de pertenencia a una clase de esa caja. A diferencia del resto de algoritmos, Yolo mira toda la imagen y no realiza regiones de proposición. La imagen que se le pasa al modelo es dividida en celdas, y si el centro de un objeto cae en esa celda entonces esa celda es la responsable de detectar ese objeto. Gracias a todas estas características es posible analizar imágenes en tiempo real, a unos 45 fotogramas por segundo. Es cierto que puede detectar objetos rápidamente, pero se encuentra problemas a la hora de detectar objetos pequeños.

La arquitectura de este modelo esta compuesta por 24 Convolutional layers seguidas por dos Fully-connected layers. En vez de emplear módulos Inception como pasaba en el modelo GoogleNet, simplemente se emplean capas de reducción de 1x1 seguidos capas de convolución con filtros 3x3. Cuando se entrena el modelo, la resolución de las imágenes es de 224x224, y luego se cambia a 448x448 para las detecciones.



**Figura 26.** Arquitectura de Yolo compuesta por 24 Convolutional layers seguidas de 2 Fully-connected layers.

Este modelo presenta una serie de desventajas. La primera es la que ya hemos comentado sobre la detección de objetos pequeños. En segundo lugar, esta la limitación en cuanto al número posible de detecciones por celda. Al dividirse la imagen en celdas, cada celda puede predecir como máximo dos cajas de objetos, las cuales solo pueden tener una clase. Esto supone un problema si es esta detectando objetos pequeños y que además están agrupados.

### 1.3.5. YOLO V2

En esta versión de Yolo se intenta mejorar la localización de objetos, sin perder la precisión de la clasificación. Para lograr estas mejoras se han hecho los siguientes cambios/adiciones:

- **Batch normalization:** consiste en normalizar las entradas a cada capa del modelo, lo cuál mejora el rendimiento y los resultados del modelo en general.
  - **Clasificador de alta resolución:** en la primera versión del algoritmo se entrenaba el modelo con una resolución de 224x224 para luego cambiarla a 448x448 para las detecciones. En esta versión del algoritmo lo que se hace es ajustar la red de clasificación a 448x448 durante 10 épocas para que los filtros funcionen mejor con entradas de mayor resolución.
  - **Anchor boxes:** este es uno de los cambios más destacables de esta versión. Se eliminan las Fully-connected layers del final y se emplean las denominadas anchors (anclas) para realizar las detecciones. La tarea de estas anclas es establecer una serie de tamaños distintos para las cajas empleadas en las detecciones, de esta forma no hay

que predecir estas cajas, si no que se predice el desplazamiento de cada una de estas anclas.

- **Tamaño de los clusters:** el problema de usar estas anclas es que están son elegidas de forma manual. El modelo las puede ajustar, pero si se establece una serie de anclas con mayor similitud a las cajas del dataset de entrenamiento, le será más sencillo al modelo realizar detecciones buenas. Para ello, se usa el algoritmo k-means, el cuál devolverá las dimensiones de estas anclas según el número de clusters establecidos.
- **Predicción de la localización directa:** en las etapas tempranas del entrenamiento se encuentran problemas con la localización de las coordenadas de las anclas. Para solventar esto se predicen las coordenadas relativas a una celda. En la primera versión se comentó que en cada celda se podía realizar como mucho dos detecciones por celda. En este caso, por cada celda se predicen cajas.
- **Características de grano fino:** las detecciones se hacen sobre un mapa de características de 13x13. Para ganar algo más de precisión con objetos más pequeños se aumenta este mapa a 26x26.
- **Entrenamiento con diferentes resoluciones:** cada 10 batches el modelo cambia la resolución de las imágenes que recibe como entrada, siempre siendo múltiplo de 32. De esta forma las resoluciones se comprenden entre 320 y 608. Esto hace que el algoritmo aprenda a predecir de forma correcta para diferentes resoluciones.

Con estos cambios se consigue mejorar la velocidad del algoritmo y su precisión. Según si se desea más precisión o menos se puede sacrificar una de estos por el otro, es decir, si queremos más velocidad tendremos que reducir la resolución de las imágenes pasadas, lo cual reducirá la precisión, y viceversa. De media el algoritmo es capaz de procesar 45 fotografías por segundo.

Para la extracción de características se emplean 19 Convolutional layers y 5 Max-pooling layers. En gran parte de las Convolutional layers se emplear filtros de tamaño 3x3, las cuales suelen ir seguidas por un filtro de tamaño 1x1 para comprimir las características y pasarlas a la siguiente Convolutional layer de tamaño 3x3.

Type	Filters	Size/Stride	Output
Convolutional	32	3 × 3	224 × 224
Maxpool		2 × 2/2	112 × 112
Convolutional	64	3 × 3	112 × 112
Maxpool		2 × 2/2	56 × 56
Convolutional	128	3 × 3	56 × 56
Convolutional	64	1 × 1	56 × 56
Convolutional	128	3 × 3	56 × 56
Maxpool		2 × 2/2	28 × 28
Convolutional	256	3 × 3	28 × 28
Convolutional	128	1 × 1	28 × 28
Convolutional	256	3 × 3	28 × 28
Maxpool		2 × 2/2	14 × 14
Convolutional	512	3 × 3	14 × 14
Convolutional	256	1 × 1	14 × 14
Convolutional	512	3 × 3	14 × 14
Convolutional	256	1 × 1	14 × 14
Convolutional	512	3 × 3	14 × 14
Maxpool		2 × 2/2	7 × 7
Convolutional	1024	3 × 3	7 × 7
Convolutional	512	1 × 1	7 × 7
Convolutional	1024	3 × 3	7 × 7
Convolutional	512	1 × 1	7 × 7
Convolutional	1024	3 × 3	7 × 7
Convolutional	1000	1 × 1	7 × 7
Avgpool		Global	1000
Softmax			

**Figura 27.** Arquitectura de Darknet 19, usado en el algoritmo Yolo v2.

### 1.3.6. YOLO V3

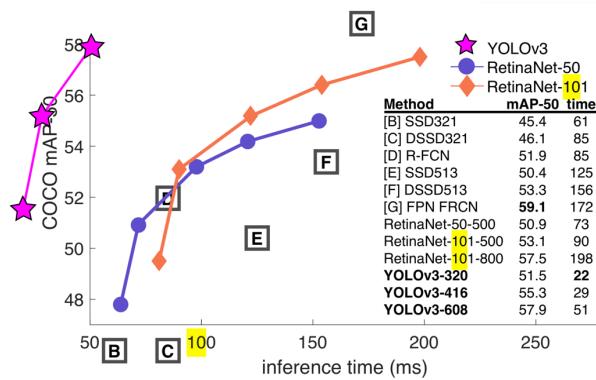
Esta última versión de Yolo pretende sacrificar algo de velocidad por precisión. Con la segunda versión se podían alcanzar velocidades de 45 FPS, mientras que en esta se llega a 30. En la versión antigua todavía se seguía teniendo problemas con la detección de objetos pequeños.

El diseño de este nuevo modelo se ha llevado a cabo usando Yolo v2, Darknet 19 y prácticas empleadas en otros modelos, como puede ser ResNet, donde se hace el uso de capas residuales con conexiones de salto hacia otras capas. Como pasaba en Yolo v2, se emplean las anclas para predecir las coordenadas de las detecciones. Una de las mejoras más importantes es que las predicciones se realizan a tres escalas diferentes, es decir, según la escala empleada se podrá detectar objetos más grandes o pequeños. Para implementar esto, se emplean mapa de características de tres tamaños diferentes, teniendo un desplazamiento de píxeles de 32, 16 y 8. De esta forma, si se pasa una imagen de 608x608 (siempre múltiplos de 32), entonces el primer mapa de características tendrá un tamaño de 19x19, el segundo de 38x38, y el tercero de 76x76. Así, la imagen se va pasando por las diferentes capas reduciéndola hasta que se llega a la primera capa de detección, que emplea un desplazamiento de 32 píxeles. Tras la detección se vuelve a ampliar la imagen hasta llegar a la siguiente capa de detección, la cuál tendrá un desplazamiento de 16 esta vez. Lo mismo vuelve a suceder una vez más hasta llegar a la última capa de detección, donde se emplea un desplazamiento de 8. En cada una de estas capas de detección se predicen 3 cajas empleando 3 anclas, lo cuál suma un total de 9 anclas empleadas por celda. De esta forma, en las capas de detección más tempranas se detectan los objetos más grandes, mientras que en las últimas se detectan los más pequeños. De esta forma, si pasamos como entrada una imagen de 608x608, el modelo genera 21630 cajas (((76x76) + (38x38) + (19x19)) x 3). Esto es un número muy grande, y lo más probable es que en una imagen haya un número de objetos cercano a las dos cifras. ¿Entonces qué cajas se escogen como correctas? Esto se consigue a través del uso de una puntuación que indica en qué grado realmente se está detectando un objeto. Si esta puntuación no supera un umbral, entonces la caja se ignora.

Se hace uso de la arquitectura Darknet 53, compuesta de 53 Convolutional layers como bien podemos deducir de su nombre. Sobre esta arquitectura se apilan otras 53 capas, teniendo un total de 106 capas en la arquitectura de Yolo v3. Las capas de detección se encuentran en las posiciones 81, 94 y 106.

Type	Filters	Size	Output
Convolutional	32	3 x 3	256 x 256
Convolutional	64	3 x 3 / 2	128 x 128
1x	Convolutional	32	1 x 1
	Convolutional	64	3 x 3
	Residual		128 x 128
	Convolutional	128	3 x 3 / 2
		64	64 x 64
2x	Convolutional	64	1 x 1
	Convolutional	128	3 x 3
	Residual		64 x 64
	Convolutional	256	3 x 3 / 2
		32	32 x 32
8x	Convolutional	128	1 x 1
	Convolutional	256	3 x 3
	Residual		32 x 32
	Convolutional	512	3 x 3 / 2
		16	16 x 16
8x	Convolutional	256	1 x 1
	Convolutional	512	3 x 3
	Residual		16 x 16
	Convolutional	1024	3 x 3 / 2
		8	8 x 8
4x	Convolutional	512	1 x 1
	Convolutional	1024	3 x 3
	Residual		8 x 8
	Avgpool		Global
	Connected		1000
	Softmax		

Figura 28. Arquitectura Darknet 53.



**Figura 29.** Gráfica con los resultados obtenidos de Yolo v3 comparado con los de otros algoritmos. Se muestra el tiempo de detección y la precisión media de los modelos.



# DESARROLLO PRACTICO



## 2.1. INTRODUCCIÓN

La intención de este trabajo es aplicar el campo de la visión artificial a la biología marina. ¿El porqué? La principal causa es el estudio de poblaciones marinas. La superficie de la tierra esta cubierta por los océanos aproximadamente en un 71%. Gran parte de estos océanos se encuentran sin explorar, y probablemente existan muchas especies que aun no se sepa de su existencia. Actualmente se sabe que existen alrededor de 33.000 especies de peces, de las cuales unas 20.000 corresponden a especies marinas [2]. Este es un gran número, y a veces puede resultar complicado identificar ciertas especies ya que no se esta en contacto de la misma forma que con la vida terrestre.

El campo al que pertenece esta aplicación es al de Machine Learning, o visión artificial si somos más específicos. La idea es diseñar un sistema que sea capaz de detectar y clasificar peces a partir de unos datos que se le proporcione, sin enseñarle específicamente como tiene que hacerlo. La idea de esta aplicación es proporcionar una herramienta útil al mundo de la biología marina que ayude al estudio de poblaciones marinas, así como el muestreo de estas. Por otro lado, también es posible llevar este modelo a mundos como puede ser el buceo recreativo, donde se podría integrar en las gafas de buceo para obtener información del entorno.

De esta forma se pretende desarrollar una primera parte que se capaz de generalizar para la detección de cualquier pez, y posteriormente se desarrollará otra parte que será la encargada de clasificar las especies de un entorno cerrado como ejemplo. Como no existen datasets con fotografías hará falta crear uno para poder llevar a caso este desarrollo. Al final del todo veremos a los resultados que se ha llegado, que mejoras se pueden hacer, y como seguir con este proyecto para poder seguir mejorándolo.



## 2.2. APLICACIONES EXISTENTES

En la actualidad, existen diversas aplicaciones de realidad aumentada y diferentes campos que tiene cierto grado de similitud al caso que se está desarrollando.

En cuanto a aplicaciones relacionadas con el ámbito marino, hasta la fecha solo existen manuales, guías o libros en los que se puede consultar información sobre diversas especies, pero nada que esté relacionado con el campo de la realidad aumentada.

A continuación, vamos a ver algunos casos similares al propuesto donde se emplea la realidad aumentada para solventar ciertos problemas o ayudar en tareas específicas.

### Detección de caras en los Smartphones

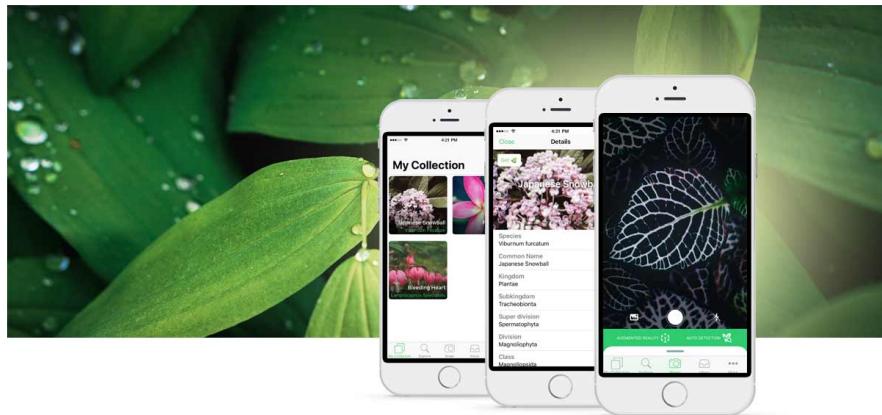
En la actualidad, compañías como Apple tienen integrado en sus dispositivos móviles y tablets mecanismos capaces de detectar caras. A través de esto, lo que hacen es extraer todos los rostros de las fotografías, y realizan una clasificación de estas para así poder crear subgrupos de imágenes en lo que se asocian a una persona a dichas imágenes en las que aparece su rostro.



**Figura 30.** Ejemplo de creación de álbumes a través de la detección de caras.

### Aplicación para la clasificación de plantas

Aplicaciones como PlantSnap o PictureThis son capaces de identificar una gran variedad de plantas a través fotografías, desde plantas convencionales y flores a cactus, plantas carnosas o setas. Todo este proceso de clasificación lo logran a través del Machine Learning, donde usan 150 millones de imágenes de 585.000 especies distintas.



**Figura 31.** PlantSnap [36].

Esta aplicación no es realmente una aplicación de realidad aumentada, ya que lo que hace es procesar una imagen para darnos un resultado, no lo hace en el cuasi-instante en el que se apunta a una planta. Pero es cierto que tiene un grado de similitud al proyecto propuesto, ya que una de las partes que se quiere hacer es la de clasificar especies de peces que van apareciendo por la pantalla.

#### Libros, guías o manuales

Refiriéndonos al campo de la biología marina, actualmente no existe ninguna aplicación capaz de aportar información a través de fotografías o vídeos sobre distintas especies subacuáticas. La única forma de poder obtener algún tipo de información es emplear libros relacionados con el tema, o guías sobre diversas zonas en las que se habla de la fauna local del ecosistema submarino.

Tanto en internet como en muchas librerías físicas es posible encontrar bibliografía relacionada con el tema sobre zonas geográficas específicas.

## 2.3. REQUISITOS

### 2.3.1. REQUISITOS DE USUARIO

**Tabla 5.**

Requisitos de usuario.

ID	Descripción	Prioridad/Importancia	Tipo
RU01	El sistema debe de aceptar vídeos	Alta	Funcional
RU02	El sistema tiene que detectar los peces de los vídeos o fotografías	Alta	Funcional
RU03	El sistema tiene que clasificar las especies de una zona definida	Alta	Funcional
RU04	El sistema tiene que producir una salida con la información en la forma deseada	Alta	Funcional
RU05	El usuario debe poder elegir el grado de aceptación de la clasificación	Medio	Funcional
RU06	El sistema debe mostrar la información sobre el vídeo o la fotografía	Medio	Funcional
RU07	La salida del vídeo tiene que tener los mismos FPS que la entrada	Alto	No funcional

### 2.3.2. REQUISITOS DE SISTEMA

#### Aceptación de vídeos.

Código: RS01

Deriva de: RU01

Tipo: funcional

La aplicación tiene que ser capaz de aceptar los formatos de vídeo más usados en internet. Cualquier otro formato no definido tiene que ser rechazado.

Para vídeo, los siguientes formatos tienen que ser aceptados:

- MP4
- AVI

#### Detección de peces.

Código: RS02

Deriva de: RU02

Tipo: funcional

Cuando al sistema se le pase un vídeo, este tiene que ser capaz de detectar cualquier pez que aparezca en la fuente original. Para que la detección sea visible al usuario, esta tiene que estar señalada con una caja de algún color que facilite su distinción.

**Clasificación de especies.****Código: RS03**

Deriva de: RU03

Tipo: funcional

El sistema debe tener una funcionalidad capaz de clasificar los peces según su especie. Para Realizar estas clasificaciones, el sistema poseerá diferentes clasificadores según la zona donde este se emplee.

La clasificación puede realizarse ya sea sobre una única imagen que contenga una especie bien delimitada, o ya sea sobre las detecciones que el propio sistema realiza para encontrar todos los peces de una fuente proporcionada

Para este caso, se creará un clasificador con las siguientes especies, todas pertenecientes a un entorno cerrado (un acuario):

- Acanthurus Sohal
- Amphirion Ocellaris
- Coris Formosa
- Halichoeres Adustus
- Naso Elegans
- Naso Vlamingii
- Paracanthurus Hepatus
- Pseudanthias Squamipinnis
- Pterapogon Kauderni
- Siganus Vulpinus
- Sphaeramia Nematoptera
- Zebrasoma Flavescens
- Zebrasoma Veliferum

**Formatos presentación de la información obtenida****Código: RS04**

Deriva de: RU04

Tipo: funcional

Cuando se haya analizado un video, toda la información obtenida tiene que mostrarse de la forma que el usuario escoja. Habrá dos formas diferentes de presentar esta información:

- El vídeo procesado con las detecciones en el, y en un lado una ventana donde se muestra información obtenida de las clasificaciones que se realizan cada cierto intervalo de tiempo.
- Toda la información será mostrada sobre el propio vídeo: tanto las detecciones, como la clasificación, y más información que se deseé mostrar.

**Grado de aceptación clasificaciones****Código: RS05**

Deriva de: RU05

Tipo: funcional

El sistema tiene que permitir elegir el grado de aceptación de las clasificaciones realizadas. Si una clasificación se da con un intervalo de 0 a 100 porciento, si el resultado que devuelve el sistema supera el umbral definido por el usuario, entonces se acepta la clasificación y se muestra la información.

En caso de no superar el umbral establecido, entonces dicha detección se mostrara como dudosa.

**Información de las detecciones****Código: RS06**

Deriva de: RU06

Tipo: funcional

El sistema tendrá que mostrar cierta información sobre las detecciones que obtenga el sistema. Alguna información que puede ser mostrada es:

- Familia de la especie
- Localización
- Dieta/Alimentación
- Profundidad localización
- Estado de conservación (amenaza de extinción)

**Mismos FPS salida que de entrada en un vídeo****Código: RS07**

Deriva de: RU07

Tipo: no funcional

Cuando se procesa un vídeo, puede haber cierta lentitud debido a que el equipo no es lo suficientemente potente, por lo que la salida procesada puede que se vea afectada en cuanto a fotogramas por segundo nos referimos. Por ello, es necesario asegurarse de que esta velocidad sea ajustada correctamente una vez el vídeo haya sido procesado.



## 2.4. BOCETOS DEL SISTEMA

### 2.4.1. BOCETOS DE LOS BLOQUES DEL SISTEMA

Una vez vistos los requisitos, se representará de forma gráfica como será el sistema para que se pueda entender de forma rápida sin indagar muy afondo en las características de este.

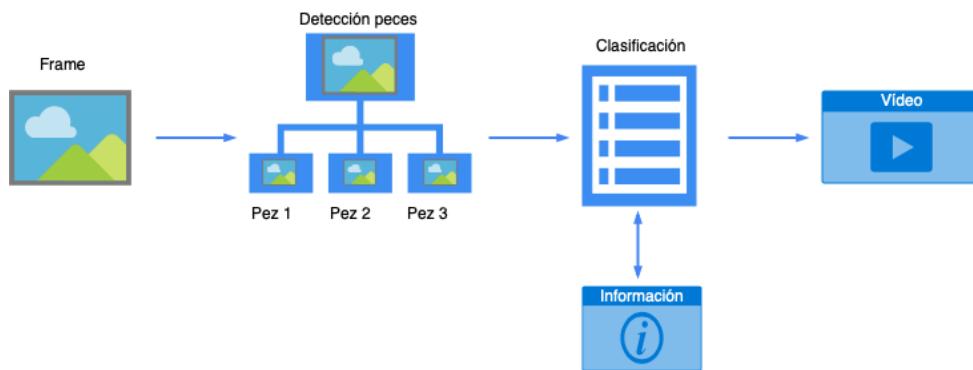


**Figura 32.** Representación sencilla del sistema.

En la Figura 32 se puede observar una sencilla representación gráfica que resume el funcionamiento del sistema propuesto. Básicamente, al sistema se le proporciona como entrada un vídeo, del cuál se van extrayendo los fotogramas individualmente para ser procesados en el sistema. Como salidas, hay dos posibilidades:

- El vídeo procesado con toda la información implícita en este.
- El vídeo con una ventana donde se muestran las detecciones, y otra ventana con la información de las detecciones, la cuál es obtenida a través de la clasificación.

Si indagamos un poco más profundo dentro de la aplicación, en la Figura 33 se puede ver como funcionaría el sistema internamente.



**Figura 33.** Representación del funcionamiento interno del sistema.

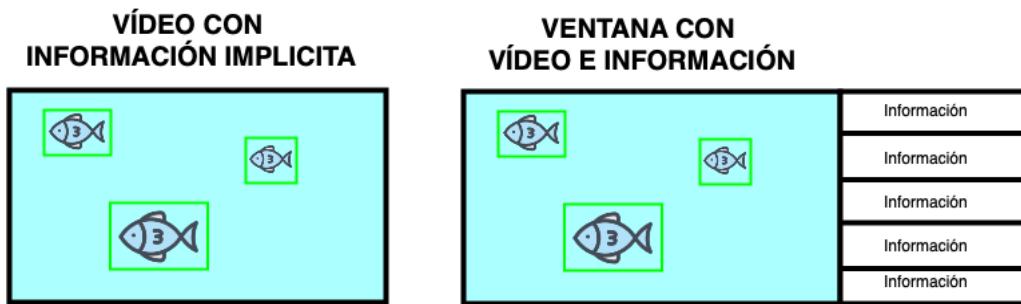
Podemos observar como se alimenta al sistema con fotogramas el cuál, primeramente, detecta todos los peces dentro de este, extrayéndolos en cajas a través de coordenadas. Una vez que se han obtenido todas las detecciones del fotograma, se alimenta al clasificador con estas detecciones, proporcionándole las imágenes recortadas empleando las coordenadas.

A través del clasificador se obtienen las especies de cada detección. Conociendo estas especies, es posible anexar a cada detección relativa a la especie. Toda esta información se muestra en un vídeo.

De esta forma hemos conseguido plasmar el sistema de tal forma que se pueda entender sin gran conocimiento sobre el tema.

#### 2.4.2. BOCETOS DE LAS SALIDAS DEL SISTEMA

Ahora vamos a pasar a ver como tendrían que ser las salidas en forma de bocetos. Las dos salidas planteadas se pueden ver en la Figura 34.



**Figura 34.** A la izquierda salida con información implícita en el vídeo, y a la derecha salida con una ventana que contiene el vídeo y la información.

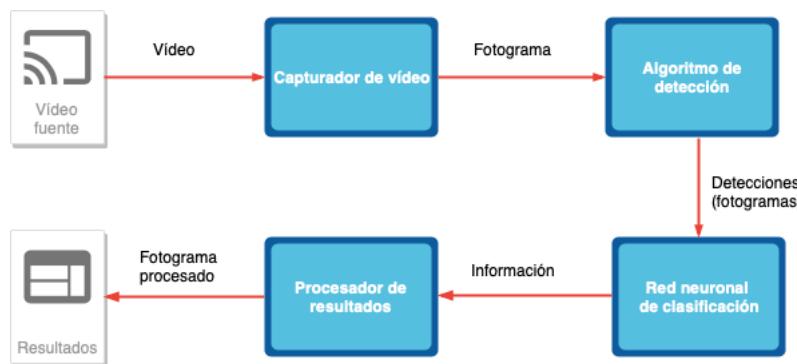
## 2.5. ARQUITECTURA DEL SISTEMA

### 2.5.1. DESCRIPCIÓN DE LA ARQUITECTURA

Ya se ha cubierto de forma genérica la descripción del sistema, ahora pasaremos a hacer una descripción más exhaustiva de este. Para esta descripción, vamos a dividir la aplicación en cuatro módulos:

- Suministrador de vídeo
- Algoritmo de detección YOLO
- Red neuronal clasificadora de imágenes
- Procesamiento de la información para su presentación

El diagrama de los bloques se puede ver en la Figura 35.



**Figura 35.** Diagrama de bloques del sistema.

En primer lugar, tenemos una fuente de información, que en este caso es un vídeo. Este vídeo es suministrado al bloque capturador de vídeo, encargado de realizar la captura de este, y poder suministrar fotogramas al siguiente bloque. Este primer bloque es necesario para poder tratar el vídeo fuente como una secuencia de fotogramas, los cuales son necesarios para nuestro sistema. Además de separar el vídeo en fotogramas, posee otras operaciones que serán de utilidad para el funcionamiento del sistema.

Después del bloque capturador de vídeo viene el bloque en el que se localiza el algoritmo de detección. Como entrada recibe fotogramas, los cuales procesa y extrae detecciones. Como salida produce una serie de coordenadas sobre la imagen pasada como entrada, o bien otros fotogramas, producto de recortar trozos de la imagen original usando las coordenadas obtenidas.

Estos fotogramas recortados de la imagen original son enviados al siguiente bloque, donde se localiza la red neuronal de clasificación (red neuronal convolucional para ser más exactos). Cada entrada individual producirá una salida, la cual indicará a la clase que pertenece, de la cual obtendremos más información de ella gracias a su clasificación.

Por último, esta clasificación con la información es enviada al último bloque, donde se pone en todo en conjunto y se crean los resultados. Estos resultados consisten básicamente en combinar cada fotograma original con la información obtenida, y todos estos fotogramas generarán el vídeo como resultado final.

Una vez visto como están conectados estos cuatro bloques, vamos a pasar a ver cada bloque en profundidad (que operaciones realiza, que herramientas se emplean, hardware necesario, etc.).

### 2.5.1.1. Suministrador de vídeo

Como comentamos con anterioridad, lo primero que hay que hacer es suministrar al sistema el vídeo con el cuál trabajará. Para ello, se ha diseñado una clase utilizando la librería OpenCV, la cuál nos permite operar con vídeos y imágenes.

Esta clase nos permite realizar operaciones como las que se detallan a continuación:

- **Abrir una captura:** a través del nombre o ruta de un archivo de vídeo, se abre una captura de vídeo de la cuál será posible ir extrayendo fotogramas de forma secuencial a través de otro método.
- **Obtener el siguiente fotograma:** una vez se tiene abierta una captura de vídeo, es posible ir llamando a un método para ir pidiendo el siguiente fotograma del vídeo, además de informarnos si el vídeo ha terminado o no.
- **Obtener los fotogramas por segundo:** es posible saber los fotogramas por segundo del vídeo que se está procesando.
- **Sacar el número total de fotogramas del vídeo:** esto puede ser empleado para asegurar de que un vídeo se procesa completamente.
- **Ajuste de resolución:** se puede ajustar tanto la anchura como la altura del vídeo para poder mostrar el tamaño deseado.
- **Calculo de retardo:** esta operación es importante, ya que cuando un vídeo es procesado, aparecen tiempos de computo que reducen los fotogramas por segundo del vídeo original. Para solventar esto, cuando ya se ha procesado un vídeo, a partir de la velocidad obtenida y de la velocidad de fotogramas original se computa el retraso para así poder ajustar la velocidad deseada en el vídeo originado.

Con todas estas operaciones se garantiza el suministro de inputs al siguiente bloque del que esta compuesto el sistema.

### 2.5.1.2. Algoritmo de detección (YOLO v3)

Esta parte es la base de todo el sistema. Si se obtienen buenos resultados en este bloque se conseguirá que el resto del proceso funcione correctamente, de lo contrario se desencadenaría una sucesión de errores que harían que el sistema produjese resultados no válidos.

Para esta parte se han usado las siguientes herramientas:

- Compilador gcc y g++
- Instalación OpenCV (versión 3.4 o inferior) en el sistema operativo
- Instalación de OpenCV para Python, preferiblemente en un entorno virtual
- Plataforma de computación CUDA de Nvidia
- Librería cuDNN de Nvidia para acelerar operaciones de Deep Learning usando GPU (en el caso de que se emplee tarjeta gráfica, que es el caso)
- Tarjeta de gráfica para mejorar las velocidades de procesamiento, aunque también es posible el uso de CPU
- Herramienta de etiquetado de imágenes [38]
- Buscador de imágenes de Google

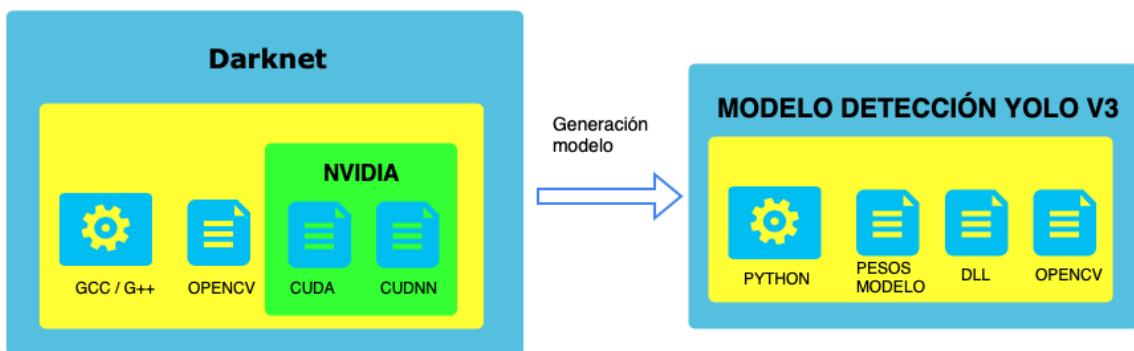
Lo primero y más básico de todo es tener los compiladores de C y C++, que son gcc y g++. Esto es necesario porque Darknet (el framework de redes neuronales open source que contiene YOLO) está escrito en C y CUDA. C++ es necesario porque ciertas librerías de OpenCV hacen uso de este.

Por otro lado, al estar trabajando con imágenes, es necesario el uso de la librería OpenCV. Para ello, son necesarias dos instalaciones: una a nivel de sistema operativo y otra para su uso en Python. La instalación a nivel de sistema operativo se debe a que el algoritmo YOLO está escrito

en C, por lo que usa la librería de OpenCV, escrito todo en C. En cambio, una vez que este listo el algoritmo para su uso, se utilizará este como una librería dinámica de enlace, de tal forma que el resto del tratamiento se realizará con Python. Es por ello por lo que es necesario tener otra instalación de OpenCV para Python.

Además, en este bloque se hará uso de GPU para aprovechar el paralelismo que se puede lograr, mejorando mucho los tiempos de proceso. Al estar trabajando con tarjetas gráficas de Nvidia, es necesario tener instalado CUDA, que es una plataforma de computación en paralelo que nos permite usar la potencia de una GPU para ejecutar operaciones en ella. Además de CUDA, también se hará uso de cuDNN, una librería de CUDA diseñada para mejorar las velocidades de la GPU para redes de Deep Learning.

Por último, en el caso de que se este usando Windows, es necesario instalar Microsoft Visual Studio para poder compilar el framework.



**Figura 36.** Esquema del funcionamiento de Darknet para la generación de modelos.

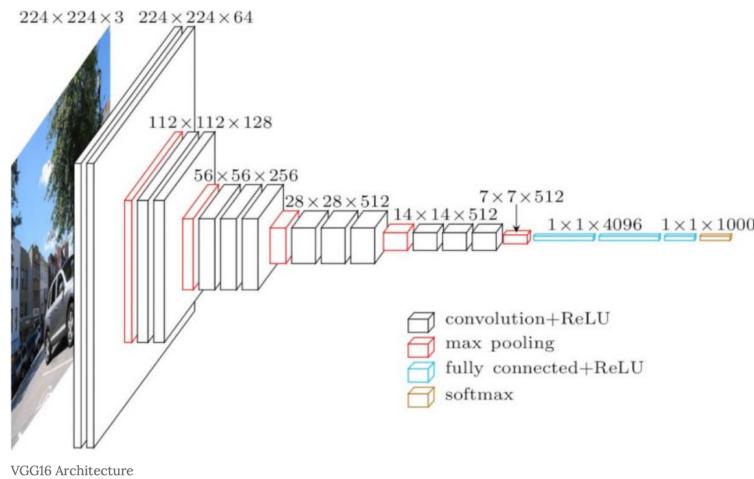
En la Figura 36 se puede observar el esquema de como se generaría un modelo de detección. Hay que destacar que hay que proporcionarle datos al algoritmo para poder generar el modelo, pero esta parte no se ha añadido al esquema debido a que se comentará más tarde en la parte correspondiente a la implementación.

### 2.5.1.3. Red neuronal de clasificación (CNN)

Para el proceso de clasificación de imágenes, se van a emplear librerías que normalmente son usadas en el campo de Data Science y Machine Learning.

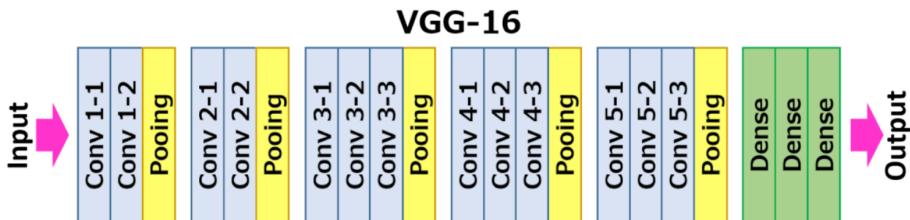
- Keras: API escrita en Python que trabaja ejecutándose sobre TensorFlow, CNTK y Theano, librerías muy conocidas en el campo del Deep Learning.
- Numpy: librería para trabajar con arrays multidimensionales de gran tamaño, ofreciendo gran diversidad de funciones.
- Pandas: librería para uso de estructuras de datos y herramientas de análisis

Además de ello, se va a partir de un modelo pre entrenado para generar nuestro clasificador, de lo contrario necesitaríamos una gran cantidad de datos para poder generar un buen clasificador, además de una gran capacidad de computo y tiempo. Para nuestro clasificador, se hará uso de la red neuronal convolucional VGG16, diseñada por K. Simonyan y A. Zisserman [6].



**Figura 37.** Arquitectura del modelo VGG16 [7].

Si observamos la Figura 37, vemos que el modelo tiene como entrada una imagen de 224x224 de 3 canales (RGB). Esta entrada es pasada por una serie de capas convolucionales con un filtro de 3x3, lo suficientemente pequeño para captar la noción de arriba/abajo, izquierda/derecha y centro. Para ir reduciendo el tamaño de la muestra conservando los detalles importantes, se van usando capas de “max-pooling”, las cuales suelen ir seguidas de otra capa de convolución (menos la última). Últimas capas de convolución, que tienen un tamaño de 7x7x512, son seguidas por 3 “fully-connected layers”, las dos primeras con un tamaño de 4096, y la última con un tamaño de 1000, que es el número de clases del modelo. Al final de la arquitectura podemos encontrar la capa “softmax”, la cuál se encarga de recibir el vector de la capa anterior, y lo normaliza a una distribución de probabilidad.



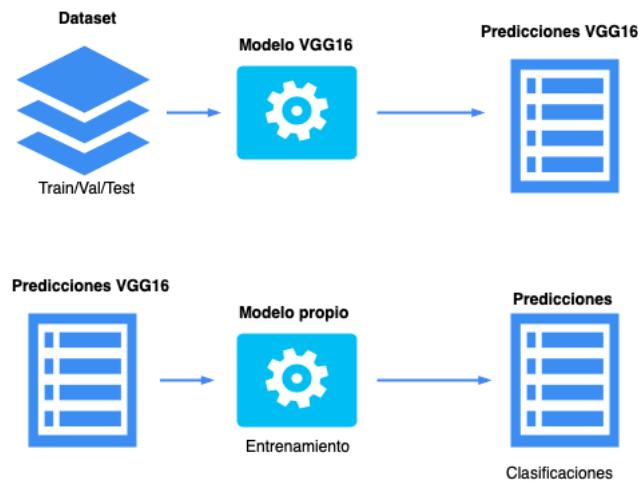
**Figura 38.** Otra representación de la arquitectura de VGG16 [7].

Este modelo es muy usado para crear clasificadores de imágenes, por lo que se ha decidido tomar como base del nuestro.

Layer (type)	Output Shape	Param #
<hr/>		
input_2 (InputLayer)	(None, None, None, 3)	0
block1_conv1 (Conv2D)	(None, None, None, 64)	1792
block1_conv2 (Conv2D)	(None, None, None, 64)	36928
block1_pool (MaxPooling2D)	(None, None, None, 64)	0
block2_conv1 (Conv2D)	(None, None, None, 128)	73856
block2_conv2 (Conv2D)	(None, None, None, 128)	147584
block2_pool (MaxPooling2D)	(None, None, None, 128)	0
block3_conv1 (Conv2D)	(None, None, None, 256)	295168
block3_conv2 (Conv2D)	(None, None, None, 256)	590080
block3_conv3 (Conv2D)	(None, None, None, 256)	590080
block3_pool (MaxPooling2D)	(None, None, None, 256)	0
block4_conv1 (Conv2D)	(None, None, None, 512)	1180160
block4_conv2 (Conv2D)	(None, None, None, 512)	2359808
block4_conv3 (Conv2D)	(None, None, None, 512)	2359808
block4_pool (MaxPooling2D)	(None, None, None, 512)	0
block5_conv1 (Conv2D)	(None, None, None, 512)	2359808
block5_conv2 (Conv2D)	(None, None, None, 512)	2359808
block5_conv3 (Conv2D)	(None, None, None, 512)	2359808
block5_pool (MaxPooling2D)	(None, None, None, 512)	0
<hr/>		
Total params:	14,714,688	
Trainable params:	14,714,688	
Non-trainable params:	0	

**Figura 39.** Resumen impreso del modelo VGG16 en Python usando el método `summary()`.

Sobre este modelo se construirá el clasificador. Más adelante se verá la implementación, pero para entender la arquitectura se va a ilustrar de forma sencilla para que se pueda entender. VGG16 actuará como base, y encima de este estará nuestro modelo.



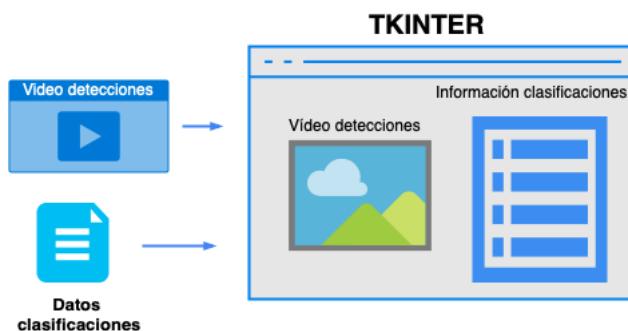
**Figura 40.** Arquitectura del clasificador de especies.

### 2.5.1.4. Procesamiento de la información para su presentación

Cuando ya se han obtenido las detecciones, las cuales posteriormente han sido clasificadas, solo queda representar toda esta información. En este bloque, los componentes necesarios han sido:

- Tkinter: librería para creación de interfaces gráficas
- OpenCV

Existen dos arquitecturas para dos tipos de salidas: una ventana que contiene el vídeo con las detecciones y al lado información sobre las clasificaciones; y un vídeo con las detecciones y información de las clasificaciones mostradas sobre este.



**Figura 41.** Esquema de la representación del análisis a través de una ventana con información.



**Figura 42.** Esquema de la presentación del vídeo procesado con información.

### 2.5.2. METODOLOGÍA SOFTWARE

Para el desarrollo de la aplicación se ha decidido usar SCRUM. Esta decisión viene dada a que SCRUM es una metodología ágil para gestión de proyectos, la cuál cada vez está siendo más famosa. Es una metodología que viene bien utilizar cuando el grado de incertidumbre en un proyecto es alto.

De esta forma, lo que se ha hecho es dividir el proyecto en diversos sprints a los cuales se les ha ido asignando diferentes tareas. Al ser yo el único participante del proyecto, todas las tareas asignadas en cada sprint recaen en mi.

A la hora de planificar un sprint (Sprint Planning) se ha seguido el siguiente proceso:

- **QUE:** de una lista de requisitos con prioridades, se seleccionan los objetivos que se quieren cumplir en el sprint.
- **COMO:** se elabora la táctica para cumplir los objetivos propuestos. Se definen las tareas necesarias para completar el objetivo basándonos en los requisitos elaborados con anterioridad. Con todas las tareas definidas hay que estimar el esfuerzo de cada una de ellas.

A continuación, vamos a ver el Product Backlog, que es una lista con todos los objetivos/tareas que tiene que ser hechos en el proyecto. Cada objetivo tendrá una estimación del esfuerzo que conlleva completar cada uno de los objetivos, además de tener una prioridad asignada. Para la

asignación de la estimación del esfuerzo (puntos de historia) se empleará la sucesión de Fibonacci: 1, 2, 3, 5, 8, 13, 21. Cada número se podría corresponder con los siguientes términos: pequeño, muy pequeño, pequeño, mediano, grande, muy grande y inmenso. Las prioridades estarán establecidas de forma numérica, siendo 1 la prioridad más alta.

**Tabla 6.**

Tareas que realizar durante el proyecto.

ID	Tarea	Requisito	Resumen	Puntos de historia	Prioridad	Dependencia (ID)
01	Recolección imágenes peces	RS02 RS03	-Obtención de todo tipo de imágenes de cualquier pez -Obtención de imágenes para las especies definidas	13	1	-
02	Preparación de imágenes	RS02 RS03	-Etiquetado de imágenes obtenidas de todos los peces -Recortado de imágenes de las especies definidas	8	1	1
03	Ampliar colección de datos	RS02 RS03	-Aumentar número de imágenes de todas las especies -Aumentar número de imágenes de las especies definidas	5	2	2
04	Diseño de un suministrador de vídeo	RS01	-Crear una clase que suministre el vídeo de entrada como fotogramas individuales -Implementar más funciones relacionadas con la modificación de parámetros del suministrador	2	1	-
05	Creación modelo de detección de peces	RS02	-Configurar modelo -Entrenar modelo -Probar y mejorar resultados	8	1	3

06	Creación modelo de clasificación de peces	RS03	-Crear y configurar modelo -Entrenar modelo -Probar y mejorar resultados	5	1	3
07	Puesta en marcha modelo de detección	RS02	-Creación de script para uso del modelo en vídeo -Crear clase para encapsular todas las operaciones	3	2	5
08	Puesta en marcha modelo de clasificación	RS03	-Creación de una clase para encapsular todas las operaciones	3	2	6
09	Puesta en funcionamiento de ambos modelos en conjunto	RS02 RS03	-Establecer un proceso para que ambos modelos cooperen y funcionen conjuntamente.	5	2	7, 8
10	Diseño de la salida del sistema en una ventana con información	RS04 RS05 RS06 RS07	-Diseñar ventana para mostrar la información de las detecciones y clasificaciones. -Conectar la salida de ambos modelos con la ventana diseñada -La velocidad del vídeo tiene que ser la misma que la del vídeo original	8	3	9
11	Diseño de la salida del vídeo con información implícita	RS04 RS05 RS06 RS07	-Mostrar sobre el vídeo la información obtenida del clasificador -La velocidad del vídeo tiene que ser la misma que la del vídeo original	3	3	9

12	Documentar el sistema		-Describir marco práctico -Describir marco teórico -Creación de manual de usuario -Realizar la introducción y la bibliografía	5	4	Todos
----	-----------------------	--	--	---	---	-------

**Tarea 01**

Recolección imágenes peces

**Descripción:**

Esta tarea esta dividida en dos partes: recolección de fotografías de peces de cualquier especie, y recolección de fotografías de una serie de especies definidas.

Para la primera parte, lo que hay que hacer es recolectar todo tipo de imágenes en las que haya al menos un pez, el cuál se pueda distinguir sin mucho problema. Para ello, se empleará una lista con diferentes familias y especies, de la cuál podemos obtener una gran variedad de fotografías. Se usará Google como herramienta de búsqueda. En la Tabla 7 se puede ver la lista con nombres de familias, géneros y especies de peces empleados para buscar fotografías.

**Tabla 7.**

Familias y especies de peces.

Sardinella Maderensis	Sardinella Aurita	Pomatomus Saltator
Spicara Maena	Chelon Labrosus	Liza Aurata
Trachurus Trachurus	Trachurus Mediterraneanus	Trachorus Ovatus
Lichia Amia	Seriola Dumerili	Dicentarchus Labrax
Sphyraena Sphyraena	Sphyraena Viridensis	Oblada Melanura
Diplodus Sargus	Diplodus Puntazo	Diplodus Vulgaris
Diplodus Annularis	Diplodus Cervinus	Pagellus Erythrimus
Pagrus Pagrus	Pargus Auriga	Sparus Aurata
Spundylisoma Cantharus	Dentex Dentex	Lithognathus Mormyrus
Sarpa Sarpa	Boops Boops	Pomadasys Incisus
Labrus Merula	Labrus Viridis	Labrus Bimaculatus
Syphodus Tinca	Syphodus Roissali	Syphodus Melops
Syphodus Ocellatus	Syphodus Cinereus	Syphodus Mediterraneanus
Syphodus Rostratus	Thalassoma Pavo	Corus Julis
Xyrichtys Novacula	Chromis Chromis	Anthias Anthias
Serranus Hepatus	Serranus Scriba	Serranus Cabrilla
Epinephelus Marginatus	Epinephelus Costae	Apagon Imberbis
Sciaena Umbra	Umbrina Cirrosa	Phycis Phycis
Capros Aper	Zeus Faber	Balistes Carolinensis
Parablennius Sanguinolentus	Parablennius Pilicornis	Coryphoblennius Galerita
Lypophrys Canevali	Lypophrys Nigriceps	Lypophrys Pavo
Gobius Buchichii	Pomatuschistus Minutus	Mullus Surmuletus
Mullus Barbatus	Trigla Lucerna	Trachinus Draco
Trachinus Radiatus	Tripterygion Delaisi	Tripterygion Tripteronotus
Acanthuridae Acanthurus	Acanthuridae Ctenochaetus	Acanthuridae naso
Acanthuridae Paracanthurus	Acanthuridae Prionus	Acanthuridae Zebrasoma

La idea es obtener el mayor número de imágenes posibles para poder garantizar unos mejores resultados en el futuro detector de peces. No importa si todas estas imágenes están mezcladas en una carpeta.

Por otro lado, hay que recolectar imágenes de una serie de especies de peces definidas, las cuales se encuentran en un entorno cerrado. Estas especies están descritas en los requerimientos y son las siguientes:

- Acanthurus Sohal
- Amphirion Ocellaris
- Coris Formosa
- Halichoeres Adustus
- Naso Elegans
- Naso Vlamingii
- Paracanthurus Hepatus
- Pseudanthias Squamipinnis
- Pterapogon Kauderni
- Siganus Vulpinus
- Sphaeramia Nematoptera
- Zebrasoma Flavescens
- Zebrasoma Veliferum

Para esta parte es importante separar las fotografías en diferentes carpetas para su posterior uso, y al igual que en la primera parte, cuantas más imágenes obtengamos mejores serán los resultados del futuro clasificador.

## Tarea 2

Preparación de imágenes

**Descripción:**

Al igual que para la tarea anterior, esta tarea está dividida en dos partes: una para las imágenes obtenidas de todas las especies y otra para las imágenes de las especies definidas.

En la primera parte hay que etiquetar todas las imágenes obtenidas usando la herramienta de etiquetado [38] diseñada para crear datasets para Darknet. Todo pez que aparezca en las imágenes tiene que ser marcado usando esta herramienta. Dicho etiquetado se realiza encuadrando cada pez dentro de un rectángulo de tal forma que contenga todo el cuerpo del pez. Cada imagen que se etiquete generará un archivo de texto que contiene todas las coordenadas de los peces etiquetados, las cuales se usarán mas adelante.



**Figura 43.** Ejemplo de como tiene que ser el etiquetado de las imágenes.

Pasando a la segunda parte, lo que hay que hacer es recortar las imágenes de tal forma que cada pez contenido en la imagen queda bien centrado y destaque. A continuación, se mostrarán unos ejemplos de como tiene que ser el resultado de esta operación.



**Figura 44.** Ejemplo de como tendría que quedar recortada una imagen de tal forma que destaque el pez.



**Figura 45.** Otro ejemplo de recortado empleando otra especie.

### Tarea 3

Ampliar colección de datos

#### Descripción:

Una vez que se haya recolectado y tratado las imágenes correspondientes a los dos apartados anteriores, hay que ampliar los datasets mediante técnicas de “data augmentation”. Para las imágenes con todo tipo de peces hay que tener en cuenta que cualquier modificación que se realice sobre la imagen real, hay que aplicarla también a las coordenadas de su fichero de texto correspondiente. Transformaciones como rotaciones, cambios de tono o distorsiones pueden ser aplicados sobre las imágenes.

### Tarea 4

Diseño suministrador de vídeo

#### Descripción:

Creación de un suministrador de vídeo, al cuál se le pase un archivo de vídeo y sea capaz de realizar operaciones básicas sobre este, destacando como importantes las siguientes:

- Obtención de fotogramas del vídeo de forma ordenada
- Cambiar la resolución del vídeo
- Obtención de los fotogramas totales que componen el vídeo

- Obtención de los fotogramas por segundo
- Calculo de un retardo para regular la velocidad de reproducción del vídeo

**Tarea 5**

Creación del modelo de detección de peces

**Descripción:**

A partir del dataset de imágenes y su correspondiente etiquetado, hay que desarrollar el modelo de detección. Para ello hay que usar Darknet, framework que contiene el algoritmo de detección Yolo.

**Tarea 6**

Creación del modelo de clasificación de peces

**Descripción:**

Empleando las imágenes recolectadas y tratadas para las especies definidas, hay que diseñar una red neuronal convolucional capaz de clasificar estas especies. Para ello hay que emplear Keras, y diseñar un modelo sobre el modelo VGG16 como base.

**Tarea 7**

Puesta en marcha modelo de detección

**Descripción:**

Una vez se tenga listo el modelo de detección, desarrollar una clase para el detector que contenga todas las operaciones necesarias para poder usar el modelo.

**Tarea 8**

Puesta en marcha modelo de clasificación

**Descripción:**

Al igual que para el modelo de detección, desarrollar una clase que contenga todas las operaciones necesarias para realizar las clasificaciones.

**Tarea 9**

Puesta en funcionamiento de ambos modelos en conjunto

**Descripción:**

Con los dos modelos totalmente funcionales, conectarlos de tal forma que primeramente se le pase una imagen al modelo detector, este realice las detecciones correspondientes, y estas detecciones sean pasadas al clasificador para su proceso.

**Tarea 10**

Diseño de la salida del sistema en una ventana con información

**Descripción:**

Diseñar una ventana que muestre el vídeo con las detecciones en un lado, y en el otro información relativa a las clasificaciones.

Una vez diseñada la ventana, conectar los dos modelos para ponerlo en funcionamiento todo en conjunto. Las clasificaciones se harán cada intervalo de tiempo definido por el usuario. Para ello, una vez transcurrido ese intervalo, se cogen todas las detecciones de ese instante y se clasifican, mostrando la información en la ventana correspondiente.

**Tarea 11**

Diseño de la salida del vídeo con información implícita

**Descripción:**

Diseñar el procesado del vídeo de tal forma que las detecciones como las clasificaciones se muestren sobre el vídeo. Habrá que crear un proceso en el que para cada fotograma se obtengan las detecciones, las cuales se pasan para clasificar, y una vez clasificados se marca la detección y se pone la información correspondiente a cada detección.

**Tarea 12**

Documentar el sistema

**Descripción:**

Al finalizar el desarrollo de la aplicación, redactar toda la documentación referida al proyecto. Describir los marcos práctico y teórico, manual de usuario, introducción con sus correspondientes apartados y la bibliografía.

**Planificación**

Con los objetivos bien definidos, se ha realizado una serie de sprints, a los cuáles se les ha asignado unos tiempos dependiendo del esfuerzo y el número de tareas asignadas. Para ver que tareas y tiempos fueron asignadas a cada sprint, se ha hecho una tabla con toda la información.

**Tabla 8.**

Planificación sprints.

Sprint	Tareas asignadas (ID)	Duración (semanas)
01	1, 2	6
02	3, 4	2
03	5, 7	3
04	6, 8	2
05	9, 11	2
06	10	2
07	12	4

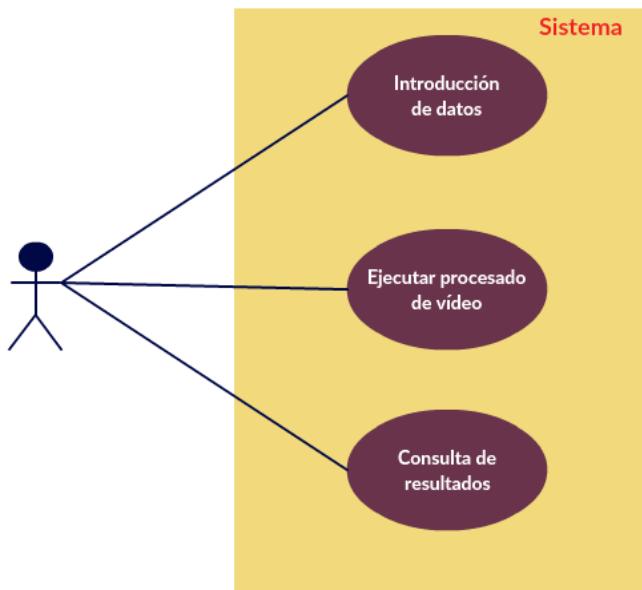
### 2.5.3. CASOS DE USO

Esta aplicación es muy sencilla en cuanto a interacción con el usuario nos referimos. A continuación, vamos a ver el caso de uso que tiene y su diagrama.

**Tabla 9.**

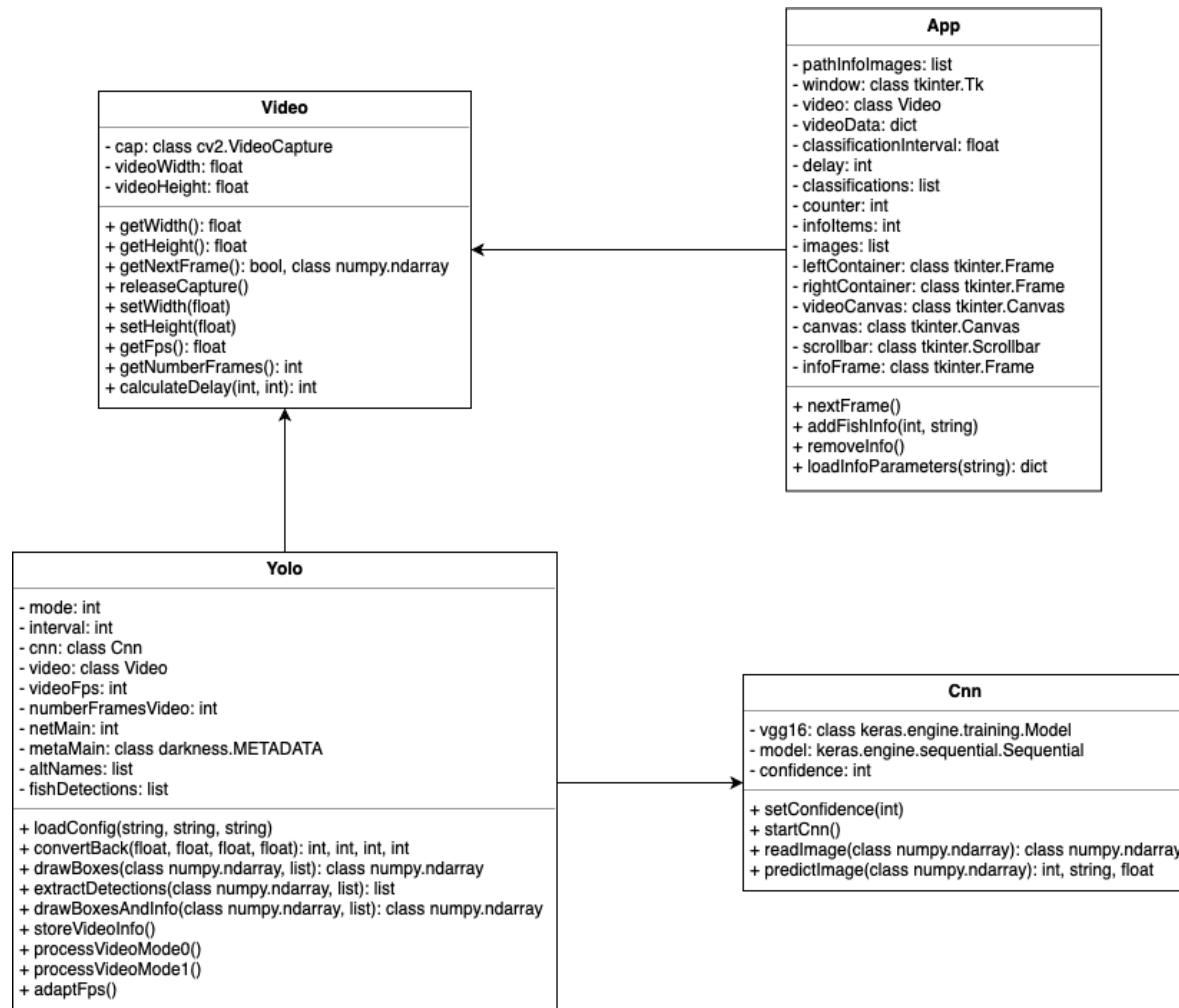
Caso de uso de la aplicación.

Procesar vídeo		
Precondición	El usuario tiene que tener todas las librerías y herramientas instaladas	
Descripción	El usuario introduce un vídeo para ser procesado	
Secuencia normal	Paso	Acción
	1	El usuario introduce el vídeo al sistema
	2	El sistema solicita al usuario seleccionar un umbral de confianza para las clasificaciones
	3	El usuario selecciona el umbral de confianza para las clasificaciones
	4	El sistema pide seleccionar modo de mostrar la salida del procesado
	5	El usuario selecciona el modo
	6	El sistema procesa el vídeo
	7	El sistema devuelve el vídeo procesado
Postcondición	Para mostrar el vídeo con la ventana con información, el usuario tiene que ejecutar otro script, o ver el vídeo con información obtenida	

**Figura 46.** Diagrama del caso de uso de procesado de vídeo.

#### 2.5.4. DIAGRAMA DE CLASES

A continuación, vamos a ver el diagrama con las clases de la aplicación, donde se explicará para cada clase qué hace cada método disponible.



**Figura 47.** Diagrama de clases de la aplicación.

### Clase Video

- **getWidth()**: devuelve la anchura de la captura de vídeo.
- **getHeight()**: devuelve la altura de la captura de vídeo
- **getNextFrame()**: devuelve el siguiente frame de una captura de vídeo, además de un booleano que indica si la captura ha terminado.
- **releaseCapture()**: cierra la captura del vídeo.
- **setWidth(float)**: ajusta la anchura de la captura de vídeo.
- **setHeight(float)**: ajusta la altura de la captura de vídeo.
- **getFps()**: devuelve el número de fotogramas de la captura de vídeo.
- **getNumberFrames()**: devuelve el número total de fotogramas de los que se compone la captura de vídeo.
- **calculateDelay(int, int)**: calcula el retraso en milisegundos que hay que aplicarle a una captura de vídeo. Esto se hace proporcionando dos valores al método: FPS actuales y FPS deseados.

### Clase Yolo

- **loadConfig(string, string, string)**: carga la configuración del modelo de detección a partir de las rutas de los ficheros necesarios, como son el archivo de configuración, los pesos y los metadatos.
- **convertBack(float, float, float, float)**: convierte las coordenadas que están formato (xcentro, ycentro, anchura, altura) a (xmin, ymin, xmax, ymax).

- **drawBoxes(class numpy.ndarray, list)**: dibuja las cajas alrededores de las detecciones a partir de una imagen y una lista con las coordenadas de las detecciones.
- **extractDetections(class numpy.ndarray, list)**: a partir de un fotograma y una lista de detecciones, realiza una clasificación de todas estas detecciones y las devuelve en una lista.
- **drawBoxesAndInfo(class numpy.ndarray, list)**: dibuja las detecciones sobre un fotograma con información de las clasificaciones.
- **storeVideoInfo()**: almacena en un fichero JSON información relativa a las clasificaciones, en el caso de que se muestre el vídeo con información en una ventana.
- **processVideoMode0()**: método para realizar el procesado del vídeo de tal forma que se muestren las detecciones como información relativa sobre las clasificaciones sobre el mismo vídeo.
- **processVideoMode1()**: método para procesar el vídeo de tal forma que posteriormente se pueda mostrar en una ventana diseñada para ello. Este método generara un vídeo con las detecciones y un archivo JSON con información del vídeo y de las clasificaciones.
- **adaptFps()**: método para ajustar la velocidad del vídeo procesado a la velocidad del vídeo original.

#### Clase Cnn

- **setConfidence(int)**: método para ajustar el umbral de aceptación para las clasificaciones. Este umbral puede ir desde 0 a 100.
- **startCnn()**: método para cargar la configuración y los pesos del modelo de clasificación.
- **readImage(class numpy.ndarray)**: método que sirve para transformar la imagen que es pasada como array de la librería numpy al formato adecuado para su entrada al modelo.
- **predictImage(class numpy.ndarray)**: a través de este método se realiza la clasificación de la imagen pasada. Si la clasificación con mayor puntuación supera el umbral establecido entonces devuelve el ID de la clasificación, el nombre de la especie y la puntuación (de 0 a 100). De lo contrario, si no se supera el umbral, entonces se devuelven valores nulos.

#### Clase App

- **nextFrame()**: es el método principal de la clase. Se encarga de mostrar el vídeo en la ventana con la información de las clasificaciones.
- **addFishInfo(int, string)**: añade un pez con su información a la ventana que muestra la información correspondiente.
- **removeInfo()**: elimina toda la información mostrada en la ventana de la información.
- **loadInfoParameters(string)**: método empleado para leer los datos de un archivo JSON y devolverlos en forma de diccionario.

## 2.6. IMPLEMENTACIÓN Y FUNCIONAMIENTO DEL SISTEMA

Para describir como se ha desarrollado la aplicación en detalle, haremos uso de la planificación de los sprints. De esta se podrá ver el proceso de forma ordenada.

### 2.6.1. SPRINT 1

Tareas: Recolección y preparación de datos

En esta primera parte se han recolectado alrededor de 9000 imágenes de todo tipo de peces para entrenar el algoritmo de detección, y unas 2600 para el entrenamiento del modelo de clasificación (unas 200 imágenes por especie). Con todas ellas se ha hecho el hash de cada una para compararlas con el resto, para así detectar si hay imágenes repetidas y eliminarlas, ya que durante el proceso de búsqueda es muy probable que nos encontremos fotos repetidas y no nos demos cuenta, y más aun si recolectamos esta gran cantidad.

Con las imágenes listas, se ha pasado a etiquetar las imágenes del dataset todas las especies. Para ello se ha hecho uso de la herramienta de etiquetado mencionada con anterioridad [38]. Al marcar cada uno de los peces que aparecen en las imágenes, se genera un archivo de texto que contiene las coordenadas y clase del objeto a detectar. El formato es el siguiente:

```
<clase-objeto> <x> <y> <ancho-bounding-box> <altura-bounding-box>
```

**<clase-objeto>**: numero entero que indica a la clase que pertenece el objeto marcado (empieza por 0 hasta clases-1).

**<x>, <y>, <ancho-bounding-box>, <altura-bounding-box>**: valores float, comprendidos entre 0.0 y 1.0.

**<x>**: centro eje X de la bounding box ( $x = ((X_{\max} + X_{\min})/2) / \text{ancho\_imagen}$ )

**<y>**: centro eje Y de la bounding box ( $y = ((Y_{\max} + Y_{\min})/2) / \text{altura\_imagen}$ )

**<ancho-bounding-box>**: ancho de la bounding box del objeto en la imagen ( $\text{ancho-bounding-box} = (X_{\max} - X_{\min}) / \text{ancho\_imagen}$ )

**<altura-bounding-box>**: altura de la bounding box del objeto en la imagen ( $\text{altura-bounding-box} = (Y_{\max} - Y_{\min}) / \text{altura\_imagen}$ )

Para el dataset de las imágenes de especies definidas, lo que se ha hecho es recortarlas de tal forma que el pez que se quiere clasificar destaque en la imagen.



**Figura 48.** A la izquierda la imagen original y a la derecha la imagen recortada.

## 2.6.2. SPRINT 2

Con todas las imágenes listas, lo que se ha decidido hacer es ampliar la colección de imágenes a partir de las que ya se tiene. Se han seguido dos caminos diferentes para ampliar los dos datasets que tenemos.

Hay que tener en cuenta que para el primer dataset de imágenes, cualquier modificación que afecte a la posición de los peces se tendrá que ver modificada acordeamente en las coordenadas de los archivos de texto. Es por ello que solo se harán dos modificaciones básicas: rotar la imagen 180° y voltear (como si pusiésemos la imagen frente un espejo, de forma que apunta al otro lado). Para llevar estas modificaciones a las coordenadas de las cajas (“bounding boxes”) es muy sencillo, ya que la altura y anchura de cada una de estas sigue siendo la misma, lo único que cambia es la coordenada del centro de cada una de estas. En el caso de que la imagen se rote, el centro nuevo se calcula de la siguiente forma para cada eje:

$$\begin{aligned}x_{\text{nueva}} &= 1 - x_{\text{original}} \\y_{\text{nueva}} &= 1 - y_{\text{original}}\end{aligned}$$

Para el caso de que se voltee una imagen, tan solo hay que recalcular el eje x, ya que los objetos siguen a la misma altura. Por lo tanto, las nuevas coordenadas al voltear la imagen quedarían de la siguiente manera:

$$\begin{aligned}x_{\text{nueva}} &= 1 - x_{\text{original}} \\y_{\text{nueva}} &= y_{\text{original}}\end{aligned}$$

A través de estas dos modificaciones, se consigue triplicar el dataset original, obteniendo alrededor de 26000 imágenes. Todas estas modificaciones se han realizado con la librería PIL.

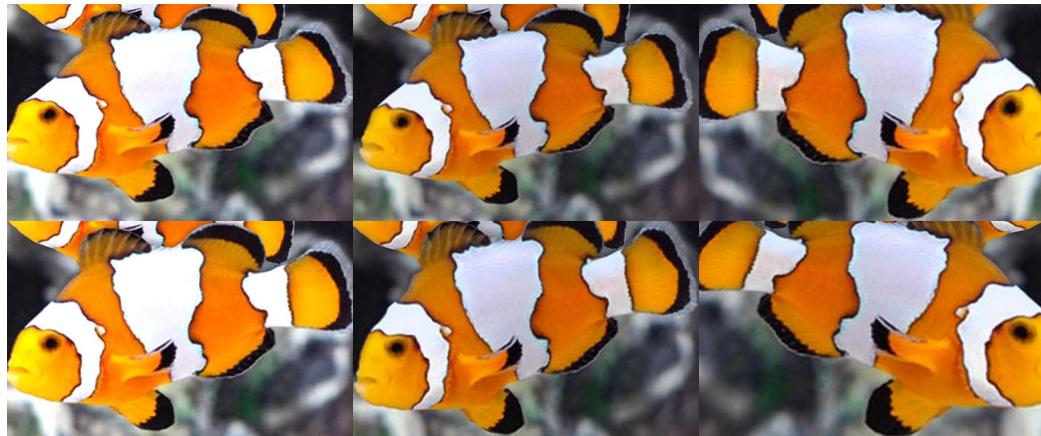


**Figura 49.** Ejemplo de como quedarían las imágenes tras las modificaciones. A la izquierda la imagen original, en el centro la imagen volteada y a la derecha la imagen rotada 180°.

A la hora de ampliar el segundo dataset, es posible realizar muchas más modificaciones, ya que no se depende de ninguna coordenadas. Para ello, se ha usado la librería Augmentor, la cuál nos permite crear un “pipeline” con diversas operaciones sobre las imágenes. Las operaciones empleadas son las siguientes:

- Distorsión: la imagen se divide en celdas (16x16) y se le aplica una ligera distorsión a cada una de ellas.
- Recortado: se recorta la imagen, pero solo una pequeña parte, es decir, nos quedamos con el 95% de la imagen.
- Volteo: se da la vuelta a la imagen de tal forma como si se viene reflejada en un espejo.
- Cambio de brillo: se cambia el brillo de la imagen en un rango de 0.5 a 1.3, donde 1.0 es el brillo original.
- Rotación: se aplica una rotación de 5° a la izquierda o derecha.
- Torcedura: la imagen se coge como un plano que se apoya sobre el centro, y se mueve hacia cualquier lado con cierta magnitud.

A cada una de estas operaciones se les ha asignado un 50% de probabilidades de que se apliquen sobre la imagen, de esta manera logramos obtener una gran cantidad de imágenes diferentes, con poca probabilidad de que se repitan. Se han generado 1000 imágenes por clase.



**Figura 50.** Diversas imágenes generadas a partir de la misma imagen original.

Además de aumentar nuestros datasets, se ha diseñado el suministrador de vídeo, que será el punto de entrada al sistema del vídeo a procesar. Se ha usado la librería OpenCV en Python para esta tarea. Básicamente se han implementado las operaciones básicas que nos ofrece OpenCV en una clase, para así poder operar sencillamente sobre el vídeo.

### 2.6.3. SPRINT 3

Lo primero que hacemos es bajarnos el repositorio de GitHub de Darknet modificado por AlexeyAB [39], el cuál ofrece más funcionalidades que el repositorio original de Joseph Chet Redmon. Compilamos el proyecto y estará listo para trabajar con él. En el manual de usuario se explica como instalar lo necesario, y las opciones de compilación disponibles. También es necesario bajarnos los pesos iniciales *darknet53.conv.74* que se indican en el repositorio.

A la hora de implementar el algoritmo de detección, lo primero que tenemos que hacer es preparar las imágenes para que podamos alimentarlas al algoritmo para su entrenamiento. Por ello, en la carpeta ‘*data/*’ creamos una carpeta llamada ‘*obs/*’. En esta carpeta se ha metido una carpeta con todas las imágenes y etiquetas. Lo siguiente es dividir los datos en un set de entrenamiento y otro de test. Para ello se ha implementado un script que coge todas las imágenes y etiquetas, y lo divide en dos directorios, uno para el entrenamiento (con el 85% de las imágenes) y otro para el test (con el 15% restante). Con los datos divididos, hay que crear dos archivos de texto en el directorio ‘*data/*’ con las rutas absolutas hacia cada una de las imágenes de entrenamiento y test. Así tendremos dos archivos de texto (*train\_images.txt* y *test\_images.txt*) con la ruta hacia todas las imágenes.

Lo siguiente es crear un archivo llamado ‘*fish.data*’ en el directorio ‘*data/*’, donde irá la información relativa a la localización de los archivos de texto creados, la dirección con el archivo que contiene los nombres de las clases a detectar, y la dirección de guardado de los pesos del modelo.

```

1  classes = 1
2  train = data/train_images.txt
3  valid = data/test_images.txt
4  names = data/fish.names
5  backup = backup/

```

**Figura 51.** Configuración del archivo ‘*fish.data*’.

Se crea otro archivo llamado ‘fish.names’, el cuál contendrá los nombres de las clases a detectar. Como solo se van a detectar peces, la única clase que va a figurar en este archivo es el nombre “fish”.

Solo queda configurar el modelo para su entrenamiento. Para ello cogemos el archivo *yolov3.cfg* como punto de partida. Lo primero que tenemos que cambiar es todos los campos de clases (*clases*) a 1, ya que solo vamos a detectar peces. Lo siguiente es que tenemos que cambiar son los filtros de las capas convolucionales que vienen seguidas de una capa yolo (que son 3). El valor de los filtros tiene que ser (*clases* + 5) \* 3, en nuestro caso 18. También se pueden cambiar las anchors (anclas), aunque no es obligatorio. Estas anchors sirven para enmarcar a los objetos predichos por el modelo, de forma que el modelo no tendrá que predecir el tamaño final del objeto, si no que tendrá que sacar la anchor más cercana a cada objeto y ajustarlas. Para calcular las anchors para nuestro dataset, usamos el siguiente comando:

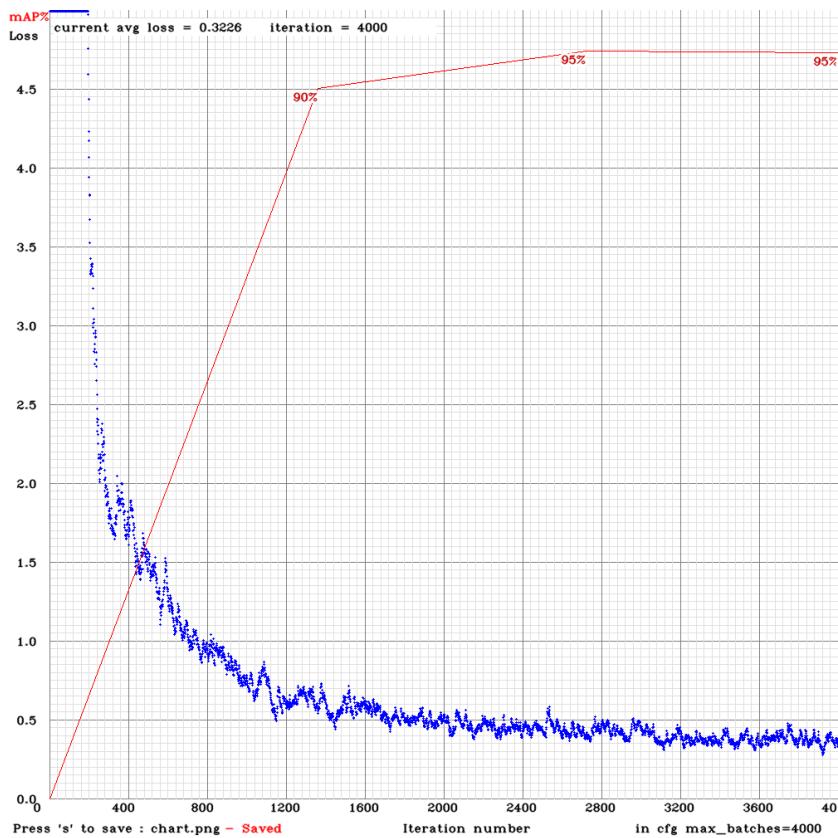
```
./darknet detector calc_anchors data/fish.data -num_of_clusters 9 -width 512 -height 512
```

El resultado lo adjuntamos en los tres campos del archivo *yolov3.cfg*. Solo queda por ajustar la resolución para el entrenamiento, cuyas dimensiones tienen que ser iguales (la anchura igual que la altura). Ajustamos esta a 512x512, el batch lo dejamos en 64, y en este caso ponemos las subdivisiones a 32, o de lo contrario nos quedaremos sin memoria en la GPU.

Con todo listo, ya solo queda poner el modelo a entrenar. El número de iteraciones recomendadas es de 2000 por clase. El algoritmo esta configurado para que guarde los pesos obtenidos cada 1000 iteraciones, pero se ha modificado para que sea cada 200, y así probar que pesos dan mejores resultados y evitar el sobre entrenamiento. El comando para poner comenzar el entrenamiento es el siguiente:

```
./darknet detector train data/fish.data cfg/yolov3.cfg darknet53.conv.74 -map
```

Usamos la opción *-map* para que se vaya mostrando la media de precisión media, la cuál se calcula cada 4 épocas.



**Figura 52.** Gráfica con la evolución del entrenamiento.

A partir de las 2000 iteraciones la precisión apenas aumenta y el average loss no desciende, por lo que hay que tener cuidado de que no se sobreentrene el modelo. Probando los diferentes pesos obtenidos en un rango de 1600-2000, se ha comprobado que los mejores resultados ofrecen son los pesos de 1800 iteraciones.

Se ha comprobado la media de la precisión media para diferentes porcentajes de intersección sobre la unión. La intersección sobre la unión (IoU) es básicamente el área que la predicción y el objeto comparten dividido entre el área de unión de ambas áreas. Los porcentajes de IoU empleados han sido 0.3, 0.4, 0.5, 0.6, 0.7 y 0.8. Estos han sido los resultados:

#### ##### IOU 0.3 #####

```
detections_count = 9244, unique_truth_count = 5256
class_id = 0, name = fish, ap = 93.96%          (TP = 3941, FP = 64)
for thresh = 0.25, precision = 0.98, recall = 0.75, F1-score = 0.85
for thresh = 0.25, TP = 3941, FP = 64, FN = 1315, average IoU = 78.33 %
IoU threshold = 30 %, used Area-Under-Curve for each unique Recall
mean average precision (mAP@0.30) = 0.939623, or 93.96 %
Total Detection Time: 91.000000 Seconds
```

#### ##### IOU 0.4 #####

```
detections_count = 9244, unique_truth_count = 5256
class_id = 0, name = fish, ap = 93.19%          (TP = 3933, FP = 72)
for thresh = 0.25, precision = 0.98, recall = 0.75, F1-score = 0.85
for thresh = 0.25, TP = 3933, FP = 72, FN = 1323, average IoU = 78.27 %
IoU threshold = 40 %, used Area-Under-Curve for each unique Recall
mean average precision (mAP@0.40) = 0.931907, or 93.19 %
Total Detection Time: 91.000000 Seconds
```

**#### IOU 0.5 ####**

*detections\_count = 9244, unique\_truth\_count = 5256  
 class\_id = 0, name = fish, ap = 91.35% (TP = 3913, FP = 92)  
 for thresh = 0.25, precision = 0.98, recall = 0.74, F1-score = 0.85  
 for thresh = 0.25, TP = 3913, FP = 92, FN = 1343, average IoU = 78.06 %  
 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall  
 mean average precision (mAP@0.50) = 0.913536, or 91.35 %  
 Total Detection Time: 91.000000 Seconds*

**#### IOU 0.6 ####**

*detections\_count = 9244, unique\_truth\_count = 5256  
 class\_id = 0, name = fish, ap = 86.18% (TP = 3850, FP = 155)  
 for thresh = 0.25, precision = 0.96, recall = 0.73, F1-score = 0.83  
 for thresh = 0.25, TP = 3850, FP = 155, FN = 1406, average IoU = 77.17 %  
 IoU threshold = 60 %, used Area-Under-Curve for each unique Recall  
 mean average precision (mAP@0.60) = 0.861770, or 86.18 %  
 Total Detection Time: 90.000000 Seconds*

**#### IOU 0.7 ####**

*detections\_count = 9244, unique\_truth\_count = 5256  
 class\_id = 0, name = fish, ap = 66.44% (TP = 3426, FP = 579)  
 for thresh = 0.25, precision = 0.86, recall = 0.65, F1-score = 0.74  
 for thresh = 0.25, TP = 3426, FP = 579, FN = 1830, average IoU = 70.16 %  
 IoU threshold = 70 %, used Area-Under-Curve for each unique Recall  
 mean average precision (mAP@0.70) = 0.664367, or 66.44 %  
 Total Detection Time: 90.000000 Seconds*

**#### IOU 0.8 ####**

*detections\_count = 9244, unique\_truth\_count = 5256  
 class\_id = 0, name = fish, ap = 25.92% (TP = 2076, FP = 1929)  
 for thresh = 0.25, precision = 0.52, recall = 0.39, F1-score = 0.45  
 for thresh = 0.25, TP = 2076, FP = 1929, FN = 3180, average IoU = 44.64 %  
 IoU threshold = 80 %, used Area-Under-Curve for each unique Recall  
 mean average precision (mAP@0.80) = 0.259240, or 25.92 %  
 Total Detection Time: 90.000000 Seconds*

Como es obvio, cuanto mayor sea el porcentaje de IoU menor será la precisión, ya que es más difícil que todo el área de detección coincida con el objeto real. Pero en general los resultados son bastante buenos.

Con los pesos listos, se ha diseñado una clase en Python para poder operar con el algoritmo. Para ello se usa la librería de enlace dinámica con un wrapper escrito en Python. Esta clase se encarga de cargar la configuración del modelo, realizar las detecciones y pintarlas sobre la imagen. Tiene dos modos de procesado: uno que pinta toda la información sobre el vídeo, y otro que pinta las detecciones sobre el vídeo, y las clasificaciones las guarda en un fichero para su posterior uso. De esto hablaremos más adelante.

## 2.6.4. SPRINT 4

Con el modelo de detección listo, toca crear el modelo de clasificación. Para esta parte se ha usado Spyder, un entorno de desarrollo diseñado para tareas de analítica en lo que concierne al campo de Data Science y Machine Learning. Como se ha comentado, se va a emplear el modelo VGG16 como modelo base para la creación del nuestro, de lo contrario necesitaríamos una gran cantidad de fotografías y gran potencia de computación. Antes de nada, se ha dividido el dataset original en tres datasets: train (70%), validate (15%) y test (15%). Esta división se hace de forma aleatoria con un script que se ha escrito.

Comenzamos por coger tanto los datasets de entrenamiento, test y validación, creamos un generador de datos que re-escalae las imágenes a arrays con valores comprendidos entre 0 y 1. Esto se hace así porque los modelos aceptan arrays con este formato. Mientras las imágenes se re-escalas, son pasadas al modelo VGG16 en batches para que vaya realizando las predicciones, las cuales guardaremos en un fichero .npy. Posteriormente, estos ficheros numpy serán cargados para usarlos en el entrenamiento del modelo que irá por encima de VGG16. También es necesario cargar las etiquetas de cada predicción que se ha hecho, para poder usarlo en el entrenamiento, validación y test. Esto se consigue cambiando el número de clase que pertenece a cada clase a un vector de categoría, el cuál tendrá tantos elementos como clases haya, y habrá un 1 en la posición correspondiente a su clase, siendo el resto 0.

Lo siguiente que queda es el diseño del modelo superior que se encargará de hacer nuestras clasificaciones. Para empezamos creando un modelo secuencial al que iremos añadiendo capas. La primera capa que se añade es Flatten, la cual básicamente toma la entrada y la transforma a un vector de una dimensión. Le sigue un par de capas Dense (usando la función de activación LeakyReLu), a las cuales se les añade Dropout para desactivar ciertas neuronas aleatoriamente durante un tiempo. De esta forma se evita sobreentrenar la red neuronal. Por último, tenemos otra capa Dense al final con 13 neuronas, que es la que nos dará las salidas con los resultados de las clasificaciones.

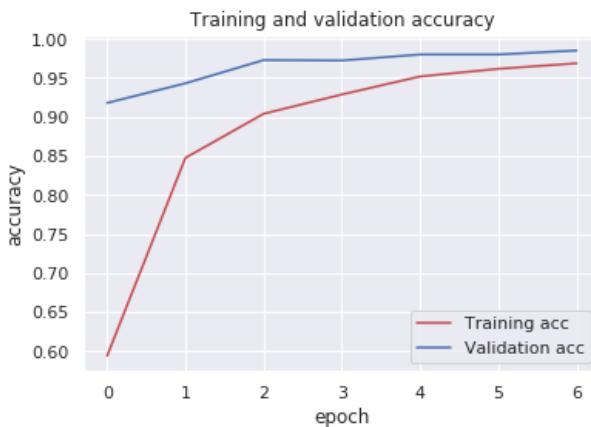
Layer (type)	Output Shape	Param #
<hr/>		
flatten_2 (Flatten)	(None, 25088)	0
dense_4 (Dense)	(None, 100)	2508900
dropout_3 (Dropout)	(None, 100)	0
dense_5 (Dense)	(None, 50)	5050
dropout_4 (Dropout)	(None, 50)	0
dense_6 (Dense)	(None, 13)	663
<hr/>		
Total params: 2,514,613		
Trainable params: 2,514,613		
Non-trainable params: 0		

**Figura 53.** Arquitectura del modelo impreso con el método summary()

Una vez tenemos lista la arquitectura de nuestro modelo, toca entrenarlo. Para ello, se ha configurado para que sean 7 épocas en batchs de 50. Este entrenamiento nos devuelve los siguientes resultados:

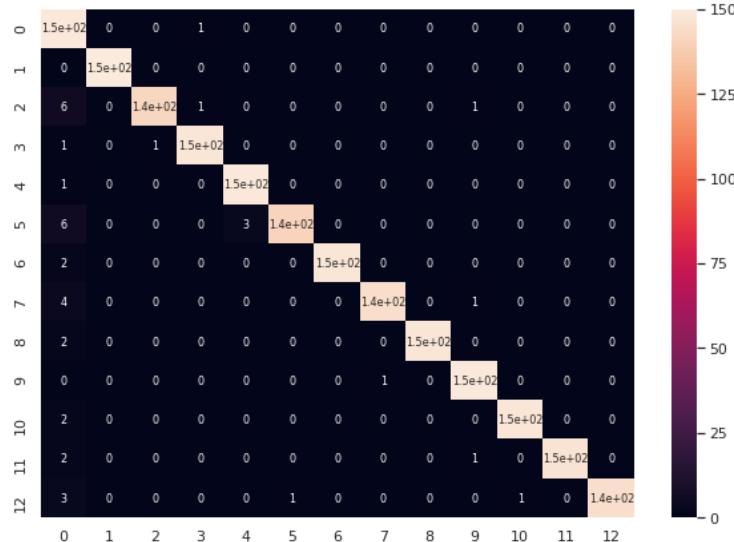
Accuracy: 98.51282101411087  
Loss: 0.0680003130617432D

Observando estos resultados parece ser que se han obtenido unos buenos resultados. Para confirmar esto tenemos que coger el dataset de test y probarlo sobre nuestro modelo. Pero antes de ello, vamos a dibujar una gráfica con la evolución del entrenamiento de nuestro modelo, para comprobar como han ido evolucionando la precisión del entrenamiento respecto a la de la validación.



**Figura 54.** Evolución del modelo durante el entrenamiento.

Como se puede apreciar en la gráfica, la precisión del entrenamiento ha ido en aumento, hasta terminar poniéndose casi a la par con el acierto de la validación. Ahora vamos a pasar a comprobar si realmente es bueno nuestro modelo con el dataset de test.



**Figura 55.** Matriz de confusión con los resultados de las predicciones del dataset de test.

Viendo los resultados obtenidos, se confirma que el entrenamiento de nuestra red neuronal convolucional ha sido un éxito, obteniendo unos muy buenos resultados con el dataset de test. Para saber que los resultados han sido un éxito tenemos que entender lo que nos dice la matriz de confusión. Los componentes de la matriz de confusión son:

- **Columnas:** predicciones de cada clase.
- **Filas:** clase actual

Es decir, en las columnas encontramos todas las predicciones que se han hecho sobre una clase. Si la predicción coincide con la clase que es, entonces estamos ante una predicción exitosa. Cualquier otra predicción que coincida en otra fila es una predicción mala. De esta forma, en la diagonal principal de la matriz encontramos todas las predicciones exitosas. Sabiendo esto, se ve que casi todas las predicciones son exitosas, menos algunas, sobre todo algunas predicciones de la clase 0 que no pertenecen realmente a ella.

Con el modelo listo, toca exportar los pesos obtenidos para poder usarlos en nuestro sistema. Los exportamos y creamos una clase dedicada a clasificar imágenes. En esta clase definimos la

arquitectura que teníamos cuando creamos nuestro modelo, y creamos los métodos necesarios para cargar una imagen en forma de array y clasificarla. Para realizar la clasificación, primero se realiza una predicción en el modelo base (VGG16), y el resultado se pasa a nuestro modelo creado. De las predicciones se extrae el índice con mayor puntuación, que será la especie predicha. Según el umbral de confianza que se establezca, si esta puntuación supera el umbral entonces se acepta la clasificación, de lo contrario el resultado nos dice nada.

Si la clasificación supera el umbral, entonces se devuelve la siguiente información:

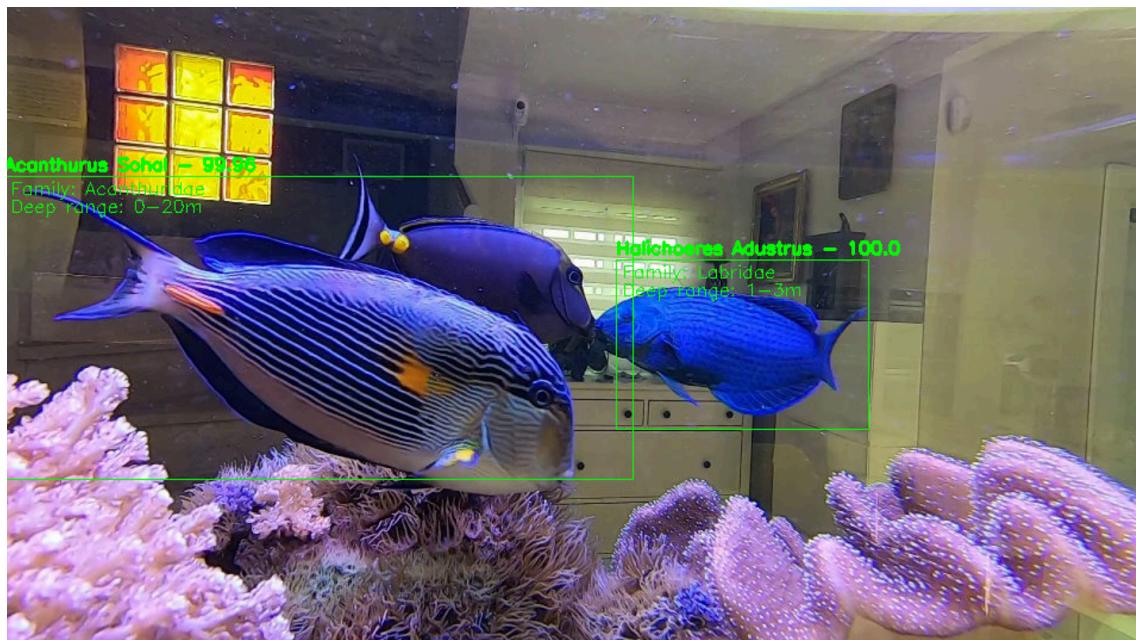
- ID especie
- Nombre de la especie
- Familia de la especie
- Rango de profundidad a la que se encuentra la especie

En caso de no superar el umbral se devuelven valores nulos (None).

### 2.6.5. SPRINT 5

Con los dos modelos listo, toca ponerlos en funcionamiento a ambos. Simplemente hay que conectar correctamente la salida de las detecciones con el clasificador, para usar las clasificaciones posteriormente, en una de las dos salidas definidas. Ahora vamos a comentar el caso de la implementación donde se muestra toda la información sobre el propio vídeo.

Esto se consigue cogiendo las detecciones de cada fotograma y enviándolas a la CNN para clasificarlas. La forma de pasar estas detecciones es recortando las detecciones de la imagen, y pasándolas al clasificador en forma de array (el recortado se realiza usando *slicing* de listas). Para cada detección será devuelta una respuesta del clasificador, la cuál se va a emplear para pintar la detección con su información correspondiente. En este caso, se ha configurado el sistema de tal forma que si las detecciones superan el umbral establecido a la hora de ser clasificadas, entonces se muestra la detección rodeada por una caja, encima de la caja el nombre con la puntuación de la clasificación, y dentro de la casa se muestra información relativa a la familia que pertenece cada especie y su localización (un rango de profundidad, por ejemplo 2-10 metros).



**Figura 56.** Ejemplo de la salida del sistema donde se muestran las detecciones con información sobre ellas.

#### 2.6.6. SPRINT 6

Con la salida por vídeo implementada, vamos a implementar el otro tipo de salida que se ha propuesto. Para ello, se comenzó diseñando la ventana donde irá el vídeo con su respectiva información. El diseño de la ventana se ha realizado con la librería Tkinter.

De esta forma, se diseña la ventana para que albergue el vídeo, dándole los siguientes tamaños:

- Altura = altura del vídeo
- Anchura = anchura del vídeo \* (3/2)

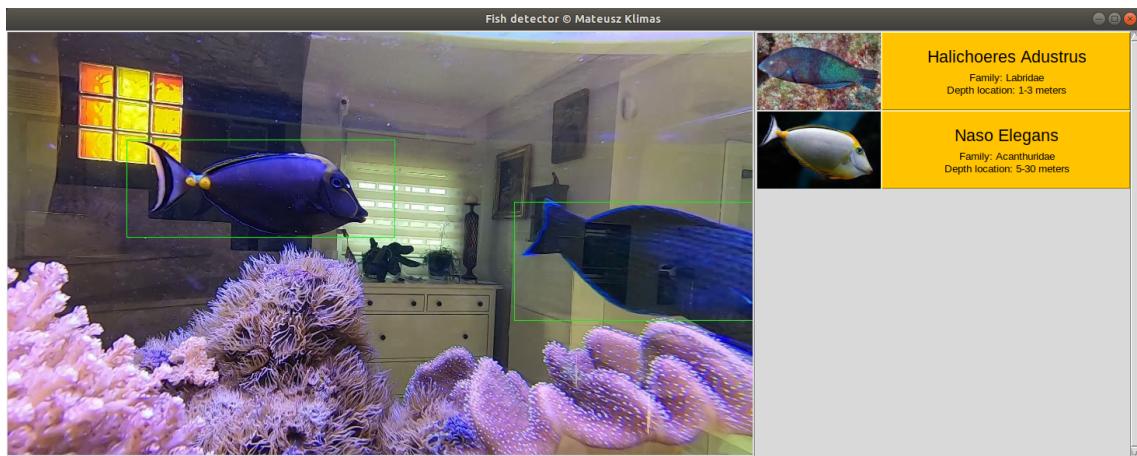


**Figura 57.** Ventana vacía donde se mostrará el vídeo con la información.

Para la creación de esta ventana, en primer lugar, se instancia una ventana. Esta ventana se divide en dos Frames, uno de ellos para el vídeo y el otro para la información. En el Frame izquierdo irá el vídeo, y en el derecho más Frames relativos a cada clasificación. Cada vez que haya peces que se hayan clasificado en un momento dado, entonces se crea un Frame para cada uno en el Frame derecho.

Antes de poder reproducir el vídeo con su información, hay que añadir unas operaciones en el modelo de detección. Básicamente se han añadido tres métodos: uno que cada realiza las clasificaciones del fotograma correspondiente y devuelve los resultados en forma de lista, otro que realiza el procesado del vídeo realizando las clasificaciones cada intervalo de tiempo, y el último que guarda información relativa al vídeo y a todas las clasificaciones hechas en un JSON.

Con el archivo de vídeo y el JSON se va reproduciendo el vídeo sobre la ventana, y a través de un contador de fotogramas y la información almacenada del procesado, se va mostrando la información en la parte derecha de la ventana.



**Figura 58.** Ventana mostrando el vídeo con información.



## CONCLUSIONES Y FUTURAS LÍNEAS

Una de las principales limitaciones que hemos tenido a lo largo de este proyecto ha sido la falta de dataset de imágenes relacionadas a la biología marina, en concreto de los peces. Esto ha conllevado una demora en la planificación del proyecto que ha sido resuelta a través de la obtención manual de los datos, los cuales fueron preparados para su correcto uso.

Con el desarrollo de esta primera versión de la aplicación es posible que se consiga la atención y concienciación necesaria para empezar a contribuir en este tema tan interesante como es la biología marina, ya que se parte de una base más sólida sobre la que trabajar, ahorrando mucho esfuerzo inicial que conlleva la creación de una colección de imágenes preparadas para ser usadas.

El desarrollo de la aplicación nos ha permitido alcanzar el objetivo principal que no es otro que diseñar e implementar un modelo que sea capaz de realizar la detección de cualquier especie de pez sobre vídeos, alcanzando unos resultados que permiten realizar detecciones en diversas situaciones, aunque todavía hay mucho margen de mejora. Este margen de mejora se encuentra principalmente en la obtención de más datos con una gran variedad de peces, posiciones y entornos para ser usados en el modelo de detección. Sobre todo, haría falta más imágenes en la que los peces se encuentren ocluidos por algún objeto y en los que aparezcan lejos (que sean pequeños en la imagen).

Para el apartado de las clasificaciones se empleó el modelo VGG16 como base, lo que permitió lograr una precisión alta (un 98% de precisión para el conjunto de test), ya que al crear un modelo que yace sobre uno ya bien conformado ha permitido obtener unos resultados muy buenos con muchos menos datos. Con tan solo una media de 150 imágenes por clase, las cuales posteriormente han sido modificadas mediante técnicas de data augmentation hasta 1000, se ha conseguido una precisión del 98,51%.

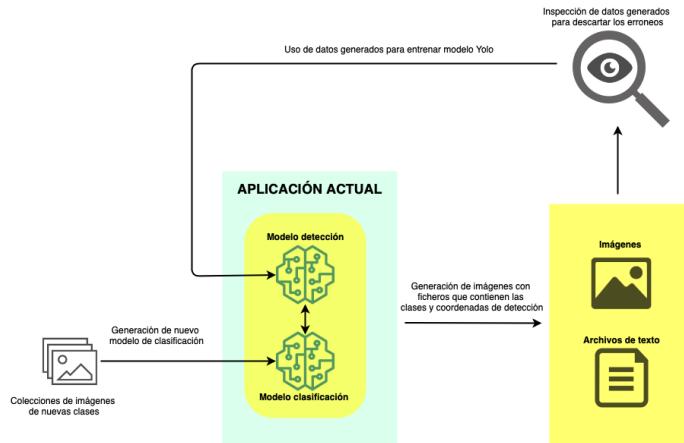
Además, se ha conseguido obtener una aplicación que permite detectar y clasificar en un entorno específico, aunque quedaría por mejorar el apartado del tiempo de procesamiento. La idea es conseguir que se pueda hacer en tiempo real.

Fruto del trabajo desarrollado se ha detectado distintas líneas futuras de trabajo que pasamos a comentar seguidamente:

En primer lugar, se propone como una posible línea futura la ampliación de la colección de imágenes para el modelo de detección, para así darle más robustez a la aplicación.

Otra línea futura que se propone es la de creación de modelos específicos para diferentes lugares, como puede ser por ejemplo el Mediterráneo. La idea es crear colecciones de datos grandes para cada especie, para así poder usarlas en el modelo que se encarga de detectar todo tipo de peces, pero en vez de generalizar, esta vez se trataría cada especie como una clase diferente. De esta forma se lograría quitar la segunda parte que hay implementada ahora, que es la encargada de realizar las clasificaciones, y entonces se podría llegar a realizar detecciones en tiempo real debido a que solo se emplearía una red neuronal. Para evitar tener que crear estas grandes colecciones de datos manualmente, la idea es emplear la aplicación actual, pero cambiando la parte de las clasificaciones. Para cada especie nueva que se introduzca habría que proporcionar unas 200 imágenes para empezar. Con estas imágenes se reentrenaría el modelo de clasificación y estaría listo para usar. A partir del nuevo modelo generado habría que configurar la aplicación para que vaya extrayendo los fotogramas con información relativa a las clases que detecta y sus coordenadas, para almacenarlos y emplearlos para el entrenamiento del modelo final. Si se realizase grabaciones específicas de cada especie se podrían obtener muchos datos para cada clase. En la Figura 59 se puede ver un esquema de lo explicado.

Por último, se propone la implementación de esta aplicación en gafas de buceo. La idea es emplear los modelos que se generasen para la detección en tiempo real en estas gafas de buceo de realidad aumentada.



**Figura 59.** Esquema de una posible implementación futura.

Para terminar esta sección, me gustaría hacer un llamamiento al cuidado del mar para que se pueda seguir conservando todas las especies de peces y otro tipo de seres vivos que existen actualmente. Estamos en una situación en la que se vierte una cantidad inimaginable de desechos al mar, lo cuál produce que poco a poco se vaya reduciendo la biodiversidad de los mares y océanos debido a la alta contaminación que esto produce. Si esta situación no se solventa pronto, este trabajo podría carecer de sentido en un futuro, debida a la posible desaparición de grandes ecosistemas submarinos.

# **MANUALES**



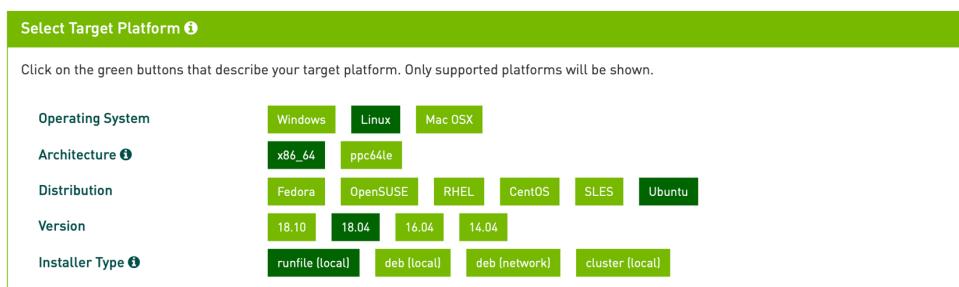
### 3.1. MANUAL DE INSTALACIÓN

Antes de poder usar la aplicación, es necesario instalar una serie de herramientas para poder poner en marcha el sistema. A continuación, se va a enumerar todo lo necesario, y se va a ir explicando como instalar cada parte:

- OpenCV 3.4 o inferior instalado en el sistema operativo.
- OpenCV 4 instalado en un entorno virtual de Python.
- CUDA 10.1
- cuDNN v7.6.0 para CUDA 10.1
- Keras

#### 3.1.1. INSTALACIÓN CUDA

Empezamos dirigiéndonos a la página de NVIDIA para descargar CUDA [40]. Al haber desarrollado la aplicación usando Ubuntu 18.04 LTS descargamos el *runfile (local)* para esta versión.



**Figura 60.** Venta de descarga de CUDA para Ubuntu 18.04.

Una vez tengamos el archivo para la instalación descargado, seguimos los pasos indicados para su instalación. Si estamos instalando CUDA en Ubuntu, no nos tenemos que olvidar de añadirlo en el PATH o en el archivo .bashrc:

```
export PATH=/usr/local/cuda-10.1/bin${PATH:+:$PATH}$
export LF_LIBRARY_PATH=/usr/local/cuda-10.1/lib64${LD_LIBRARY_PATH:+:$LD_LIBRARY_PATH}
```

#### 3.1.2. INSTALACIÓN OPENCV EN SISTEMA OPERATIVO

Lo primero a tener en cuenta es que la versión de esta instalación tiene que ser la 3.4 o inferior. Esto es muy importante, debido a que se emplean funciones que en futuras versiones son cambiadas o eliminadas, lo cual puede hacer que la aplicación no nos funcione. Empezamos comprobando si hay actualizaciones de paquetes:

```
sudo apt-get update
sudo apt-get upgrade
```

Hecho esto, tenemos que instalar todas las dependencias necesarias:

```
sudo apt-get install build-essential cmake git libgtk2.0-dev pkg-config libavcodec-dev libavformat-dev libswscale-dev
sudo apt-get install python3.6-dev python3-numpy libtbb2 libtbb-dev
sudo apt-get install libjpeg-dev libpng-dev libtiff5-dev libjasper-dev libdc1394-22-dev libeigen3-dev libtheora-dev libvorbis-dev libxvidcore-dev libx264-dev sphinx-common libtbb-dev yasm
```

*libfaac-dev libopencore-amrnb-dev libopencore-amrwb-dev libopenexr-dev libgstreamer-plugins-base1.0-dev libavutil-dev libavfilter-dev libavresample-dev*

Ahora tenemos que descargar del repositorio de OpenCV la versión 3.4 y el repositorio *opencv\_contrib* y descomprimirlos:

```
wget -O opencv.zip https://github.com/opencv/opencv/archive/3.4.0.zip
wget -O opencv_contrib.zip https://github.com/opencv/opencv_contrib/archive/3.4.0.zip
unzip opencv.zip
unzip opencv_contrib.zip
```

Y lo instalamos:

```
cd opencv
mkdir release
cd release
cmake -D ENABLE_PRECOMPILED_HEADERS=OFF -D BUILD_TIFF=ON -D
WITH_CUDA=OFF -D ENABLE_AVX=OFF -D WITH_OPENGL=OFF -D WITH_OPENCL=OFF -
D WITH_IPP=OFF -D WITH_TBB=ON -D BUILD_TBB=ON -D WITH_EIGEN=OFF -D
WITH_V4L=OFF -D WITH_VTK=OFF -D BUILD_TESTS=OFF -D BUILD_PERF_TESTS=OFF -D
CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local -D
OPENCV_EXTRA_MODULES_PATH=../../opencv_contrib/modules ../../opencv/
make -j4
make install
ldconfig
```

Para comprobar que se ha instalado, introducimos el siguiente comando:

```
pkg-config --modversion opencv
```

```
mateusz@mateusz:~$ pkg-config --modversion opencv
3.4.0
```

**Figura 61.** Mensaje con confirmación de que esta instalada la versión 3.4 de OpenCV

Si probando el comando anterior nos da error, entonces hay que introducir los siguientes comandos y volver a intentarlo:

```
sudo apt-get install libopencv-dev
sudo apt-get install pkg-config
```

#### Instalación OpenCV en entorno virtual Python

La instalación de OpenCV en un entorno virtual de Python es muy parecido al caso anterior. Comenzamos descargándonos la versión más reciente de OpenCV:

```
git clone https://github.com/opencv/opencv.git
git clone https://github.com/opencv/opencv_contrib.git
```

Instalamos las dependencias de Python (las dependencias de OS ya las tenemos instaladas de antes):

```
sudo apt-get install python-dev python-pip python3-dev python3-pip
sudo -H pip3 install -U pip numpy
```

Lo siguiente que necesitamos es instalar lo necesario para poder usar entornos virtuales en Python 3:

```
sudo pip3 install virtualenv virtualenvwrapper  
mkvirtualenv cv -p python3
```

Con el entorno virtual listo, pasamos a instalar OpenCV:

```
cd opencv  
mkdir release  
cd release  
cmake -D CMAKE_BUILD_TYPE=RELEASE \  
-D CMAKE_INSTALL_PREFIX=/home/mateusz/opencvpy/installation \  
-D INSTALL_PYTHON_EXAMPLES=ON \  
-D INSTALL_C_EXAMPLES=OFF \  
-D OPENCV_ENABLE_NONFREE=ON \  
-D OPENCV_EXTRA_MODULES_PATH=../../opencv_contrib/modules \  
-D PYTHON_EXECUTABLE=~/.virtualenvs/cv/bin/python \  
-D WITH_OPENGL=ON \  
-D BUILD_EXAMPLES=ON ..  
make -j4  
make install  
ldconfig
```

Por último, tenemos que encontrar el binario de OpenCV de Python (cv2.so) y enlazarlo:

```
#Localizar el binario  
find /usr/local/lib/ -type f -name "cv2*.so"  
#Binario en dist-packages  
/usr/local/lib/python3.7/dist-packages/cv2.cpython-37m-x86_64-linux-gnu.so  
## Binario en site-packages  
/usr/local/lib/python3.6/site-packages/cv2.cpython-36m-x86_64-linux-gnu.so  
#Linking  
cd ~/.virtualenvs/facecourse-py3/lib/python3.7/site-packages  
ln -s /usr/local/lib/python3.7/dist-packages/cv2.os
```

### 3.1.3. INSTALACIÓN KERAS

Para instalar Keras, tan solo hay que estar dentro del entorno virtual de Python y poner el siguiente comando:

```
pip install keras
```



## 3.2. MANUAL DE USUARIO

Existe un script llamado “*makeDetections.py*” al cual le tenemos que pasar los siguientes parámetros:

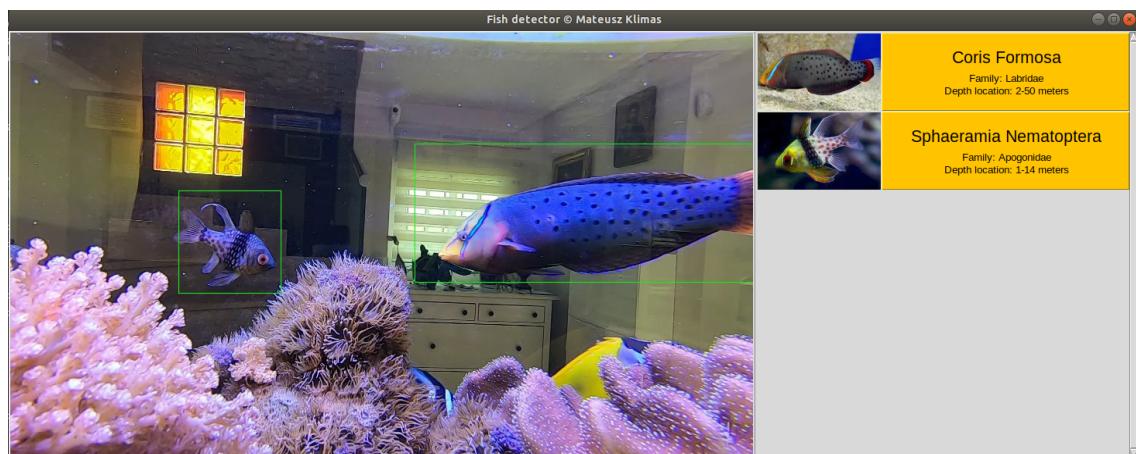
- Nombre del fichero de vídeo
- Umbral de confianza para las clasificaciones
- Tipo de salida para el resultado: 0 para mostrar la información directamente, 1 para procesar el vídeo que posteriormente se empleará para mostrar la información con las detecciones en una ventana.

Una vez que se ejecute el script se empezará a procesar el vídeo. Durante este proceso se muestra una barra con el progreso que falta para que se complete. Si se ha elegido el modo 0, entonces se genera un vídeo con el nombre “*output.avi*” con las detecciones y la información implícita. En caso de que se use el modo 1, entonces se generarán dos archivos: el vídeo con el nombre “*output.avi*” y un archivo JSON con el nombre “*data.json*” que contiene información relativa a las detecciones. Para poder mostrar la ventana con toda esta información hay que usar otro script, el cuál tiene el nombre “*showInfoWindow.py*”. Es muy importante tener en cuenta que los archivos generados tienen que tener el nombre original con el que fueron creados, o de lo contrario nos dará error, o se mostrará información que no es.

Si hacemos uso del modo 0, tendrían que salirnos resultados como los de la Figura 62. Si se usa el modo 1, entonces los resultados tendrían que tener una apariencia similar a la de la Figura 63.



**Figura 62.** Ejemplo de como se tendría que ver el vídeo resultante al ser usado el modo 0.



**Figura 63.** Ejemplo de como se tendría que ver el vídeo con la información al usar el modo 1.

## PRESUPUESTO DEL PROYECTO



## PRESUPUESTO DEL PROYECTO

Para la realización del presupuesto de la aplicación propuesta, se ha dividido este en costes materiales (hardware) y costes de desarrollo (mano de obra).

### COSTES MATERIALES

En los costes materiales incluimos el equipo que se ha empleado para el desarrollo de la aplicación. El cálculo de los costes presentes se ha realizado con los precios vigentes en el primer semestre del año 2019.

**Tabla 10.**

Costes materiales del proyecto.

Descripción	Cantidad	Precio	Precio total
Placa base MSI Z270 MPower Gaming Titanium	1	280	280
Procesador Intel Core i7-7700K 4.2GHz	1	373	373
Tarjeta gráfica Gigabyte VGA Gigabyte Nvidia G-Force GTX 1080 G1	1	540	540
Memorias RAM G.Skill Ripjaws V Red DDR4 2400 PC4-19200 16GB 2x8GB CL15	1	89	89
Disco de almacenamiento Samsung 860 EVO Basic SSD 1TB SATA3	1	140	140
Fuente de alimentación EVGA Supernova G3 750W 80 Plus Gold Modular	1	136	136
Ventilador Noctua NH-D15	1	90	90
Caja Corsair 230T Graphite Series Windowed Negra	1	85	85
<b>TOTAL</b>			<b>1733</b>

### COSTE DE DESARROLLO

Para el cálculo de los costes de desarrollo, se ha empleado la planificación del proyecto con los tiempos establecidos en cada sprint, donde se ha trabajado aproximadamente 4 horas por días, 5 días a la semana. Para obtener el coste/hora de cada cargo se ha consultado numerosas páginas de empleo para sacar el salario anual bruto, el que luego se multiplicará por 1.3 (costes de empresa) y se dividirá entre todas las horas anuales, que son 1944 para una jornada completa.

**Tabla 11.**

Costes de diferentes cargos.

Cargo	Salario anual bruto (€)	Salario con costes de empresa (+ 30%)	Coste/Hora (€)
Ingeniero Técnico Informático	40.000	52.000	26.75
Analista	30.000	39.000	20.06
Programador	35.000	45.500	23.41
Secretario/a	15.000	19.500	10.03

**Tabla 12**

Costes de desarrollo del proyecto.

Cargo	Coste/Hora €	Horas	Coste total
Jefe de proyecto (Ingeniero Técnico Informático)	26.75	20	535
Analista	20.06	160	3209.6
Programador	23.41	220	5150.2
Secretario/a	10.03	80	802.4
<b>TOTAL</b>			<b>9697.2</b>

## GASTOS GENERALES

A parte de los costes materiales descritos, se debe incluir una serie de gastos que pueden aparecer a lo largo del desarrollo.

En primer lugar será necesario material de oficina, como puede ser material de escritura, cuadernos, folios, cartuchos para la impresora, etc.

Además de esto, es probable que sea necesario incluir dietas para los empleados, costear los traslados, ofrecer medios de comunicación, etc.

Para ello, se ha establecido aplicar un recargo del 15% a la suma de los costes materiales y de desarrollo.

**Tabla 13**

Gastos generales del proyecto.

Costes	Cantidad (€)	Recargo 15% (€)
Costes materiales	1733	259.95
Costes de desarrollo	9697.2	1454.58
Gastos generales		1714.53

## COSTES TOTALES

Los costes totales se obtienen de sumar los costes materiales con los costes de desarrollo.

**Tabla 14**

Costes totales del proyecto.

Tipo de coste	Coste (€)
Hardware	1733
Desarrollo	9697.2
General	1454.58
<b>TOTAL</b>	<b>13144.73</b>

## BIBLIOGRAFIA



## BIBLIOGRAFIA

- [1] Real Academia Española, *Diccionario de la Real Academia Española*, Disponible en:  
<https://dle.rae.es/> (Junio 2019)
- [2] FishBase, *FishBase: A Global Information System on Fishes*, Disponible en:  
<http://www.fishbase.org/home.htm>
- [3] ImageNet, *ImageNet Large Scale Visual Recognition Competition (ILSVRC)*, Disponible en:  
<http://image-net.org/challenges/LSVRC/> (Junio 2019)
- [4] Coco Dataset, *COCO – Common Objects in Context*, Disponible en:  
<http://cocodataset.org/#home> (Mayo 2019)
- [5] Joseph Chet Redmon, *Survival Strategies for the Robot Rebellion*, Disponible en:  
<https://pjreddie.com/darknet/yolo/> (Junio 2019)
- [6] Augment, *3D and augmented reality product visualization platform | Augment*, Disponible en:  
<https://www.augment.com>
- [7] The Verge, *IMAX is shutting down its virtual reality arcade business for good – The Verge*, Disponible en:  
<https://www.theverge.com/2018/12/13/18139981/imax-virtual-reality-arcades-shut-down-write-off-los-angeles-bangkok-toronto>
- [8] Niantic, *Homepage | Pokémon Go*, Disponible en:  
<https://www.pokemongo.com/en-us/>
- [9] Beat Games, *Beat Saber – VR rythm game*, Disponible en:  
<https://beatsaber.com>
- [10] YOUTUBE, “OpenCV+Aruco+OpenGL Testing”, en Youtube  
<<https://www.youtube.com/watch?v=IU7SoMvWNeA>>
- [11] OpenCV, *OpenCV*, Disponible en:  
<https://opencv.org>
- [12] ThinkMobiles, *Use of Augmented Reality Reality in Education: tools, apps and tips -2019*, Disponible en:  
<https://thinkmobiles.com/blog/augmented-reality-education/>

- [13] Volvo, *HoloLens* | Volvo Cars UK, Disponible en:  
<https://www.volvocars.com/uk/about/humanmade/projects/hololens>
- [14] Griffon Webstudios, *sephora-augmented-reality-app* – Griffon Studios, Disponible en:  
<https://griffonwebstudios.com/augmented-reality-in-marketing/sephora-augmented-reality-app/>
- [15] NVIDIA, *The Difference Between AI, Machine Learning and Deep Learning?* | NVIDIA Blog, Disponible en:  
<https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/>
- [16] SAS, *Machine Learning: What it is and why it matters* | SAS, Disponible en:  
[https://www.sas.com/en\\_us/insights/analytics/machine-learning.html](https://www.sas.com/en_us/insights/analytics/machine-learning.html)
- [17] MathWorks, *Machine Learning: Tres cosas que es necesario saber – MATLAB & Simulink*, Disponible en:  
<https://es.mathworks.com/discovery/machine-learning.html>
- [18] Nils J. Nilsson, *Introduction to Machine Learning*, 1998.
- [19] Wikipedia, *Neuron – Neurona* – Wikipedia, Disponible en:  
<https://ast.wikipedia.org/wiki/Neurona#/media/Ficheru:Neurona.svg>
- [20] MathWorks, *screenshot.png 655x318 pixeles*, Disponible en:  
<https://es.mathworks.com/matlabcentral/mlc-downloads/downloads/83c1a385-9192-4c78-b695-8d6c73d0a6a5/8efda0f4-dc26-44a5-87b0-13a57efefbd6/images/screenshot.png>
- [21] Wikipedia, *Activation logistic – Activation function* – Wikipedia, Disponible en:  
[https://en.wikipedia.org/wiki/Activation\\_function#/media/File:Activation\\_logistic.svg](https://en.wikipedia.org/wiki/Activation_function#/media/File:Activation_logistic.svg)
- [22] Wikipedia, *Activation rectified linear – Activation function* – Wikipedia, Disponible en:  
[https://en.wikipedia.org/wiki/Activation\\_function#/media/File:Activation\\_rectified\\_linear.svg](https://en.wikipedia.org/wiki/Activation_function#/media/File:Activation_rectified_linear.svg)
- [23] Wikipedia, *Activation prelu – Activation function* – Wikipedia, Disponible en:  
[https://en.wikipedia.org/wiki/Activation\\_function#/media/File:Activation\\_prelu.svg](https://en.wikipedia.org/wiki/Activation_function#/media/File:Activation_prelu.svg)
- [24] Computer Science Wiki, *MaxpoolSample2.png 750x313 pixeles*, Disponible en:  
<https://computersciencewiki.org/images/8/8a/MaxpoolSample2.png>

- [25] Computer Science Wiki, *MaxpoolSample.png* 634x417 pixeles, Disponible en: <https://computersciencewiki.org/images/9/9e/MaxpoolSample.png>
- [26] Alex Krizhevsky, *ImageNet Classification with Deep Convolutional Neural Networks*, 2012.
- [27] Karen Simonyan & Andrew Zisserman, *Very Deep Convolutional Networks for Large-Scale Image Recognition*, 2014.
- [28] Google, *Going Deeper with Convolution*, 2014.
- [29] Ross Girshick, *Rich feature hierarchies for accurate object detection and semantic segmentation*, 2013.
- [30] Ross Girshick, *Fast R-CNN*, 2015.
- [31] Shaoqing Ren, *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*, 2015.
- [32] Joseph Chet Redmon, *You Only Look Once: Unified, Real-Time Object Detection*, 2015.
- [33] Joseph Chet Redmon, *YOLO9000: Better, Faster, Stronger*, 2017.
- [34] Joseph Chet Redmon, *YOLOv3: An Incremental Improvement*, 2018.
- [35] Apple, *Acerca de las personas en Fotos del iPhone, iPad o iPod Touch – Soporte Técnico de Apple*, Disponible en: <https://support.apple.com/es-es/HT207103>
- [36] PlantSnap, *PlantSnap – Plant Identifier App, #1 Mobile App for Plant Identification*, Disponible en: <https://www.plantsnap.com>
- [37] draw.io, *draw.io*, Disponible en: <https://www.draw.io>

[38] Tzutalin labelimg, *tzutalin: LabelImg is a graphical image annotation tool and label object bounding boxes in images*, Disponible en:  
<https://github.com/tzutalin/labelImg>

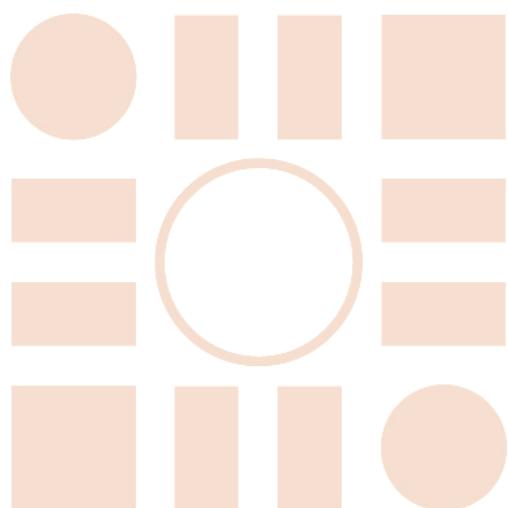
[39] AlexeyAB, *AlexeyAB/darknet: Windows and Linux version of Darknet Yolo v3 & v2 Neural Networks for object detection (Tensor Cores are used)*, Disponible en:  
<https://github.com/AlexeyAB/darknet>

[40] NVIDIA, *CUDA Toolkit 10.1 Update 1 Download | NVIDIA Depeloper*, Disponible en:  
<https://developer.nvidia.com/cuda-downloads>

[41] Bruno Arnaldi, *Virtual reality and Augmented Reality: Myth and Realities*, 2018.



Universidad de Alcalá  
Escuela Politécnica Superior



ESCUELA POLITECNICA  
SUPERIOR

