ESCUELA TÉCNICA SUPERIOR DE INGENIEROS
INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

*Trabajo Fin de Grado*

# DETECCIÓN AUTOMÁTICA DE ESPECIES BENTÓNICAS EN FONDOS PROFUNDOS MEDIANTE IMÁGENES OBTENIDAS CON VEHÍCULOS REMOTOS Y TÉCNICAS DE DEEP-LEARNING

**(Automated analysis of benthic species in deep bottoms using images from remote operated vehicles and deep-learning techniques)**

**Para acceder al Título de**

***Graduado en***

***Ingeniería de Tecnologías de Telecomunicación***

**Autor: Celia Díaz Cuesta**

**Julio - 2018**

**GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN**

CALIFICACIÓN DEL TRABAJO FIN DE GRADO

**Realizado por: Celia Díaz Cuesta**
**Director del TFG: Elena Prado Ortega**
**Título: "Detección automática de especies bentónicas en fondos profundos mediante imágenes obtenidas con vehículos remotos y técnicas de deep-learning"**
**Title: "Automated analysis of benthic species in deep bottoms using images from remote operated vehicles and deep-learning techniques"**

**Presentado a examen el día: 27/07/2018**

para acceder al Título de

**GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN**

Composición del Tribunal:
Presidente (Apellidos, Nombre): Adolfo Cobo García
Secretario (Apellidos, Nombre): Jesús Ibáñez Díaz
Vocal (Apellidos, Nombre): Jesús Mirapeix Serrano

Este Tribunal ha resuelto otorgar la calificación de: .....

Fdo.: El Presidente

Fdo.: El Secretario

Fdo.: El Vocal

Fdo.: El Director del TFG
(sólo si es distinto del Secretario)

Vº Bº del Subdirector

Trabajo Fin de Grado Nº
(a asignar por Secretaría)

Trabajo realizado en parte con los medios aportados por los proyectos TEC2016-76021-C2-2-R (Ministerio de Economía y Competitividad) y Photomare (Sodercan, Gobierno de Cantabria).

Las imágenes utilizadas para este trabajo han sido obtenidas dentro del Plan de Seguimiento del Área Marina Protegida de El Cachucho a partir del proyecto ECOMARG del Instituto Español de Oceanografía

# *AGRADECIMIENTOS*

*En primer lugar, agradezco este trabajo a Adolfo, por todo el tiempo dedicado y apoyo prestado, sin él, este proyecto no habría salido adelante. Por supuesto, agradecer al Instituto Español de Oceanografía, en especial a Elena por todo el contenido y ayuda aportado.*

*Agradecer también a mí familia, en especial a mis padres y mi hermano, por todos los sacrificios que han tenido, enseñándome que con perseverancia y constancia se puede conseguir todo.*

*A ti, Javier, por apoyarme y demostrarme que puedo conseguir todo lo que me proponga. Gracias.*

*Y, por último, a mis amigos, a los de siempre, que a pesar de la distancia en estos años de carrera seguimos apoyándonos y queriendo incluso más. Y, por supuesto, a todos estos amigos que he ido formando a lo largo de estos años.*

# *RESUMEN*

*Este trabajo pretende aplicar técnicas de procesado de imágenes basadas en Deep-learning, para la detección, clasificación y análisis automático de especies marinas en fondos de gran profundidad, a partir de imágenes obtenidas con vehículos remotamente operados (ROV).*

*En concreto se estudiará el uso de redes CNN pre-entrenadas y recopilando una base de datos de imágenes específicas y con técnicas de transfer-learning, se evaluará la capacidad de estas redes para la detección automática de especies específicas de interés para el seguimiento medioambiental.*

# *ABSTRACT*

*This project results in techniques of image processing in Deep-learning, for the detection, classification and automatic analysis of marine species in deep depths, from images with vehicles operated at a distance (ROV).*

*In particular, the use of CNN networks with a database of images and deep-learning techniques will be studied, the capacity of these networks for the automatic detection of specific species of interest for environmental monitoring will be evaluated.*

# *INDICE*

# *INDICE DE FIGURAS*

# *INDICE DE TABLAS*



# *INDICE DE ECUACIONES*

# 1. Introducción

## 1.1. Objetivo

El objetivo principal de este trabajo es poder solucionar un tema importante dentro del estudio del hábitat marino, tener un seguimiento para la conservación de estos hábitats. Para ello, se realizará de forma automática la detección de especies bentónicas en un conjunto de vídeos del fondo marino. Esto podrá servir, entre otras cosas, para estudiar la densidad de una especie en concreto, en ciertos intervalos de tiempo. Actualmente, esta labor se realiza de forma manual por un Biólogo que visualiza los vídeos, una tarea muy tediosa o que simplemente no puede hacerse por falta de personal.

Por ello, se va a proceder a diseñar e implementar una red neuronal artificial (RNA) de tipo *Deep Learning*, para así poder clasificar automáticamente especies propias del fondo marino, así como erizos, esponjas marinas, gorgonias…

Para ello, con la ayuda de unos vídeos proporcionados por el Instituto Español de Oceanografía (IEO), se hará una comparativa de cada fotograma de dichos vídeos con la red neuronal que se deberá entrenar. Este procedimiento será explicado con detalle en posteriores capítulos.

## 1.2. Estructura

Este proyecto se divide en capítulos, los cuales están estructurados de la siguiente forma:

- **Capítulo 1: Introducción**

  Se realizará una breve introducción, donde se realizará una breve descripción de este proyecto, donde se puede encontrar el objetivo principal, así como la estructura del contenido.

- **Capítulo 2: Fundamentos teóricos.**

  Explicación teórica sobre el *Deep Learning*. Entrando en detalle en otros temas como los tipos de redes neuronales que se van a mencionar a lo largo del proyecto, qué es el *Transfer Learning* y en qué consiste el *Data Augmentation*.

- **Capítulo 3: Procedimiento**

Explicación en detalle de todo el procedimiento, incluyendo los errores y problemas detectados, hasta llegar a la solución final.

- **Capítulo 4: Conclusiones y líneas futuras**

Por último, se tratarán las conclusiones sacadas, donde se compararán dos tipos de redes neuronales, y alguna diferencia entre los tipos de entrenamiento realizados. Y, por último, se comentará brevemente la visión de futuro de este trabajo.

# 2. Fundamentos teóricos

## 2.1. Deep Learning

En primer lugar, se va a definir qué es el *Deep Learning* y cómo funciona, en capítulos siguientes se explicará cómo se va a aplicar a la hora de analizar vídeos marinos.

El *Deep Learning* o también llamado aprendizaje profundo, es un conjunto de algoritmos de aprendizaje automático, o *Machine Learning,* que llevan a cabo un aprendizaje de principio a fin. No es algo nuevo, pero sí ha aumentado su popularidad a lo largo de los últimos años, esto se debe, entre otras, a las siguientes razones:

- Los investigadores han demostrado que al utilizar *Deep Learning* pueden obtenerse resultados notablemente mejores que utilizando técnicas estándar de visión artificial.
- La computación con GPU ha permitido que redes neuronales profundas puedan entrenarse en un tiempo razonable
- Gracias a la colaboración abierta entre investigadores, se ha conseguido grandes cantidades de etiquetados.

Como se ha comentado, el aprendizaje automático lleva a cabo un aprendizaje de principio a fin, es decir, recibe una serie de datos en bruto, una tarea a realizar al algoritmo y el algoritmo aprende por sí sólo a clasificarlo de modo automático.

A continuación, se explicará cómo es el flujo de trabajo clásico en *Machine Learning.* En primer lugar, se comienza trabajando con un conjunto de imágenes, para ello se realizará una extracción manual de características para estas imágenes, una vez ya extraídas estas características se procede a una clasificación utilizando diversos métodos, de modo que así se podrán clasificar correctamente las imágenes.

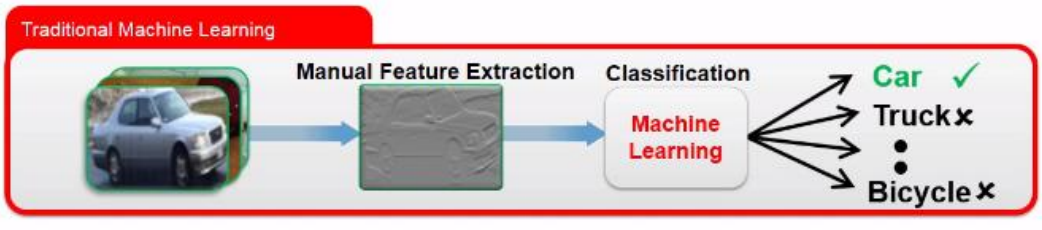


*Figura 1: Flujo de trabajo clásico en Machine Learning*

Por otro lado, el flujo de trabajo de *Deep Learning*, también se trabaja con un conjunto de imágenes a clasificar como en el caso anterior, pero en vez realizar una extracción manual, se pasa por una red neuronal convolucional (CNN), que por sí sola aprenderá las características directamente de las imágenes y las clasificará. Esto es lo que se conoce como un aprendizaje de principio a fin, llevando a cabo el aprendizaje de características como la clasificación.

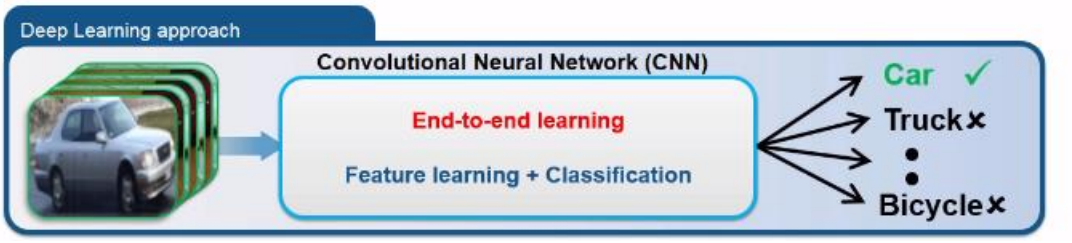


*Figura 2: Flujo de trabajo en Deep Learning*

Alcanza unos niveles de precisión muy elevados, por lo que contribuye a que la electrónica de consumo satisfaga las expectativas de los usuarios. Los avances en el ámbito del aprendizaje profundo han llegado a un punto en el que este supera a las personas en algunas tareas.

Es cierto que hasta ahora no ha empezado a ser útil el aprendizaje profundo, y esto se debe a dos motivos. El primero, se requiere unas grandes cantidades de datos etiquetados; y el segundo motivo, requiere una potencia de cálculo significativa. Las GPU de alto rendimiento tienen una arquitectura paralela que resulta eficiente para el aprendizaje profundo. Algunas de las aplicaciones que se pueden tener actualmente en cuenta, son:

- Conducción autónoma: empleando el *deep learning* para detectar señales, semáforos o peatones, por lo tanto, ayuda a reducir accidentes.
- Sector aeroespacial y de defensa: se utiliza para identificar objetos desde satélites que localizan áreas de interés e identifica las zonas seguras.
- Investigación médica: los investigadores de cáncer utilizan el *Deep Learning* para detectar células cancerígenas de forma automática.
- Automatización industrial: el aprendizaje profundo está ayudando a mejorar la seguridad de los trabajadores en entornos con maquinaria pesada, gracias a la detección automática de personas u objetos cuando se encuentran a una distancia no segura de las máquinas.
- Electrónica (CES): el aprendizaje electrónico se usa en la audición automatizada y la traducción del habla.

La mayor parte de los métodos de aprendizaje emplean arquitecturas de redes neuronales profundas. El término "profundo" suele hacer referencia al número de capas

ocultas en la red neuronal, las redes neuronales tradicionales sólo contienen dos o tres capas ocultas, en cambio las redes profundas pueden tener hasta 150 capas.

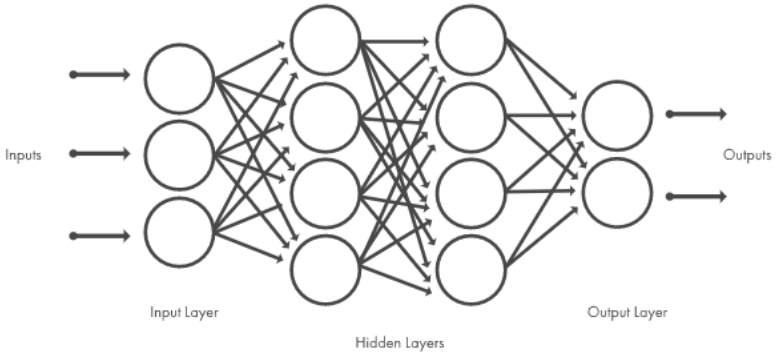


*Figura 3: Redes neuronales organizadas en capas*

## 2.2. Redes Neuronales

**Redes Neuronales Artificiales (RNA)**

Las redes neuronales artificiales son una técnica de aprendizaje y procesamiento automático inspirada en el funcionamiento del cerebro humano. Podemos definir las redes neuronales artificiales como una estructura de procesamiento paralelo masivo constituida por unas unidades muy sencillas (neuronas), que tienen la capacidad de almacenar conocimiento experimental y ponerla a disposición para su uso. Se asemejan a las redes neuronales biológicas en varios aspectos:

- Las neuronas son elementos simples y altamente interconectados, en cambio las neuronas artificiales son mucho más simples.
- Las conexiones entre las neuronas determinan la función de la red. Estas conexiones se utilizan para almacenar el conocimiento adquirido.
- El conocimiento es adquirido a partir de su entorno mediante un proceso de aprendizaje.

La arquitectura de la red hacer referencia a la disposición de las neuronas de la red, estas neuronas se organizan formando capas, de modo que la red puede consistir en una o más capas de neuronas.

Cada neurona recibe un conjunto de entradas multiplicadas por su conexión, que son sumados y operados por una función de transferencia antes de transmitirse a la siguiente capa o como salida de la red.

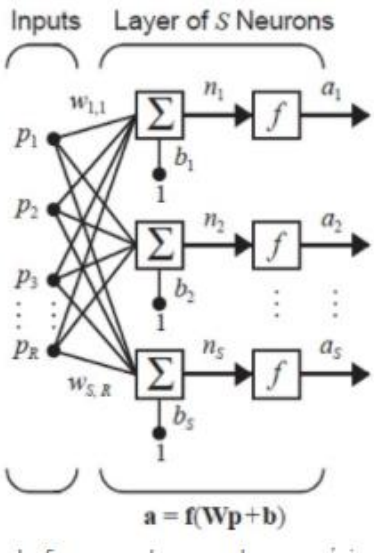


*Figura 4: Red con una única capa*

**Red Neuronal Convolucional (CNN)**

Una red neuronal convolucional es una red multicapa jerárquica comúnmente utilizada para el aprendizaje profundo. Las CNN a menudo se utilizan para reconocer objetos, aprenden directamente de los datos de la imagen, eliminando la necesidad de extracción manual de las características. Al eliminar este tipo de extracción, no es necesario identificar las características utilizadas para clasificar las imágenes.

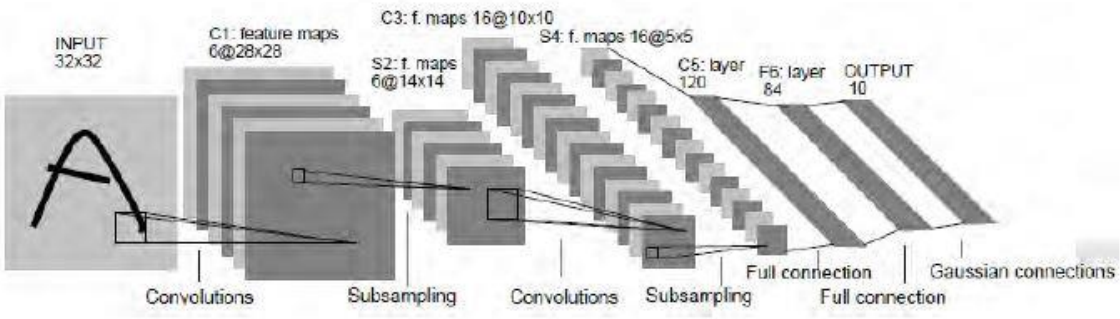


*Figura 5: Arquitectura de una CNN*

La CNN funciona mediante la extracción de características directamente de las imágenes. Las características relevantes no se entrenan previamente, se aprenden mientras la red se entrena con un conjunto de imágenes.

Esta extracción de características automatizada hace que los modelos de aprendizaje profundo sean muy precisos para tareas de visión artificial, tales como la clasificación de objetos.

Las CNN aprenden a detectar diferentes características de una imagen mediante decenas o cientos de capas ocultas. Cada capa oculta aumenta la complejidad de las características de la imagen aprendida.

Por ejemplo, la primera capa oculta podría aprender cómo detectar bordes, mientras la segunda aprende cómo detectar formas más complejas propias de la forma del objeto que se intenta reconocer.

A continuación, se observa una representación típica de una CNN, una representación de los datos a medida que viajan por la red. A alto nivel, es decir el primer conjunto de capas, estarán detectando características (sección *Feature Learning*). Las capas finales, se ocupan de la clasificación de la imagen (*Classification*).

Todas las capas se van a entrenar conjuntamente, es decir, el proceso de entrenamiento va a consistir en ajustar correctamente los pesos en todas estas capas para que las imágenes sean clasificadas correctamente.

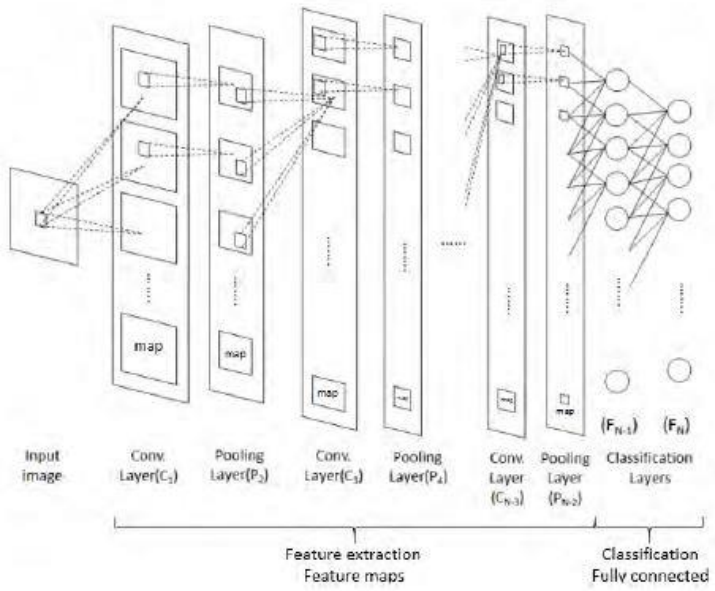


*Figura 6: Capas que conforman una CNN*

Las CNNs se conforman de dos etapas: extracción de características y clasificación. En la etapa de extracción, como su propio nombre indica, se extraen las características que poseen los datos de entrada o entrenamiento, formando los mapas de características (*feature map*s), los cuales van aumentando en número y disminuyendo en tamaño a medida que haya más capas en esta etapa. Además, esta consta de dos capas, la capa convolucional y la capa de *pooling*, las dos capas van en cascada y no se suelen separar, es más, a veces se suelen tratar las dos capas (convolucional + *pooling*) como una sola.

Al ser las características propagadas, estas siguen siendo abstraídas y combinadas para producir unas características de más alto nivel, a medida que los mapas de características se van volviendo más pequeños. Este proceso se sigue efectuando hasta tener mapas de características de muy baja resolución.

Cuando estos mapas de características han sido extraídos, una red '*fully connected*' como clasificadora se encarga de separar y clasificar cada característica. Las CNN varían, principalmente, en cómo las capas convolucionales + *pooling* son realizadas, pero, además, también hay que tener en cuenta cómo estas son entrenadas. Se va a proceder ahora a hacer una explicación más detallada de estas capas.

- **Capa convolucional:**

  Esta capa extrae las características de los datos de entrenamiento por medio de la convolución entre dichos datos y filtros "entrenables" tipo *kernels*. Las características que se suelen encontrar son bordes, esquinas y/o cruces.

  La capa realiza la operación de convolución entre, una imagen o las características extraídas, y un *kernel*, el cual se compone de neuronas y pueden ser entrenables. La salida de la red se describe de la siguiente manera.

$$y_{i,j} = f\left(\sum_i k_j * x_i + b_j\right)$$

*Ecuación 1: Salida de la red*

  Donde $k_j$ son los pesos de los que compone el *kernel* y $b_j$ el grado de libertad o *bias* que acompaña a cada *kernel* o filtro. X es la entrada y la función f(x) es la función de activación.

  El tamaño del mapa de salida es definido de la siguiente manera, en donde M es la cantidad de mapas de una manera determinada capa, K es el tamaño del *kernel* utilizado, S son los factores de desplazamiento que indican cuántos píxeles en x y en y se salta u omite entre cada convolución, teniendo en cuenta que el *kernel* siempre debe quedar dentro del mapa de características o imagen y, por último, n

es el índice de la capa presente. Esta ecuación sirve para determinar el tamaño de la capa presenta como de la capa *pooling*.

$$M_x^n = \frac{M_x^{n-1} - K_x^n}{S_x^n + 1} + 1$$

$$M_y^n = \frac{M_y^{n-1} - K_y^n}{S_y^n} + 1$$

*Ecuación 2: Definición del tamaño del mapa de salida*

- **Capa *pooling*:**

Esta capa es usada para obtener una representación que es invariante contra pequeñas translaciones y distorsiones. Dicha respuesta se logra mediante la combinación (*pooling*) de las respuestas obtenidas de la capa de convolución, consiguiendo un valor característico de las respuestas anteriores.

La salida de esta capa depende de la elección del usuario, debido a que puede decidir, entre promediar, valor máximo y *subsampling*.

| Tipo de pooling | Implementación | Descripción |
|---|---|---|
| *Max-pooling* o valor máximo | $\max_{rxr}(x_i)$ | Se saca el máximo de una región cuadrada de tamaño rxr. |
| Promedio | $\text{mean}_{rxr}(x_i)$ | Se saca el promedio de una región cuadrada de tamaño rxr. |
| *Subsampling* | $\tanh\left(\beta \sum_{r,r} \frac{x_i}{r^2} + b\right)$ | Se toma el promedio de las entradas de una región cuadrada de tamaño rxr y se multiplican por un escalar entrenable (β) y se le suma un entrenable *bias* (b) y se calcula el resultado a través de la función no linear. |

*Tabla 1: Funciones aplicables en la capa pooling*

- **Capa *fully connected*:**

Esta es la última capa y etapa de una CNN, es la encargada de clasificar o etiquetar a cada clase de características. Como se ha mencionado anteriormente, las capas anteriores han hecho que las características estén en baja resolución para ser estas quienes alimenten a una RNA totalmente conectada.

Se dice que las anteriores capas no están totalmente conectadas debido a que la respuesta de una capa depende de la capa anterior, no de toda la red.

En esta capa, al igual que la convolucional, se maneja una ecuación para la salida de las neuronas en la misma. A diferencia de la convolucional, además de su operación característica, los mapas de características previos a esta capa son unidimensionales, es decir, las coordinadas 2D son ahora omitidas. Por lo tanto, la salida de cada neurona es computada a través de la siguiente ecuación:

$$y_j = f\left(\sum_i w_{ij} x_i + b_j\right)$$

*Ecuación 3: Salida de cada neurona*

**ImageNet**

ImageNet es un conjunto de datos de imágenes organizado según la jerarquía de WordNet. Cada concepto significativo en WordNet se llama *synset.* Hay más de 100.000 sintonizadores en WordNet, la mayoría de ellos son sustantivos. En ImageNet, nuestro objetivo es proporcionar un promedio de 1000 imágenes para ilustrar cada *synset.* Las imágenes de cada concepto están controladas por la calidad y anotadas por humanos.

Está diseñada para su uso en la investigación de software de reconocimiento de objetos visuales. Más de 14 millones de URLs de imágenes han sido anotadas a mano por ImageNet para indicar qué objetos se representan, también se proporcionan recuadros delimitadores. ImageNet contiene más de 20 mil categorías ambiguas; una categoría típica, contiene varios cientos de imágenes. Las anotaciones de nivel de imagen indican la presencia o ausencia de una clase de objeto en una imagen.

**AlexNet**

AlexNet es el nombre de una red neuronal convolucional compuesta por 25 capas diferentes, es la red que en concreto se usará para el desarrollo de este proyecto. Es la primera CNN famosa, fue escrita originalmente con CUDA para funcionar con soporte GPU, que compitió en el desafío de reconocimiento visual de gran escala ImageNet (ILSVRC) en 2012. La red logró un error de 15.3% más de 10.8 puntos de porcentaje por delante del finalista. AlexNet fue diseñado por el grupo *SuperVision*. El modelo está

entrenado en más de un millón de imágenes y puede clasificar en 1000 categorías de objetos. Como resultado, el modelo ha aprendido representaciones de características enriquecidas para una amplia gama de imágenes.

| Team name | Filename | Error (5 guesses) | Description |
|---|---|---|---|
| SuperVision | test-preds-141-146.2009-131-137-145-146.2011-145f. | 0.15315 | Using extra training data from ImageNet Fall 2011 release |
| SuperVision | test-preds-131-137-145-135-145f.txt | 0.16422 | Using only supplied training data |
| ISI | pred_FVs_wLACs_weighted.txt | 0.26172 | Weighted sum of scores from each classifier with SIFT+FV, LBP+FV, GIST+FV, and CSIFT+FV, respectively. |

*Tabla 2: Posición competición ILSVRC 2012*

**GoogLeNet**

Es otra red convolucional, es una red entrenada a partir de la base de datos ImageNet, al igual que AlexNet. La diferencia principal con AlexNet, es que GoogLeNet se compone de 144 capas. El funcionamiento es igual a AlexNet, sólo que bastante mejorado. Es así, que en la competición ILSVRC de 2014, consiguió una tasa de error del 6.66%.

A continuación, se muestra una comparativa entre AlexNet y GoogLeNet al ser las redes utilizadas a lo largo del proyecto, los tiempos establecidos han sido con pruebas usando 7 categorías. Explicado más en detalle en capítulos posteriores.

| | AlexNet | GoogLeNet |
|---|---|---|
| Capas | 25 | 144 |
| Tiempo | 40 minutos | 149 minutos |
| Efectividad | Buen resultado | Da un resultado mejor |

*Tabla 3: Comparativa entre AlexNet y GoogLeNet*

**R-CNN**

Las redes neuronales convolucionales basadas en la región (R-CNN), se basan en tener como idea principal dos pasos. En primer lugar, utilizar una búsqueda selectiva, identifica un número manejable de candidatos de regiones de interés (ROI). En segundo lugar, extrae las características de cada región de forma independiente para la clasificación.

Estas redes utilizan, dentro de su funcionamiento, una medida de evaluación de nominada Intersección sobre Unión (IoU), esta medida es utilizada para medir la precisión de un detector de objetos en un conjunto de datos particular.

Este tipo de redes funcionan de la siguiente manera:

1. Pre-entrena una red CNN en tareas de clasificación, por ejemplo, usando AlexNet.
2. Proponer regiones de interés independientes de la categoría mediante búsqueda selectiva, eligiendo hasta 2000 regiones por imagen.
3. Las especies de cada región son deformadas para tener un tamaño fijo según lo requerido por la CNN.
4. Se continúa ajustando la CNN en las regiones del punto anterior para un número de clases k+1, es decir, se añade una clase adicional, siendo esta clase adicional la correspondiente al fondo.
5. Dada cada región, una propagación hacia delante de la CNN genera un vector de características.
   Las muestras positivas son regiones propuestas con un umbral de superposición IoU >= 0.3, siendo las muestras negativas irrelevantes.
6. Para reducir errores de localización, se entrena un modelo de regresión para corregir la ventana de detección en el desplazamiento del cuadro delimitador (*bounding box*) usando las características de la CNN.

Observando estos pasos, se puede deducir con facilidad, que el entrenamiento será costoso y lento, ya que algunos pasos implican mucho trabajo. Entre ellos, la ejecución selectiva para proponer regiones para cada imagen o el vector de características de CNN para da región de imagen, entre otros.

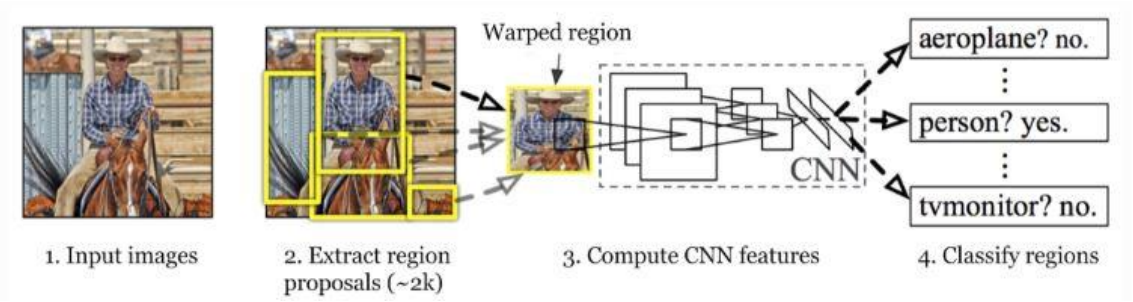


*Figura 7: Cómo funciona una RCNN*

### ***Fast* R-CNN**

Para hacer que la R-CNN sea más rápida, se mejoró el entrenamiento en 2015. Para ello, se unificó tres modelos independientes en un marco de entrenamiento conjunto y aumentando los resultados del cálculo compartido. En lugar de extraer

vectores de características de la CNN para cada región, este modelo los agrega en la CNN mientras pasa por toda la imagen y las regiones comparten esta matriz de características. Luego, se ramifica la misma matriz de características para usarla para aprender el clasificador de objetos y el regresor del cuadro delimitador. En conclusión, la compartición de cómputo acelera la R-CNN.

Los pasos a realizar en este tipo de redes son los siguientes:

1. En primer lugar, se pre-entrena una red neuronal convolucional para la clasificación de imágenes.
2. Una vez concluido ese pre-entreno, se proponen unas regiones por búsqueda selectiva.
3. Se altera la CNN pre-entrenada. En primer lugar, reemplazando la última capa de *max pooling* de la CNN pre-entrenada por una capa de agrupación ROI. Este tipo de capa genera vectores de características de longitud fija propuestas por la región. Y seguido, se reemplaza la última capa *fully connected* y la última capa de *softmax* (k clases) con una capa *fully connected* y la última capa *softmax* sobre k+1 clases.
4. Finalmente, el modelo se bifurca en dos capas de salida: La primera, un estimador de *softmax* y la segunda un modelo de regresión de los *bounding boxes*.

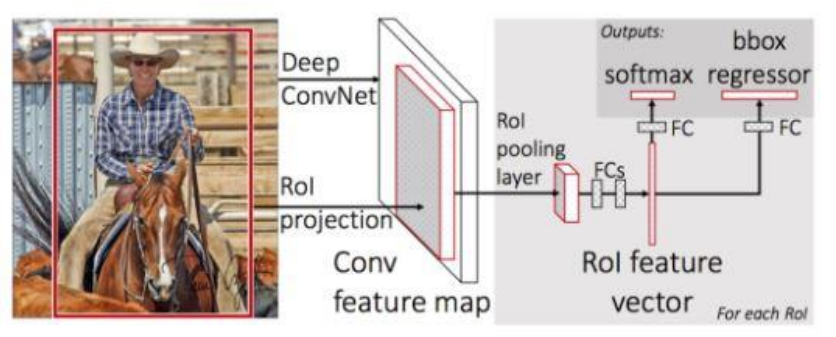


*Figura 8: Funcionamiento Fast-RCNN*

***Faster* R-CNN**

La diferencia principal entre *Fast* R-CNN y *Faster* R-CNN es que no se usa un método de propuesta de región especial para crear dichas regiones. En su lugar, se entrena una red de propuesta regiones, la cual toma los mapas de características como entrada y salidas de las propuestas de la región.

En resumen, estas tres técnicas se pueden definir de la siguiente manera:

- R-CNN: Elige la imagen completa y busca regiones de interés (alrededor de 2000 por imagen) y cada región de interés se utiliza como una imagen completa a la hora de entrenar con AlexNet, además se añade como clase adicional el fondo.

- *Fast* R-CNN: Usa la imagen original para entrenar AlexNet, y a partir del mapa de características se sacan las regiones de interés, realizando posteriormente la clasificación.
- *Faster* R-CNN: Integra la estimación de regiones de interés con posibles objetos de la red AlexNet, se añade una capa de propuesta de regiones a partir del mapa de características de AlexNet.

## 2.3. *Transfer Learning*

*Transfer Learning* consiste en reentrenar una Red Convolucional, es decir, usar el conocimiento aprendido de las tareas para las cuales hay una gran cantidad de datos etiquetados disponibles en entornos donde sólo hay pocos datos etiquetados disponibles. Crear los datos es algo que lleva bastante tiempo, por eso es muy útil aprovechar los conjuntos ya existentes.

Lo que se quiere conseguir con el *Transfer Learning* es que se asemeje lo máximo posible a la forma de aprender de los humanos, es decir, la capacidad de aprender de un gran número de experiencias y exportar el "conocimiento" a nuevos entornos.

**Etiquetado de imágenes**

En el caso de querer realizar una base de datos desde el inicio, se necesitará para ello etiquetar las imágenes. En el caso de MatLab, herramienta que se usa en este trabajo, tiene incorporada una aplicación de etiquetado *training Image Labeler.*

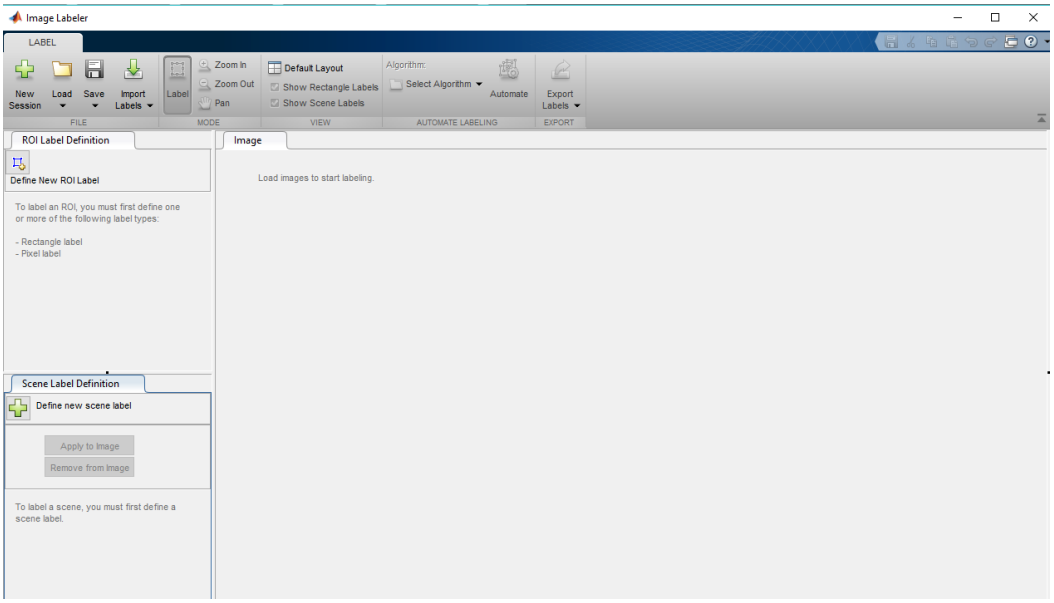


*Figura 9: Aplicación training Image Labeler.*

Consiste en ir introduciendo imágenes de lo que se quiera clasificar e ir etiquetándolas hasta conseguir una gran cantidad de datos. Para ello, se añaden todas las imágenes que se desean etiquetar, se crean todas las etiquetas que sean necesarias, se va etiquetando cada imagen con cada especie u objeto que se desea. Y, por último, para exportarlo, se guarda la sesión como tal para poder modificarla en un momento futuro y se exportan las etiquetas de las imágenes como un archivo diferente. Ambos archivos se generan como '.mat' pero el que se necesitará cargar en el script serán las etiquetas.

Otra forma de etiquetado es poder etiquetar un vídeo de forma automática usando un algoritmo interno. Para ello se necesita tener un fondo fijo, después se selecciona el objeto que se quiera etiquetar y automáticamente etiqueta en todos los *frames* siguientes que aparezca dicho objeto.

La primera forma, la manual, puede llegar a ser un trabajo tedioso y lento, pero como se demostrará en el capítulo siguiente al final llega a ser la solución más efectiva.

## 2.4. *Data Augmentation*

Es común que ocurra que no se disponga de la cantidad suficiente de datos, ya que, a mayor cantidad de datos, mayor será el rendimiento de la red. Cuando se entrena un modelo de aprendizaje automático, lo que realmente se hace es ajustar sus parámetros de manera que pueda asignar una entrada particular a una salida.

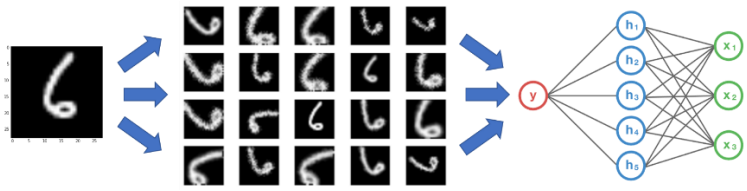


*Figura 10: Ejemplo de Data Augmentation*

Para obtener más datos, sólo es necesario hacer pequeñas modificaciones al conjunto de datos existente, estos cambios pueden ser pequeñas rotaciones o giros, cada pequeño cambio la red neuronal lo traduce a imágenes distintas. Es decir, si de una imagen se hace un giro de 90°, 180° y 270°, se llegaría a tener 4 imágenes distintas, la original y las otras que han recibido un giro.

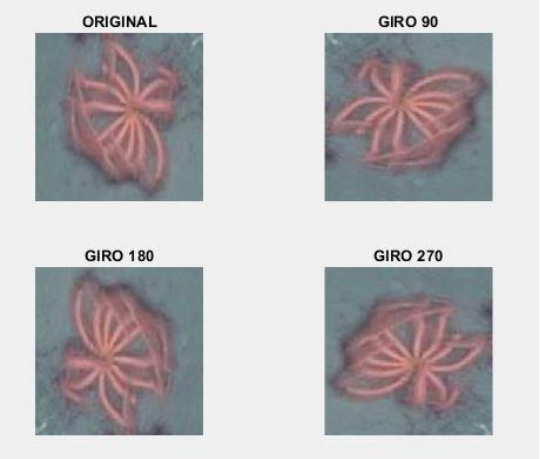


*Figura 11: Grupo imágenes data augmentation*

## 2.5. Vehículos utilizados

A lo largo del proyecto se trata sobre vídeos aportados por el IEO, estos vídeos son grabados con dos tipos de submarinos.

Uno de ellos, con el cual se comenzaron a estudiar sus vídeos, y la mayor parte de este proyecto es usando dichos vídeos. Es el Politolana, se trata de un submarino tipo trineo, el cual es arrastrado por un barco con un cable haciendo así que se vaya desplazando por el fondo, lleva incorporada una cámara de vídeo.

*Figura 12: Submarino Politolana*

El otro tipo de submarino utilizado es el denominado Lander, este, a diferencia al Politolana, es una especie de trípode el cual se queda en el mismo punto durante periodos de tiempo y va sacando fotos del mismo sitio constantemente.

*Figura 13: Submarino Lander*

# 3. Procedimiento

## 3.1. Selección de imágenes

Lo primero que se debe de tener en cualquier uso de *Deep Learning*, es una gran cantidad de imágenes, para luego poder reentrenar una red, como AlexNet o GoogLeNet, explicadas anteriormente.

En primer lugar, se obtuvo una serie de especies del registro mundial de especies marinas (WoRMS). Inicialmente, antes de tener una reunión con el equipo del IEO, se eligieron esponjas, estrellas de mar del filo *Echinodermata,* algas, corales clase *Anthozoa* del filo *Cnidaria,* peces y crustáceos. Se hizo esta selección para comprobar el funcionamiento de la red una vez reentrenada, para así poder reconocer ciertos errores futuros.

El problema que se encontró en ese momento fue que estas imágenes a veces eran bocetos o imágenes sacadas de un microscopio, y las que se podía ver la especie como tal, eran fotos hechas con una luz y un ángulo casi perfecto, algo imposible de alcanzar comparándolo con los vídeos del submarino aportados por el IEO.

En la siguiente imagen se ha elegido de forma aleatoria una imagen de cada especie con la que inicialmente se empezó a estudiar este proyecto.

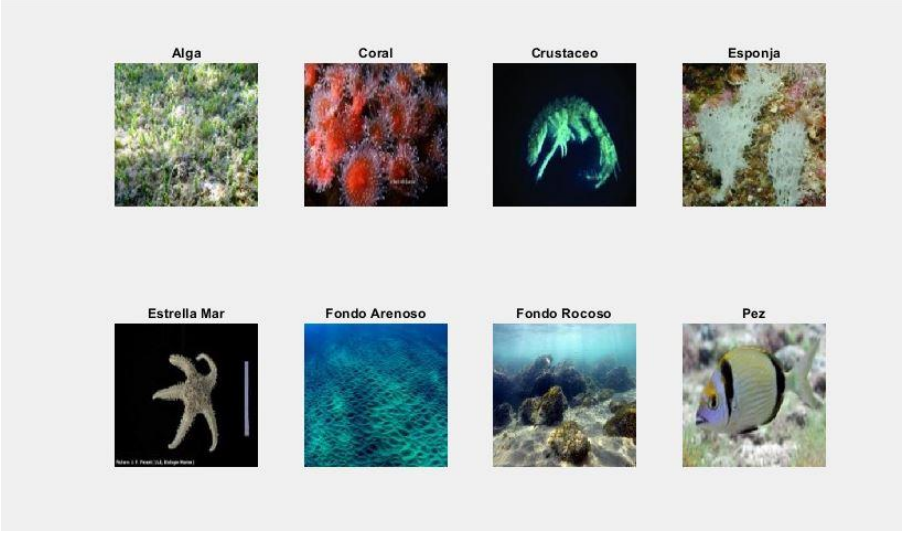


*Figura 14: Primer conjunto de especies.*

Una vez visualizado el problema anterior, se procedió a extraer las imágenes de las especies directamente de los vídeos proporcionados por el IEO. Para ello se llegó a un acuerdo en que las especies a detectar fuesen: Gorgonia *Paramuricea Placomus* (Gorgonia amarilla), Gorgonia *Callogorgia Verticilata* (Gorgonia blanca), Esponja *Geodia Barreti*, Esponja *Asconema Setubalense*, Brisíngida, erizo de cuero y esponjas en general.


*Figura 15: Segundo conjunto de especies*

Con este cambio, al ser imágenes sacadas de los propios vídeos, se deduce que la detección será mucho más fácil, ya que las tonalidades verdosas de los vídeos y a veces la falta de iluminación, podría causar una mala detección si se usaban imágenes sacadas de WorMS o de alguna otra base de datos, ya que las imágenes poseían unos colores más vivos y más claros que lo que se posee en los vídeos.

## 3.2. Redimensionado de las imágenes

Para poder reentrenar la red con dichas imágenes lo que hay que hacer es redimensionar las mismas, con una dimensión de 227x227 que es la utilizada por AlexNet. Además, como ya se ha comentado anteriormente, un problema muy común es no tener una cantidad suficiente de imágenes a la hora de reentrenar la red, para ello se utiliza el concepto de *Data Augmentation*, esto nos permite aumentar de forma considerable la cantidad de imágenes por especie.

Para realizar dicho redimensionado y las rotaciones se realizan a partir del siguiente código.

```
a=dir('*.jpg');  % Elegir todas las imágenes .jpg
                 % del directorio.

for k=1:length(a)
    i=imread(a(k).name);

    imag=imresize(i, [227 227]);  % Redimensionar las imágenes a 227x227

    imag2=imrotate(imag,90);    % Rotar las imágenes 90°
    imag3=imrotate(imag,180);   % Rotar las imágenes 180°
    imag4=imrotate(imag,270);   % Rotar las imágenes 270°

    imwrite(imag,sprintf('Esponja%i.jpg',k));    % Guardar las imágenes
    imwrite(imag2,sprintf('Esponja90_%i.jpg',k));
    imwrite(imag3,sprintf('Esponja180_%i.jpg',k));
    imwrite(imag4,sprintf('Esponja270_%i.jpg',k));
end
```

*Figura 16: Código para redimensionar y rotar.*

## 3.3. Reentrenar las redes

**AlexNet**

Una vez realizado todo el redimensionado de las imágenes y las rotaciones de estas, se intentó proceder a la detección directamente. Para ello se eligió AlexNet para reentrenar la red. Aproximadamente, para reentrenar la red se necesitaba alrededor de 30-40 minutos, variable en función de la cantidad de categorías que se tenga, un tiempo más o menos razonable sobre todo a la hora de realizar cambios y tener la necesidad de volver a reentrenarlo.

Para realizar esto, en primer lugar, se carga la red AlexNet, se especifica el número de categorías que se tienen, en este caso eran 7 y por último se reentrena la red generando un archivo.mat.

```
Training on single CPU.
Initializing image normalization.
|=========================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |     Loss     |      Rate       |
|=========================================================================================|
|       1 |           1 |       00:00:07 |       18.75% |       2.8326 |          0.0010 |
|       3 |          50 |       00:05:18 |       96.88% |       0.1030 |          0.0010 |
|       5 |         100 |       00:10:31 |      100.00% |       0.0223 |          0.0010 |
|       8 |         150 |       00:15:57 |      100.00% |       0.0096 |          0.0010 |
|      10 |         200 |       00:20:44 |      100.00% |       0.0123 |          0.0010 |
|      13 |         250 |       00:25:38 |      100.00% |       0.0038 |          0.0010 |
|      15 |         300 |       00:30:40 |      100.00% |       0.0038 |          0.0010 |
|      18 |         350 |       00:35:50 |      100.00% |       0.0012 |          0.0010 |
|      20 |         400 |       00:40:46 |      100.00% |       0.0024 |          0.0010 |
|=========================================================================================|
```

*Figura 17: Tiempo en ejecutar AlexNet*

Una vez reentrenada la red, se intentó clasificar directamente el vídeo, para ello había que cargar el archivo.mat anterior, y analizar el vídeo *frame* a *frame*, es decir, usando la función "*classify*" de Matlab, que consiste en comparar la red entrenada con cada *frame* del vídeo. Para ello también habría que redimensionar el vídeo a 227x227, y es donde se empezó a ver un nuevo problema. Al redimensionar el vídeo y hacer la imagen tan pequeña, no se podía observar nada y por tanto una detección relativamente buena, era casi imposible.

*Figura 18: Tamaño imagen 227x227*

**GoogLeNet**

En un primer momento se pensó que no clasificaba bien por culpa de la red usada, por lo que se intentó realizar el mismo procedimiento, pero usando GoogLeNet. Como se ha comentado en los fundamentos teóricos, GoogLeNet posee un mayor número de capas, 144 frente a las 25 de AlexNet, y está demostrado que tiene una mayor capacidad de detección. Se volvió a realizar el procedimiento de redimensionado de imágenes, ya que GoogLeNet utiliza tamaños de 224x224.

Un cambio muy notable es el tiempo de ejecución, mientras que AlexNet tardaba alrededor de 30-40 minutos, GoogLeNet ha llegado a tardar 149 minutos sin usar para nada más la CPU. Era un aumento de tiempo muy considerable, por lo que a la hora de hacer cambios llegaba a ser tedioso.

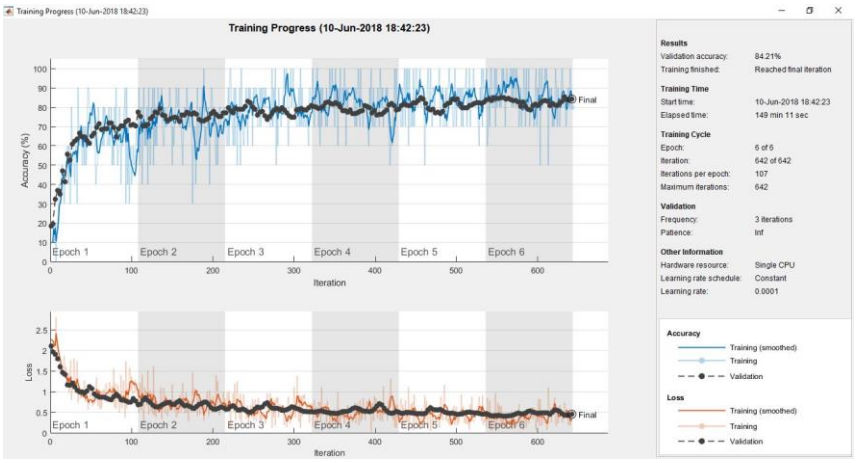


*Figura 19: Tiempo de ejecución GoogLeNet*

Pero realmente con GoogLeNet se tenía el mismo problema anterior, se redimensiona el video a 224x224 y sigue siendo un tamaño muy pequeño donde no se podría diferenciar nada por lo tanto la detección no era viable.

*Figura 20: Tamaño imagen 224x224*

Como se puede ver en el caso de AlexNet y de GoogLeNet, ambas imágenes no se pueden diferenciar especies, siendo esta imagen la imagen original donde se pueden ver una gran agrupación de brisíngidas.

*Figura 21: Imagen original*

La solución que se planteó fue que se pudiesen recuadrar las especies y así en cada cuadro se marque el nombre de la especie que corresponda. Estos cuadros son llamados en Matlab como '*bounding box*' o cuadro delimitador.

## 3.4. Reconocimiento de especies cámara estacionaria

Como se ha comentado al final del apartado anterior, se planteó la idea de crear '*bounding boxes*' o rectángulos en regiones de interés (ROI). Un rectángulo en regiones de interés corresponde a una región que se tiene un interés particular de forma rectangular. Estas etiquetas contienen dos componentes, el nombre de la etiqueta, por ejemplo "Brisíngida", y la región que ocupa dicha especie. Viendo los problemas anteriores, lo que se necesitaba era delimitar mejor las detecciones, por lo que se necesitaba que se encuadrasen las especies. Para ello, se utilizan los '*bounding boxes*'.

Para este proceso, se necesitaba reentrenar una red, AlexNet en este caso. Como ya se ha comentado, para reentrenar la red se hizo una selección de imágenes de los vídeos obteniendo, en conjunto de todas las especies, más de dos mil imágenes.

Se procedió, por tanto, a trabajar en esa idea: introducir un vídeo y a medida que avance el vídeo vayan apareciendo rectángulos alrededor de las especies. Pero se volvió a plantear otro problema, la cámara que graba los vídeos es móvil y la gran mayoría de los estudios realizados hasta ahora de detección de objetos son con cámaras fijas, por ejemplo, una cámara de tráfico.

Por lo tanto, el IEO nos ofreció un nuevo tipo de vídeos: con una cámara de un submarino diferente, el Lander, realiza una foto cada 30 segundos al mismo lugar. Es decir, se trata de una cámara estacionaria, lo necesario para esta idea.

Para este tipo de vídeos Matlab trabaja de la siguiente manera. Con la ayuda de *ForegroundDetector* se compara un fotograma de vídeo a un modelo de fondo para determinar si los pixeles individuales son parte del fondo o no. Esto calcula una máscara de primer plano.

```
detector = vision.ForegroundDetector(...
    'NumTrainingFrames', 5, ...
    'InitialVariance',10*10);

blob = vision.BlobAnalysis(...
    'CentroidOutputPort', false, 'AreaOutputPort', false, ...
    'BoundingBoxOutputPort', true, ...
    'MinimumBlobAreaSource', 'Property', 'MinimumBlobArea', 500);

shapeInserter=vision.ShapeInserter('BorderColor', 'White');
framenum=1;
```

*Figura 22: Generar detector y ROI*

En concreto, usar vision.*ForegroundDetector* calcula y devuelve una máscara de primer plano utilizando un modelo de mezcla Gaussiana. Además, se han incluido unas propiedades adicionales, *NumTrainingFrames* establece el número de fotogramas de video iniciales, debe ser un número entero. Igualmente, se incluye la propiedad *InitialVariance* la cual indica la varianza inicial del modelo de mezcla, debe ser un número escalar. Asimismo, se definen las regiones de interés (ROI), donde con las características establecidas se indican que lo que se quiere devolver son las coordenadas de salida de los cuadros delimitadores, también se detalla el área mínima en píxeles de los ROI.

Una vez establecidas todas estas características se realiza la detección del vídeo. Para el entrenamiento de la red se ha utilizado uno de los primeros entrenamientos realizados, el cual poseía la especie “pez”, como se ha comentado anteriormente este conjunto de datos se había obtenido a partir del registro mundial de especies marinas (*WoRMS*).

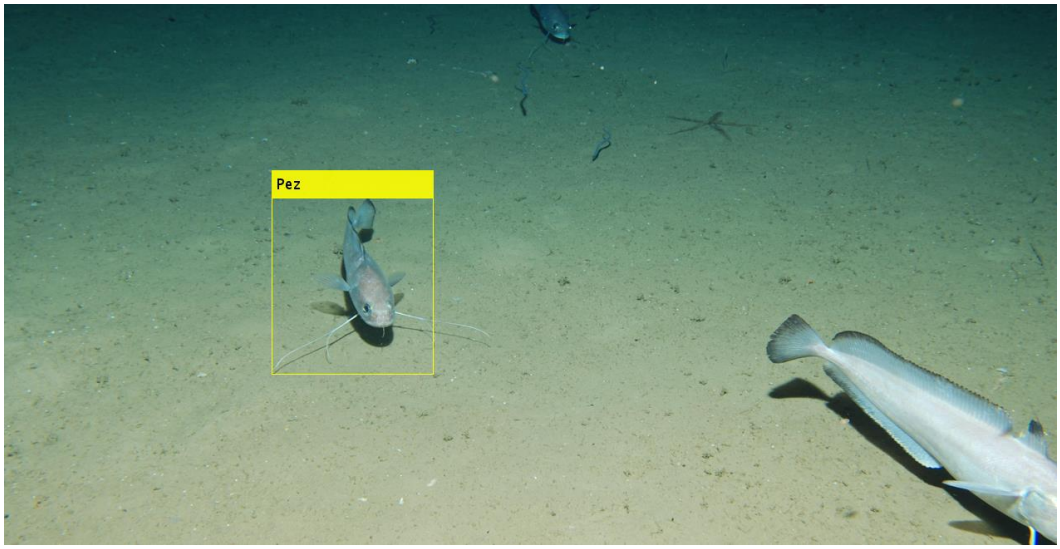


*Figura 23: Detección de especies con cámara fija*

Si se usa todo este desarrollo para el estudio con una cámara móvil, es decir, no estacionaria, se observará como la detección no es correcta. A parte de no detectar todas las especies que aparecen, tampoco detecta de forma precisa cada especie. Es decir, a una brisíngida le puede poner la etiqueta de erizo, como se muestra en la siguiente imagen.

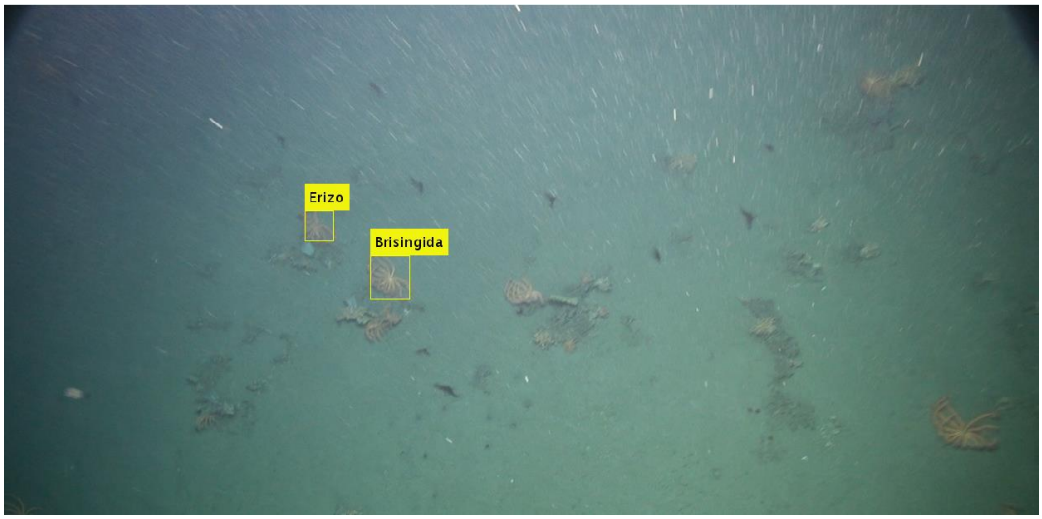


*Figura 24: Detección eliminando el fondo, cámara móvil.*

## 3.5. Reconocimiento de especies cámara no estacionaria

En este caso se plantea la siguiente solución: entrenar un detector de tipo *faster* RCNN. Para ello, en vez de utilizar las imágenes sacadas de los vídeos como anteriormente, se utiliza la aplicación *Image Labeler* de Matlab. Es decir, realizar un nuevo etiquetado de las imágenes para así poder guardarlas en un archivo.mat.

Como *Image Labeler* funciona etiquetando imágenes en vez de vídeos, el primer paso será dividir todos los vídeos en sus respectivos *frames*, o al menos en una cantidad considerable de *frames*, para así luego proceder a dicho etiquetado.

Para realizar esta extracción de *frames*, se tiene en cuenta el uso de '*CurrentTime*', lo que facilita que a la hora de la lectura del vídeo empiece a leerlo en los segundos indicados en el '*CurrentTime*'. Ya que, en varias ocasiones, en un vídeo de 17 minutos no salen las especies constantemente, por ello durante la visualización de los vídeos se va anotando los tiempos donde van apareciendo, porque encontrarlas entre todos los *frames* que tiene un vídeo sería algo muy lento.

```
vid=VideoReader('C:\Users\Celia\Desktop\TFG_Deep Learning\TransferLearning\VIDEOS\SPONGES\S0617_TV08_00062.MTS',...
                'CurrentTime',746);

Folder = 'C:\Users\Celia\Desktop\TFG_Deep Learning\RCNN\Frames';
for iFrame = 1:500
   frames = readFrame(vid);
   imwrite(frames, fullfile(Folder, sprintf('AS3%06d.jpg', iFrame)));
end
FileList = dir(fullfile(Folder, '*.jpg'));
for iFile = 1:length(FileList)
   aFile = fullfile(Folder, FileList(iFile).name);
   img   = imread(aFile);
end
```

*Figura 25: Código para extraer frames*

Como se ha comentado en el capítulo anterior, esto puede llegar a ser un trabajo lento y tedioso, pero cuantas más imágenes estén etiquetadas mejor será el entrenamiento futuro. Las etiquetas realizadas serán como se muestra a continuación, en cada imagen si aparece más de una especie habrá que seleccionar todo lo que sea relevante para el reconocimiento posterior.

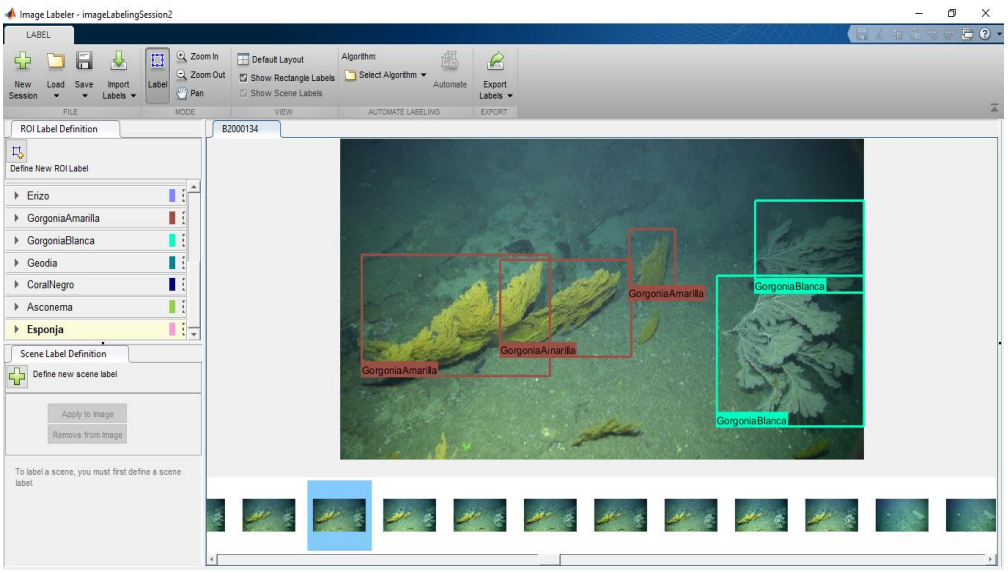


*Figura 26: Etiquetado de especies*

Una vez realizado todo el etiquetado, se exportan las etiquetas como un archivo.mat para así poder cargarlas en Matlab y realizar así el entrenamiento del detector. Así, en nuestro archivo posee las siguientes etiquetas:

| Name | Type |
|---|---|
| 'Brisingidas' | Rectangle |
| 'Erizo' | Rectangle |
| 'GorgoniaAmarilla' | Rectangle |
| 'GorgoniaBlanca' | Rectangle |
| 'Geodia' | Rectangle |
| 'CoralNegro' | Rectangle |
| 'Asconema' | Rectangle |
| 'Esponja' | Rectangle |

*Tabla 4: Tipo de etiquetas*

Para ello, con la ayuda de la función '*objectDetectorTrainingData*' se crea así una variable ('*trainingData*') que contiene en forma de tabla todos los datos extraídos del archivo anterior, el que contiene todas las etiquetas.

```
load('EspMar2.mat'); %373 imagenes a partir de varios videos del politolana
trainingData = objectDetectorTrainingData(gTruth,'SamplingFactor',1);
```

*Figura 27: realizar Training Data*

Este entrenamiento consiste en cuatro etapas, cada etapa se divide en 10 *epochs* cada una, es un entrenamiento que implica bastante tiempo. Cada etapa tiene sus

```
% Options for step 1.
optionsStage1 = trainingOptions('sgdm', ...
    'MaxEpochs', 10, ...
    'MiniBatchSize', 256, ...
    'InitialLearnRate', 1e-3, ...
    'CheckpointPath', tempdir);
% Options for step 2.
optionsStage2 = trainingOptions('sgdm', ...
    'MaxEpochs', 10, ...
    'MiniBatchSize', 128, ...
    'InitialLearnRate', 1e-3, ...
    'CheckpointPath', tempdir);
% Options for step 3.
optionsStage3 = trainingOptions('sgdm', ...
    'MaxEpochs', 10, ...
    'MiniBatchSize', 256, ...
    'InitialLearnRate', 1e-3, ...
    'CheckpointPath', tempdir);
% Options for step 4.
optionsStage4 = trainingOptions('sgdm', ...
    'MaxEpochs', 10, ...
    'MiniBatchSize', 128, ...
    'InitialLearnRate', 1e-3, ...
    'CheckpointPath', tempdir);
```

correspondientes características, establecidas por la función '*trainingOptions*' y guardadas en la variable '*optionStageX*', siendo X el número de la etapa. Una vez entrenada cada etapa, se agrupan todas las '*optionStage*' en una un array llamado '*options*'.

*Figura 28: Características de cada entrenamiento*

```
options = [
    optionsStage1
    optionsStage2
    optionsStage3
    optionsStage4
    ];
```

*Figura 29: Array con todas las opciones de cada etapa*

Como se ha comentado, este entrenamiento es lento. En una primera prueba, con el etiquetado de tres especies y teniendo unas 100 imágenes un ordenador portátil tardaba alrededor de 7 horas, en cambio, se usó un ordenador proporcionado por el departamento de TEISA, el cual tardaba en realizar el mismo entrenamiento 40 minutos. Por eso mismo, para el entrenamiento definitivo, se utilizó el ordenador del departamento para ahorrar tiempo. Aun así, tardó en ejecutarlo 112 minutos.

```
Finished training Faster R-CNN object detector.

Elapsed time is 6723.863115 seconds.
```

*Figura 30: Tiempo en realizar el entrenamiento*

A la hora de entrenar el detector usando *faster* R-CNN, tendrá como elementos de entrada la tabla '*trainingData*', la propia red, en este caso AlexNet, y el array *options*. Con todo ello se forma el detector, para así posteriormente compararlo con los vídeos del submarino.

```
detector = trainFasterRCNNObjectDetector(trainingData, net, options, ...
    'NegativeOverlapRange', [0 0.3], ...
    'PositiveOverlapRange', [0.6 1], ...
    'BoxPyramidScale', 1.2);
```

*Figura 31: Crear el detector*

Para detectar dentro del vídeo, únicamente habrá que leer el vídeo, y *frame* a *frame* realizará lo siguiente.

En primer lugar, se comprueba si existe un *frame* y así poder leerlo, de esta manera se va a utilizar la función '*detect*' cuya misión es detectar objetos en la imagen, para ello como variables de entrada están el detector entrenado y el *frame* de cada momento, y devolverá como salida '*bbox*', '*score'* y '*label'*. Siendo la variable '*bbox*' correspondiente a los *bounding boxes*, que como ya se ha explicado son los cuadros delimitadores. La variable '*score'* aporta datos sobre la seguridad con la que detecta una especie, y por último, la variable '*label'* donde se guarda el nombre de dichas especies.

En algunos casos ocurre, que en un *frame* no detecta nada, por lo tanto, las variables anteriormente explicadas, *bbox*, *label* y *score*, están vacías. Para ello se utiliza la función '*isempty*', la cual devuelve un uno si la variable está vacía, o un cero si la variable no está vacía.

```
videoReader = VideoReader('E0717_TV18_019.MTS');
videoPlayer = vision.DeployableVideoPlayer;

while hasFrame(videoReader)
    for n=1:30 % se salta N frames
      frame = readFrame(videoReader);
     end
      outputImage=frame;
      [bbox,score,label] = detect(detector,frame);
      comp=isempty(label);

      if comp==1
          outputImage=frame;
      end

      if comp==0
       annotation=cell(size(bbox,1),1);

        for iii = 1:size(bbox,1)
        annotation{iii} = sprintf('%s', label(iii));
        end
        outputImage = insertObjectAnnotation(frame,'rectangle',bbox,annotation,'TextBoxOpacity',0.9,'FontSize',18);
      end
step(videoPlayer,outputImage)
end
```

*Figura 32: Detección de un vídeo.*

Una vez realizado todo, se comprueba si la detección es la esperada o no. A continuación se mostrará la eficacia de este entrenamiento.

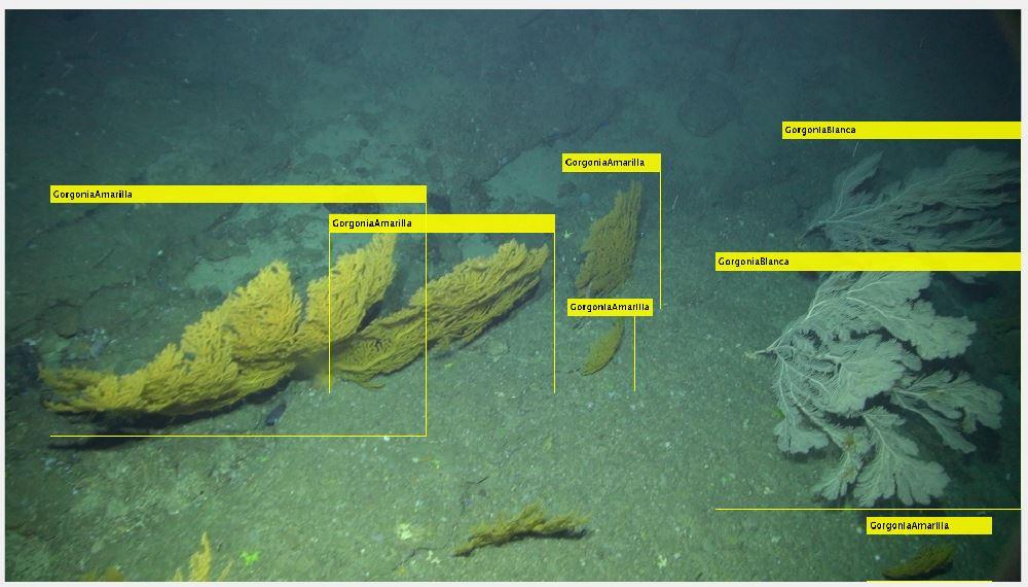


*Figura 33: Resultado de la detección de Gorgonias Amarillas y Blancas*

Como se puede observar, detecta todas las especies correctamente, sin confundir ninguna especie con otra y detectando prácticamente todas las que aparecen en ese *frame*.

Por último, se añadió un contador de etiquetas, es decir, a lo largo de un vídeo a estudiar, se pudiese ir contando las veces que se detectaba una especie en cada *frame*, para así llevar un control más exacto.

```
'GorgoniaAmarilla = 0
 Brisingidas = 163
 Erizo = 46
 GorgoniaBlanca = 0
 CoralNegro = 0
 Esponja = 0
 Geodia = 0
 Asconema = 0'
```

*Figura 34: Contador de especies*

# 4. Conclusiones y líneas futuras

Como conclusión, una vez seguidas todas las pautas de este proyecto, y viendo todos los problemas que se pueden llegar a obtener en la realización de este, se puede afirmar que se ha llegado a una solución buena, en la que se ha conseguido que, al introducir una serie de vídeos grabados en fondos de alta profundidad por un submarino, se lleguen a detectar unas especies concretas de alto interés para el Instituto Español de Oceanografía.

Estos resultados son bastante buenos, aunque hay algunas especies que no llega a detectarlas. El proceso de visualización del vídeo es algo lento, por lo que no se podría ver a tiempo real, para que llegase a verse en tiempo real en procesar un *frame* no debería de tardar más de 1/30 de segundo.

Este proyecto, puede servir en un futuro, para poder realizar un estudio de densidad de algunas especies en ciertos periodos de tiempo. Esto servirá para proteger especies que puedan llegar a estar en peligro de extinción, ya que al llevar un contador de las especies etiquetadas se podrá llevar un control más exacto de la cantidad de especies que hay y ahorrando una gran cantidad de tiempo a aquellas personas que realicen un estudio de densidad de especies dentro del IEO.

Por lo tanto, uno de los estudios posibles a realizar es, con la investigación aquí aportada poder entrenar con más imágenes de más especies para así tener una información más amplia para el seguimiento medioambiental. Como se ha comentado anteriormente, aunque la detección sea precisa, a veces esa detección no se realiza correctamente, por lo que un estudio interesante sería analizar el porqué esos ejemplares no son detectados.

Un tema destacable en casi cualquier ámbito es la velocidad con la que podemos llegar a un resultado final aceptable. En este proyecto se han comparado tres tipos de R-CNN siendo cada cual más rápida que la anterior, pero también existen otras técnicas que podrían llegar a ser más rápidas.

Por último, al principio del trabajo se hizo una comparativa entre dos redes neuronales, AlexNet y GoogLeNet, se observaba cómo AlexNet trabajaba más rápido pero también podía a llegar a ser menos eficiente, por lo tanto, un estudio útil sería ver cómo reacciona este proyecto usando otro tipo de red, si ese aumento en tiempo llega a ser notable y beneficioso.

# 5. Bibliografía


[1] Definición Deep Learning:

https://es.mathworks.com/discovery/deep-learning.html

[2] Definición Transfer Learning:

https://www.datacamp.com/community/tutorials/transfer-learning

[3] Definición Data Augmentation:

https://medium.com/nanonets/how-to-use-deep-learning-when-you-have-limited-data-part-2-data-augmentation-c26971dc8ced

[4] Definición fast y faster RCNN:

https://jhui.github.io/2017/03/15/Fast-R-CNN-and-Faster-R-CNN/

[5] Funciones Matlab:

https://es.mathworks.com/

[6] GIRSHICK, Ross. Fast r-cnn. En Proceedings of the IEEE international conference on computer vision. 2015. p. 1440-1448.

[7] REN, Shaoqing, et al. Faster r-cnn: Towards real-time object detection with region proposal networks. En Advances in neural information processing systems. 2015. p. 91-99.

[8] GLEN JHAN PIERRE RESTREPO ARTEAGA. CNN. En Aplicación del aprendizaje profundo ("deep learning") al procesamiento de señales digitales. 2015. P 39-48

# ANEXOS

```
% Entrenamiento de una red tipo Faster R-CNN

load('EspeciesMarinas.mat'
trainingData = objectDetectorTrainingData(gTruth,'SamplingFactor',1);
%% Look at structure of pre-trained network
net = alexnet;
% Notice the last layer performs 1000 object classification
layers = net.Layers %#ok

%% Or even inspect the types of object this network was trained on
layers(25).ClassNames % #ok we want to display this in command window

num_objects = size(trainingData,2)-1; % número de categorías
layers(23) = fullyConnectedLayer(num_objects, 'Name','fc8');
layers(25) = classificationLayer('Name','myNewClassifier')

layers(end-2).WeightLearnRateFactor = 10;
layers(end-2).BiasLearnRateFactor = 20;

% Options for step 1.
optionsStage1 = trainingOptions('sgdm', ...
    'MaxEpochs', 10, ...
    'MiniBatchSize', 256, ...
    'InitialLearnRate', 1e-3, ...
    'CheckpointPath', tempdir);

% Options for step 2.
optionsStage2 = trainingOptions('sgdm', ...
    'MaxEpochs', 10, ...
    'MiniBatchSize', 128, ...
    'InitialLearnRate', 1e-3, ...
    'CheckpointPath', tempdir);

% Options for step 3.
optionsStage3 = trainingOptions('sgdm', ...
    'MaxEpochs', 10, ...
    'MiniBatchSize', 256, ...
    'InitialLearnRate', 1e-3, ...
    'CheckpointPath', tempdir);

% Options for step 4.
optionsStage4 = trainingOptions('sgdm', ...
    'MaxEpochs', 10, ...
    'MiniBatchSize', 128, ...
    'InitialLearnRate', 1e-3, ...
    'CheckpointPath', tempdir);

options = [
    optionsStage1
    optionsStage2
    optionsStage3
    optionsStage4
    ];
```

```
% inicialización
    rng(0);
    tic;
    % Train Faster R-CNN detector. Select a BoxPyramidScale of 1.2 to
allow
    % for finer resolution for multiscale object detection.
    detector = trainFasterRCNNObjectDetector(trainingData, net,
options, ...
        'NegativeOverlapRange', [0 0.3], ... % era 0.3
        'PositiveOverlapRange', [0.6 1], ...  % era 0.5
        'BoxPyramidScale', 1.2);
    toc;
```

---

```
%% Prueba del detector con una imagen
% Read test image

frame = imread('EspeciesMarinas\001434.jpg');

% Detect stop signs
[bbox,score,label] = detect(detector,frame);

annotation=cell(size(bbox,1),1);
for iii = 1:size(bbox,1)
%
            annotation{iii} = sprintf('%s ', label(iii));
end
            outputImage =
insertObjectAnnotation(frame,'rectangle',bbox,annotation,'TextBoxOpaci
ty',0.9,'FontSize',18);
%end
figure(7);
imshow(outputImage)
```

---

```
%% Prueba del detector con un vídeo

grabarVideo = 1;

videofile =
'G:\Proyectos\otros\DeepLearning4DeepBichos\videos1\S0617_TV02_00030_t
rozo2';
videoReader = VideoReader([videofile '.mp4']);
videoPlayer = vision.DeployableVideoPlayer;
if(grabarVideo)
    outputVideo = VideoWriter([videofile '_etiquetado.avi']);
    outputVideo.FrameRate = videoReader.FrameRate;
    open(outputVideo)
end

while hasFrame(videoReader)
    for n=1:1 % se salta N frames
        frame = readFrame(videoReader);
    end
    %frame = imresize(frame, [720 1280]);
```

```
    % Detect stop signs
    [bbox,score,label] = detect(detector,frame);

    annotation=cell(size(bbox,1),1);
    for iii = 1:size(bbox,1)
        if label(iii)=='Brisingidas'
            Brisingida=Brisingida+1;

        else if label(iii)=='GorgoniaAmarilla'
                GorgoniaAmarilla=GorgoniaAmarilla+1;

            else if label(iii)=='GorgoniaBlanca'
                    GorgoniaBlanca=GorgoniaBlanca+1;

                else if label(iii)=='Erizo'
                        Erizo=Erizo+1;

                    else if label(iii)=='Geodia'
                            Geodia=Geodia+1;

                        else if label(iii)=='Asconema'
                                Asconema=Asconema+1;

                            else if label(iii)=='CoralNegro'
                                    CoralNegro=CoralNegro+1;

                                else if label(iii)=='Esponja'
                                        Esponja=Esponja+1;
                                end
                            end
                        end
                    end
                end
            end
        end
        annotation{iii} = sprintf('%s (%1.3f)', label(iii),
score(iii));
    end
    if (~isempty(annotation))
        outputImage =
insertObjectAnnotation(frame,'rectangle',bbox,annotation,'TextBoxOpaci
ty',0.9,'FontSize',18);
    else
        outputImage = frame;
    end
    step(videoPlayer,outputImage)
    if(grabarVideo)
        writeVideo(outputVideo,outputImage)
    end
end

if(grabarVideo)
    close(outputVideo);
end
```