ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y SISTEMAS DE TELECOMUNICACIÓN

PROYECTO FIN DE GRADO

TÍTULO: Sistema de conteo de peces en piscifactorías mediante redes neuronales

AUTOR: María Castillo Moral

TITULACIÓN: Ingeniería Telemática

TUTOR: Juana María Gutiérrez Arriola

DEPARTAMENTO: Departamento de Telemática y Electrónica (DTE)

Firmado digitalmente por GUTIERREZ ARRIOLA, JUANA MARIA (FIRMA)
Fecha: 2022.07.05 09:49:52 +02'00'

VºBº

Miembros del Tribunal Calificador:

PRESIDENTE: Alejandro García

TUTOR: Juana María Gutiérrez Arriola

SECRETARIO: Nicolás Sáenz Lechón

Fecha de lectura: 20 de julio 2022

Calificación:

El Secretario,

Agradecimientos

Gracias a mis padres y a mi hermano por animarme a dar lo mejor de mí en estos años de carrera.

Gracias a Juana por su apoyo y disponibilidad.

Gracias a Luka, Javi y Alberto por compartir conmigo la pasión por el mundo de las Telecomunicaciones.

Y gracias a "La Empresa" por hacerme ver que la universidad es mucho más que un sitio para seguir formándose.

Resumen

La estimación de la biomasa es un tema de gran relevancia en el área de la acuicultura. La estimación de forma precisa del número y tamaño de los peces presentes en un tanque es una información sumamente valiosa para las piscifactorías.

Existen gran cantidad de trabajos en los que se han propuesto sistemas capaces de resolver esta necesidad mediante métodos muy distintos como la emisión de ondas acústicas, el uso de sensores electrónicos o el tratamiento digital de imágenes.

En este último campo existen distintas técnicas que se pueden utilizar para extraer información contenida en imágenes, siendo uno de los más comunes la utilización de redes neuronales convolucionales**.** Esto se debe a que este tipo de algoritmos ha demostrado ser particularmente útil en tareas que involucran la extracción de información de datos visuales.

En este proyecto, por tanto, se propone la utilización de un sistema basado en una red neuronal convolucional Mask R-CNN con una arquitectura ResNeXt para realizar un conteo de los peces contenidos en una imagen dada de un tanque con rodaballos en etapa larvaria. Este conteo se lleva a cabo mediante un proceso de detección de objetos en el que se consideran dos clases: "rodaballo aislado" y "grupo de rodaballo", de tal forma que se tengan en cuenta los rodaballos que puedan estar parcialmente solapados.

Para la elaboración de este proyecto se ha realizado una labor de investigación en torno al funcionamiento general de las redes neuronales, el funcionamiento específico de las redes neuronales convolucionales y algunos algoritmos y arquitecturas ampliamente usadas en este tipo de redes.

Además, para el desarrollo del sistema en cuestión se ha llevado a cabo, en primer lugar, el etiquetado de un conjunto de imágenes de rodaballos contenidos en un tanque de piscifactoría. Después, se han formateado los datos obtenidos en el etiquetado al formato COCO JSON debido a que este es el formato de datos que entiende la red neuronal. Por último, se han entrenado seis modelos distintos de red neuronal en los que se han ido variando el *learning rate* y el número máximo de iteraciones y se han analizado los resultados obtenidos.

En este análisis realizado se puede ver que el mejor modelo (con un *learning rate* de 0.02 y un número máximo de iteraciones de 5500), aunque en ocasiones detecte los grupos de rodaballos como rodaballos aislados, presenta unos resultados satisfactorios, llegando a tener resultados muy buenos especialmente en la clase "rodaballo aislado" donde se han obtenido tasas de precisión y *recall* de hasta el 90%.

Abstract

Fish biomass estimation is one of the most relevant practices in aquaculture. The accurate estimation of the number and size of fish in a tank is an extremely valuable information for fish farms.

That is why multiples systems have been developed to solve this problem. Each of these systems uses different methods, for example, image processing, acoustic wave emissions or electronic sensors.

In the field of digital image processing different techniques such as, edge detection, have been used in order to obtain the objects contained in an image.

However, this field has been revolutionized by the appearance of convolutional neural networks. This type of algorithm has proven to be particularly useful in tasks involving the extraction of information from visual data such as images or videos.

Because of this, in this project the use of a system based on a Mask R-CNN convolutional neural network with a ResNeXt architecture to count the fish contained in a given image of a tank with turbot in larval stage is proposed. This count is carried out by an object detection process in which two classes are considered: "isolated turbot" and "turbot group", so that partially overlapped turbots are also taken into account.

For the development of this project, some research work about the general operation of neural networks, the specific way of functioning of convolutional neural networks and some algorithms and architectures widely used in this type of networks has been carried out.

In addition, for the development of this system a set of images of turbot contained in fish farm tanks was labeled. The data obtained in this labeling has been formatted to COCO JSON format as this is the data format understood by the neural network. Finally, the results obtained by six different neural network models (in which the learning rate and the maximum number of iterations were varied) have been analyzed.

In this analysis it can be seen that the best model (with a learning rate of 0.02 and a maximum number of iterations of 5500), even if sometimes detects groups of turbot as isolated turbot, presents satisfactory outcomes, with very good results especially in the "isolated turbot" class, were values of precision and recall rates of up to 90%.

Índice de contenidos

Índice de imágenes

Índice de tablas

Lista de acrónimos

COCO	Common Objects in Context
CNN	Convolutional Neural Network
Fast R-CNN	Fast Region based Convolucional Neural Network
Faster R-CNN	Faster Region based Convolucional Neural Network
FN	False negative
FP	False positive
JSON	JavaScript Object Notation
LR	Learning Rate
Mask R-CNN	Mask Region based Convolucional Neural Network
R-CNN	Region based Convolucional Neural Network
ResNet	Residual Network
TN	True negative
TP	True positive
VGG	Visual Geometry Group

1. Introducción

La estimación de la biomasa de los peces es una de las prácticas más comunes e importantes que se llevan a cabo en la acuicultura y consiste en determinar el número de peces y su tamaño. La obtención regular de información sobre la biomasa de los peces se ha identificado como una necesidad urgente para poder optimizar la alimentación diaria, controlar la densidad de población y, en última instancia, determinar el momento óptimo para la cosecha [1].

Actualmente, la obtención aproximada de la población de peces en una piscifactoría se suele realizar mediante procedimientos que suelen consumir mucho tiempo, ser laboriosos, y que pueden ser invasivos y poco precisos. Además, estos procedimientos suelen requerir de intervención humana, como ocurre, por ejemplo, con la toma de muestras o la aproximación visual [1, 2].

Esta necesidad de intervención humana en el proceso se debe, principalmente, a que la adquisición de datos a partir de ciertos tipos de información supone un gran reto para los ordenadores. La información visual, es decir, las imágenes y los vídeos, es un ejemplo de esto, ya que, para los computadores una imagen no es otra cosa que una secuencia de bytes. Esto implica que determinar patrones que indiquen donde se encuentran determinados tipos de objetos puede resultar una tarea muy compleja.

Debido a esto, a lo largo de tiempo se han ido desarrollando distintas técnicas para trabajar con imágenes, como la detección de bordes mediante la identificación de las zonas con mayor variación de luminosidad y color.

Sin embargo, en 1989, el informático Yan LeCun creó la primera red neuronal (LeNet) basándose en cómo el córtex visual de los animales y los humanos procesa la información visual que recibe. Gracias, en gran parte, a estas redes neuronales, se ha podido desarrollar un campo de la inteligencia artificial, conocido como *computer visión*. Este campo se encarga de extraer datos contenidos en imágenes y vídeos y trabaja con tareas como la clasificación de imágenes (que permite detectar si una imagen contiene un tipo de objeto concreto) y la detección de objetos (que permite identificar objetos de interés presentes en distintas imágenes).

Por este motivo, en este proyecto se ha decidido desarrollar un sistema que sea capaz de realizar un conteo de los rodaballos presentes en tanques de piscifactoría mediante el uso de una red neuronal convolucional. De esta manera, este sistema podrá monitorizar el número de ejemplares presentes en cada tanque sin necesidad de tener ningún tipo de intervención humana.
La red neuronal utilizada ya ha sido previamente entrenada para detectar objetos en imágenes, por ello, se ha aplicado el método de *Transfer Learning*. Este método consiste en aplicar conocimientos adquiridos en la resolución de un problema en la resolución de otro problema distinto.

Este documento consta de siete partes fundamentales. Primero se introduce el marco tecnológico, donde se describen las tecnologías existentes que son de interés para este proyecto. En este apartado se describen las soluciones que existen actualmente para el conteo de peces, se definen los distintos campos de la inteligencia artificial y se explica tanto el funcionamiento de las redes neuronales en general como el funcionamiento de las redes neuronales convolucionales. A continuación, se describen las especificaciones y restricciones de diseño del sistema. Después, se describe el desarrollo de la solución propuesta dividiendo el proyecto en tres fases: fase de

etiquetado de imágenes, fase de reformateo de datos y fase de entrenamiento de la red neuronal. Tras esto, se presenta el análisis de los resultados obtenidos junto con las conclusiones y las líneas futuras de trabajo derivadas de este estudio de los resultados. Por último, se incluye el presupuesto del proyecto realizado y las referencias consultadas para la elaboración del mismo.

2. Marco tecnológico

2.1. Tecnologías para conteo de peces en piscifactorías [1, 2]

Como ya se ha mencionado en el capítulo anterior, es de gran utilidad poder llevar un conteo del número de peces presentes tanto en entornos naturales como en piscifactorías. Con este objetivo, en los últimos años se han ido desarrollando distintos sistemas basados en tecnologías como los sensores electrónicos, el análisis de vídeos, el tratamiento de imágenes o la emisión de señales acústicas.

2.1.1. Sensores

Este tipo de soluciones suelen utilizar un sensor compuesto por un transmisor y un receptor. El transmisor emite de forma continuada una señal que el receptor capta con una intensidad mayor o menor en función de si algún pez pasa entre medias de las dos partes del sensor.

Sin embargo, este tipo de soluciones no siempre son precisas, ya que no son capaces de distinguir cuándo está pasando un solo pez y cuando han sido varios peces los que han atravesado el sensor al mismo tiempo. Además, en estos sistemas no se puede saber si algún pez ha atravesado varias veces el sensor y, por tanto, se ha contado más de una vez.

Por último, cabe destacar que en algunas ocasiones se ha visto que un incremento de la turbidez del agua ha provocado una disminución de la eficiencia en este tipo de sistemas, lo que supone un gran inconveniente.

2.1.2. Señales acústicas

Los sistemas acústicos son una buena alternativa a los sistemas basados en sensores ya que su eficiencia no se ve afectada por la turbidez del agua. Estos sistemas suelen emitir ondas sonoras para posteriormente captar el eco de estas señales emitidas. De forma similar a como ocurre con los sistemas basados en sensores, se detectará un objeto siempre que se reciba el eco de una señal emitida.

No obstante, no todos estos sistemas funcionan de la misma manera: algunos sistemas son capaces de determinar la densidad del objeto detectado en función de la señal acústica recibida, lo que permite diferenciar a los peces de otros tipos de objetos. Otros sistemas, en cambio, utilizan un sonar para crear una imagen del entorno y, de esta forma, poder realizar el conteo de peces procesando la imagen acústica obtenida.

Sin embargo, estos sistemas muchas veces no son capaces de hacer un conteo preciso cuando el tamaño de los peces es pequeño.

2.1.3. Análisis de videos

También existen sistema que se basan en el análisis de vídeos para dar solución al problema del conteo de peces. Los vídeos tienen la ventaja de almacenar información sobre el movimiento que han realizado distintos objetos a lo largo del tiempo. Esto permite llevar a cabo un proceso de "*tracking*", que consiste en identificar las trayectorias que ha seguido cada uno de los objetos presentes en el vídeo. De esta forma, al conocer el número de trayectorias detectadas, se puede obtener el número de peces presentes en el vídeo.

Este tipo de soluciones muchas veces usan sistemas de "*machine learning*" para poder detectar las trayectorias de los distintos objetos.

2.1.4. Segmentación de imágenes

En lugar de trabajar directamente con vídeos, en algunas soluciones se opta por utilizar tratamiento de imágenes. Estas soluciones tienen como principal objetivo realizar una segmentación de los objetos de una imagen, que consiste en diferenciar qué parte de la imagen compone el fondo y qué parte no. Este proceso de segmentación se puede realizar de múltiples formas, pero casi todas ellas se basan en la detección de bordes, ya que los cambios bruscos de luminosidad o color suelen corresponder a la presencia de un borde en la imagen.

En este proceso de segmentación se pueden detectar objetos que no son peces. Por ello, se suele llevar a cabo un procesado de las características morfológicas del objeto (área, perímetro, forma, …) que permita descartar los objetos cuya forma no sea similar a la de un pez.

Este tipo de soluciones presenta un problema en relación a la calidad en las imágenes con las que se quiere trabajar, ya que, si las imágenes tienen poca luminosidad, poca resolución o el agua está demasiado turbia los resultados pueden no ser satisfactorios.

2.1.5. Detección de objetos

Los sistemas de conteo de peces que se basan en la detección de objetos también trabajan con imágenes fijas. Sin embargo, al contrario que los sistemas basados en segmentación de imágenes, este tipo de soluciones suelen hacer uso de sistemas de *machine learning* para detectar la ubicación de los objetos de interés presentes en una imagen.

Los sistemas de *machine learning* son sistemas que son capaces de aprender a realizar una tarea partiendo de un banco de datos que contienen tanto el problema planteado como la solución al que debe llegar el sistema.

En el caso de los sistemas de conteo de peces, el problema planteado sería la imagen en la que se quiere saber cuántos peces hay y, la solución vendría dada por la ubicación de cada uno de los peces contenidos en esa imagen.

2.2. Inteligencia artificial y *machine learning*

Actualmente, la inteligencia artificial es una herramienta ampliamente utilizada. Gran parte de su éxito reside en que su aplicación no se tiene que limitar a un ámbito concreto, sino que la inteligencia artificial permite la resolución de problemas muy diversos como puede ser la clasificación de datos, la predicción de eventos según una serie de datos históricos dados o la identificación de distintos tipos de objetos en imágenes.

Sin embargo, la inteligencia artificial no es una tecnología en sí misma, sino que es la ciencia que se encarga del desarrollo de sistemas computacionales capaces de llevar a cabo tareas que, de otra manera, requerirían de la inteligencia humana para poder realizarse **[3]**. Estos sistemas computacionales, que sí son tecnologías en sí mismas, se dividen en varios tipos tal como se puede ver en la siguiente imagen:

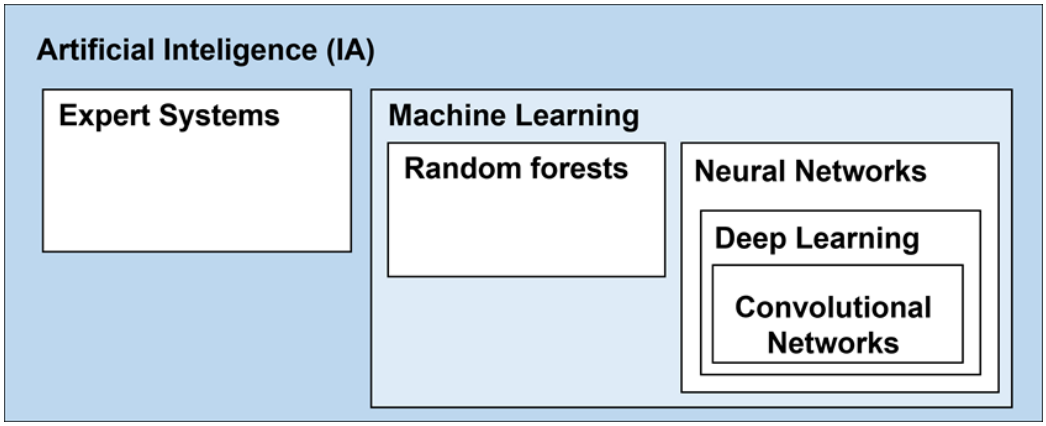


Imagen 1 - Campos de la inteligencia artificial (Fuente: [4])

Dentro de la inteligencia artificial existen dos campos fundamentales: los sistemas expertos y el aprendizaje máquina o *machine learning*. La principal diferencia entre estos dos campos reside en que en los sistemas expertos un experto en un área concreta enseña a la red cómo realizar una tarea concreta, mientras que en el *machine learning* es el propio sistema computacional el que aprende cómo realizar una tarea concreta a partir de una serie de datos históricos recogidos, también conocido como *dataset* **[4]**.
Se puede decir, por tanto, que mientras que un sistema experto se basa en el conocimiento de un experto humano, un sistema de *machine learning* es capaz de diseñar por sí mismo todo el proceso que finalmente utilizará para completar la tarea asignada.

Los sistemas de *machine learning* también se dividen en función de cómo son los datos que se le introducen al mismo y, por tanto, del tipo de aprendizaje que tienen a partir de esos datos suministrados. Hay cuatro tipos de aprendizaje: **aprendizaje supervisado**, **aprendizaje no supervisado**, **aprendizaje semi-supervisado** y **aprendizaje reforzado o *reinforcement learning***.

Estos tipos de aprendizaje, como ya se ha mencionado anteriormente, se diferencian en qué datos se introducen al sistema para que éste aprenda lo necesario para llevar a cabo una tarea concreta.
En un *dataset* hay dos tipos de datos fundamentales: las características o *features* y las etiquetas o *labels*. Las características son los datos que se le suministran al sistema y que componen el problema que éste debe resolver, mientras que las etiquetas son las respuestas que el sistema debería obtener. A estas etiquetas también se les conoce como *ground truth*. Por ejemplo, si un sistema debe identificar si un inicio de sesión en una cuenta es sospechoso, las características serían todos aquellos datos relacionados con el inicio de sesión que se quiere analizar (equipo desde el que se ha realizado el inicio de sesión, ubicación geográfica de ese equipo, hora del inicio de sesión, etc) mientras que la etiqueta sería un valor binario, por ejemplo, 1 si ha sido un intento de sesión malicioso y 0 si no lo ha sido.

Este proceso es el que se lleva a cabo si se tiene un **aprendizaje supervisado**: se **le suministran al sistema tanto las características como las etiquetas** y el sistema se ajusta a sí mismo, mediante un procedimiento que se explicará más adelante, para poder que las salidas que obtenga a partir de las características sean lo más cercanas posibles al *ground truth*. Esto permite que, después de que se haya llevado a cabo este aprendizaje, el sistema pueda obtener una salida para unas características dadas sin que se conozca su etiqueta.

El **aprendizaje no supervisado** difiere con el aprendizaje supervisado en que desde el primer momento **no se le suministra ninguna etiqueta para las características** dadas debido a que el valor de estas etiquetas se desconoce o no es relevante. Este tipo de aprendizaje suele ser útil para tareas de agrupamiento en las que, por ejemplo, se deben agrupar elementos según ciertas características similares sin que sea necesario identificar qué grupos se han creado ni qué elementos contiene cada grupo. Este sería el caso en el que se quisiera agrupar una serie de coches según el modelo al que pertenecen, pero no nos interesase saber a qué modelo pertenece cada uno de ellos.

El **aprendizaje semi-supervisado** está en un punto intermedio entre el aprendizaje supervisado y el no supervisado. Es este tipo de aprendizaje una parte del *dataset* está formado por características con sus etiquetas correspondientes y la otra parte de está formada por características que no tienen asignada ninguna etiqueta. El aprendizaje supervisado es muy útil cuando el número de características con sus respectivas etiquetas que se tiene no es suficientemente grande para entrenar al sistema.

El procedimiento para entrenar a este tipo de sistemas es el siguiente **[5]**:

1. Entrenar el sistema con las características de las que se disponen las etiquetas para generar un modelo.
2. Introducir en el modelo obtenido en el paso anterior las características de las que se desconoce las etiquetas. De esta forma se obtienen las “pseudo-etiquetas”, que no tienen por qué ser correctas en todos los casos.
3. Se escogen las “pseudo-etiquetas” que tengan una mayor confianza por parte del sistema, y que, por tanto, tienen un mayor probabilidad de ser correctas, y se añaden al conjunto inicial de características etiquetadas.
4. Se vuelve a entrenar el sistema con los datos etiquetas originales y los datos seleccionados en el paso anterior para mejorar el rendimiento del sistema.

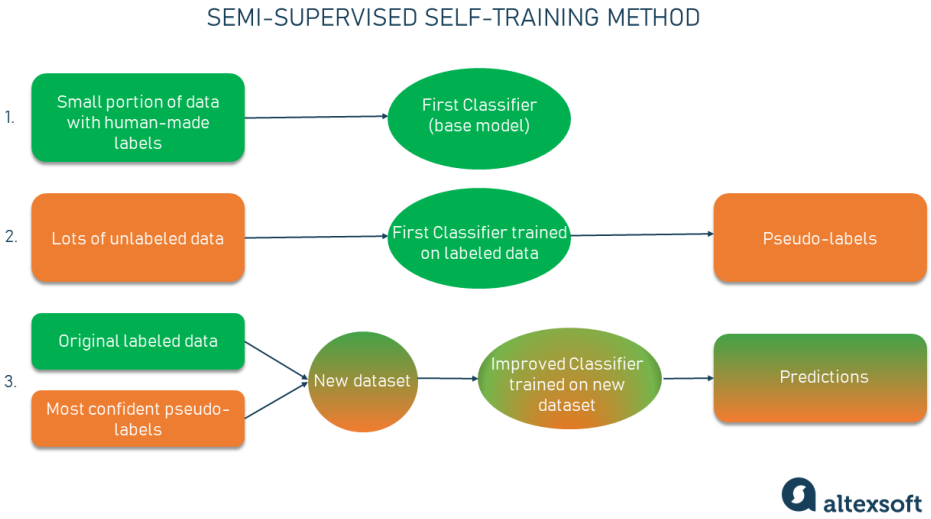


Imagen 2 - Método de entrenamiento de un sistema con aprendizaje semi-supervisado (Fuente: [5])

El último tipo de aprendizaje es el **aprendizaje reforzado o *reinforcement learning***. Este tipo de aprendizaje es similar al aprendizaje supervisado con la diferencia de que no se le suministran características ni etiquetas. Este tipo de sistemas va tomando distintas decisiones y realizando unas u otras acciones en consecuencia. Según las acciones que el sistema lleve a cabo se le dará una recompensa si la acción ayuda a alcanzar el objetivo final o una penalización en caso contrario. Este tipo de aprendizaje se podría usar para enseñar a un sistema a jugar al videojuego Pac-Man. El sistema recibiría una recompensa, más o menos grande, al obtener puntos o comerse una fruta o a un fantasma y recibiría una recompensa negativa cada vez que perdiese una vida **[6] [7]**.

Además de tipos de aprendizajes, también existen tipos de sistemas de *machine* learning. De entre los distintos tipos existentes, los más conocidos y usados actualmente son los basados en redes neuronales, sin embargo, también se utilizan otros sistemas, como los bosques aleatorios (también conocidos como *random forests*) o las máquinas de vectores de soporte (*SVM* por sus siglas en inglés *Support Vector Machines*).

2.3. *Random forests*

Los *random forests* son algoritmos formados por un conjunto de árboles de decisión que se utilizan para resolver tareas de clasificación y regresión (ambos conceptos se explicarán más adelante). Los árboles de decisión que forman los *random forest* son otro tipo de algoritmos, en este caso predictivos, que funciones mediante el planteamiento de una serie de preguntas **[8]**.

En la *Imagen 2* se puede ver un ejemplo de árbol aleatorio creado pata determinar cuándo es mejor coger el autobús o ir andando a un destino conocido.

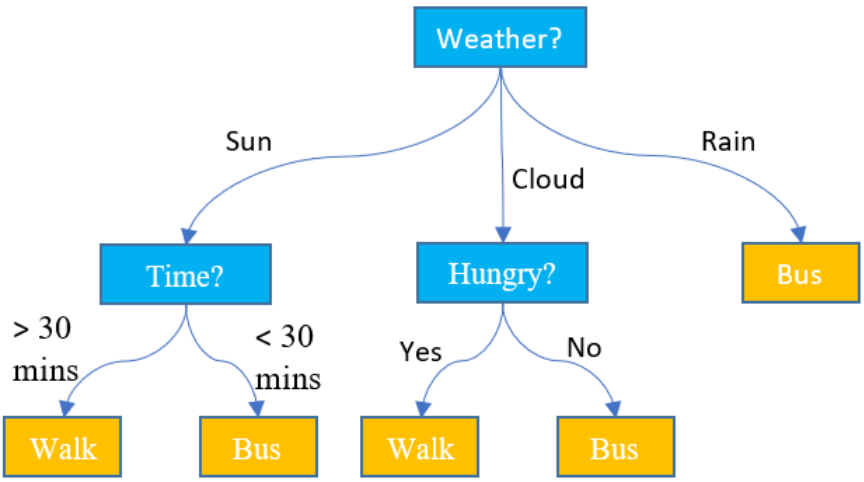


Imagen 3 - Ejemplo de árbol aleatorio (Fuente: [8])

A estos árboles se les van suministrando una serie de datos históricos registrados con los que son capaces de ir creando su propia estructura. En este ejemplo concreto se habrían suministrado datos de condiciones existentes en el momento de realizar un trayecto (tiempo necesario para llegar al destino, condiciones meteorológicas, día de la semana en el que se realiza el trayecto, …) y la decisión que finalmente se tomó (ir en autobús o ir andando) y es gracias a este conjunto de datos que el árbol es capaz de ir determinando qué preguntas debe realizar y qué opciones contempla como respuesta a cada una de ellas.

Siguiendo este método, para construir un *random forest* se entrenan varios de estos árboles de decisión con distintas secciones de un mismo *dataset* para que, trabajando conjuntamente, sean capaces de resolver la tarea de regresión o clasificación que se les haya asignado **[9]**.

2.4. *Support Vector Machines* (SVM)

Los SVM son otro tipo de algoritmos utilizados principalmente en problemas de clasificación, aunque también pueden ser usados en problemas de regresión.
El objetivo del algoritmo SVM es calcular un hiperplano que es un subespacio que tiene una dimensión menos que el espacio en el que está contenido y que divide a este espacio en dos regiones. Por ejemplo, en $\mathbb{R}^3$ un hiperplano sería un plano de dimensión 2, sin embargo, en $\mathbb{R}^2$ un hiperplano sería una recta.

En tareas de regresión el plano que se debe obtener es el que sea capaz de contener el mayor número de puntos, que representan los distintos datos con los que se está trabajando **[10]**.

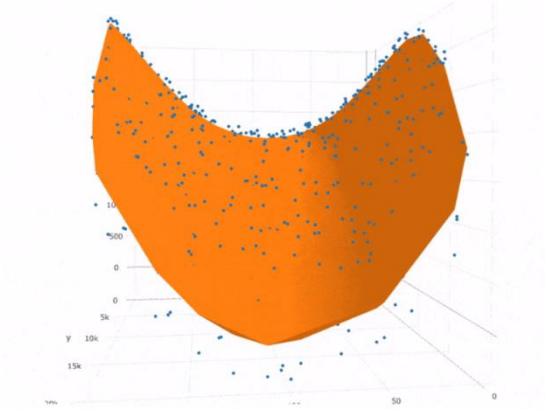


Imagen 4 - Ejemplo de un hiperplano (en naranja) calculado con SVM a partir de los datos (en azul) de un problema de regresión. (Fuente: [10])

En cambio, en las tareas de clasificación este hiperplano debe ser capaz de separar los datos pertenecientes a distintas clases y se obtiene maximizando la distancia entre el plano y los datos de cada clase que estén más cerca de éste **[11]**.

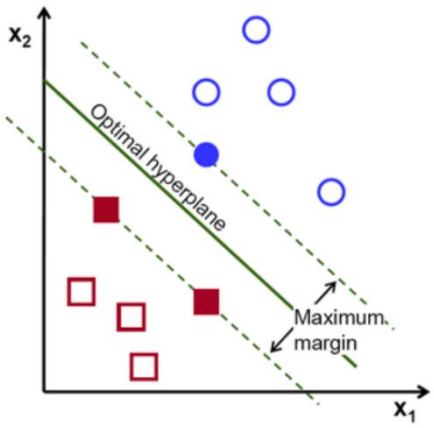


Imagen 5 - Ejemplo de un hiperplano calculado con SVM para un problema de clasificación (Fuente: [11])

2.5. Redes neuronales

Como se ha mencionado anteriormente, hoy en día las redes neuronales son más conocidas y utilizadas que los *random forest*. Esto se debe principalmente a que los problemas que pueden solucionar las redes neuronales son mucho más variados que los que se pueden solucionar con los bosques aleatorios.

Algunos usos comunes que se les da a las redes neuronales son:

- **Regresión:** los problemas de regresión son aquellos en los que la salida que se quiere obtener es un valor numérico. Un ejemplo de problema de regresión sería querer determinar el número de personas que van a ir a un evento partiendo de datos como la fecha, el lugar y la duración del mismo.

- **Clasificación**: los problemas de clasificación son aquellos en los que se busca determinar a qué categoría, de entre varias opciones, pertenece una entrada de datos. En las tareas de clasificación el valor de salida corresponde a la probabilidad de que un objeto pertenezca a una clase determinada, por lo que su valor siempre estará comprendido entre 0 y 1. En los problemas de clasificación se distinguen dos tipos: la clasificación binaria y la multiclase. La principal diferencia es que en la clasificación binaria solo existe una clase por lo que lo único que se debe determinar es si un elemento pertenece a esa clase o no, mientras que en la clasificación multiclase se busca determinar a qué clase, de entre dos o más clases distintas, pertenece un elemento.

Aparte de estos dos problemas, hay un campo en el que las redes neuronales suelen obtener unos resultados superiores a otros sistemas conocido como visión artificial o *computer vision*.

La visión artificial es un campo de la IA que permite que los ordenadores y sistemas informáticos puedan obtener e interpretar información significativa presente en distintos tipos de entradas visuales, como imágenes digitales o vídeos. En estos casos muchas veces se desea saber si una serie de objetos están presentes en una imagen determinada, sin embargo, dependiendo del tipo de información que se quiera obtener sobre estos objetos existen distintos tipos de tareas. A continuación, se explican algunas de las tareas más comunes indicando cuál sería la salida de una red entrenada para detectar gatos si la entrada dada fuese la *Imagen 4*:

Imagen 6 - Entrada de una red hipotética que debe detectar gatos en una imagen. (Fuente [27])

- Detección de objetos: El objetivo principal de la detección de objectos consiste en localizar objetos de interés en una imagen detectando a qué clase corresponden. Los datos de salida de la red deben ser la probabilidad de que ese objeto pertenezca a esa clase y las coordenadas del cuadrado más pequeño que pueda contener al objeto, también conocido como *bounding box*. En el caso de tener como entrada la *Imagen 4*, la red daría como salida la siguiente información **por cada gato detectado**: las coordenadas del *bounding box* y la probabilidad de que el objeto pertenezca a cada una de las clases que puede detectar la red. Además, las redes neuronales suelen proporcionar a la salida un dato adicional conocido como nivel de confianza. Este dato indica la seguridad que tiene la red en la respuesta que ha dado.

a)
```
classes     = ["cat", "dog", "bicycle", "nothing"]

prediction  = [ 0.8 ,  0.1 ,   0.05,      0.05   ]
```

b)
```
legend     = [ "X-Position", "Y-Position", "Length", Height"]

prediction = [     130,          285,       100,     185   ]
```

Imagen 7 - Ejemplo de salida de la red. a) Probabilidad de que un objeto pertenezca a cada una de las clases que puede detectar la red. b) Coordenadas del "Bounding Box" (Fuente: [27])

La representación visual de esta información consistiría en la siguiente imagen:

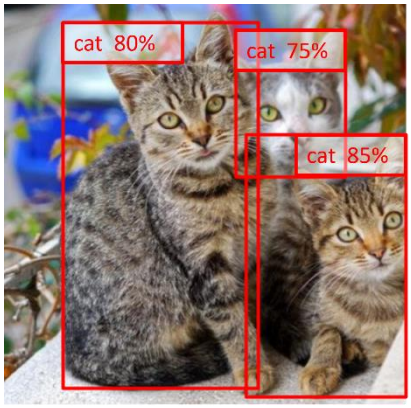


Imagen 8 - Representación visual de la salida de una red que implementa la detección de objetos (Fuente: [27])

- Segmentación de instancia: La segmentación de instancia consiste en una detección de objetos en la que queremos indicar qué píxeles concretamente pertenecen a cada objeto. Como salida se obtienen los mismos datos que en la detección de objetos (es decir, probabilidad de que el objeto pertenezca a cada clase y coordenadas del *bounding box*), además de generarse una máscara binaria del objeto. La máscara binaria es un array con las mismas dimensiones en píxeles que la imagen original, en la que los valores de estos píxeles pueden ser 0 o 1 dependiendo de si el píxel con la misma posición en la imagen original es parte del objeto de interés o no.

Las máscaras binarias (una por cada objecto detectado) tendrían un aspecto similar al siguiente:

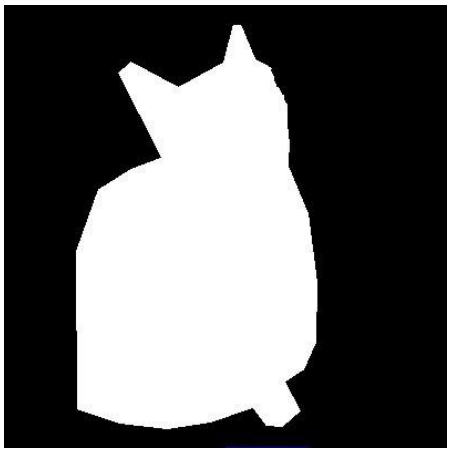


Imagen 9 - Máscara binaria correspondiente al gato de la izquierda de la Imagen 4 (Fuente: [27])

En esta ocasión, la representación visual de esta información consistiría en la siguiente imagen:

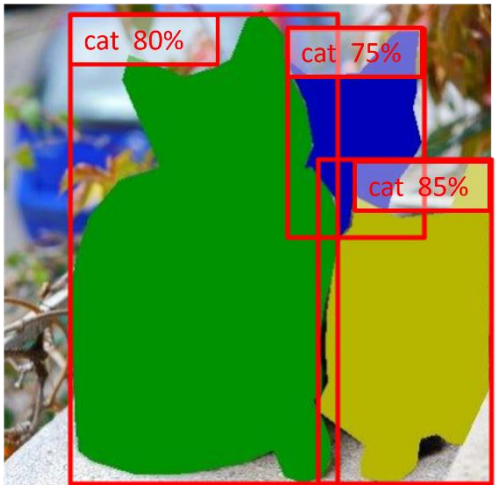


Imagen 10 - Representación visual de la salida de una red que implementa segmentación de instancia (Fuente: [27])

- Segmentación semántica: La segmentación semántica es similar a la segmentación de instancia en cuanto a que se trabaja a nivel de píxel, sin embargo, no se hace distinción entre objetos de la misma clase, sino que lo que interesa es identificar a qué clase de objeto pertenece cada una de las áreas de la imagen. En este tipo de segmentación se tienen en cuenta todas las áreas de la imagen, no solo las áreas que contienen los objetos de interés. A los elementos que se pueden contar se les suele llamar "*things*", mientras que al resto de elementos se les suele llamar "*stuff*" (por ejemplo, el suelo o el cielo).

En la *Imagen 4* se pueden diferenciar tres clases de objetos: suelo, fondo y gato. Por lo tanto, la salida de la red representada gráficamente sería la siguiente:

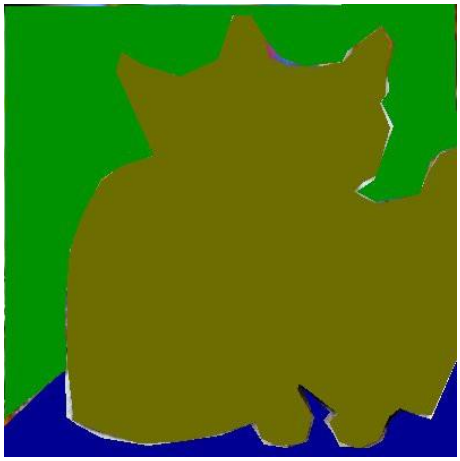


Imagen 11 - Representación visual de la salida de una red que implementa segmentación semántica (Se representa en azul el suelo, en verde el fondo y en verde amarronado el conjunto de los gatos) (Fuente: [27])

- Segmentación panóptica: Este tipo de segmentación es una mezcla entre segmentación semántica y segmentación de instancia. Su principal objetivo es indicar los píxeles que pertenecen a cada clase y, dentro de ellos, a qué instancia pertenece cada área de píxeles. La salida de la red se podría interpretar como una imagen de 2 canales (o capas de datos): uno para indicar a qué clase pertenecen los píxeles y otra para indicar la instancia. La instancia se identificará como "*None*" en el caso de que esa clase fuera región del tipo "*stuff*".

  A la salida de la red se le habrían asignado dos valores a cada píxel de la imagen de la siguiente manera:

```
["L", "Z"] => ["Label", "Instance Number"]
```

Imagen 12 – Ejemplo de salida de una red que implemente segmentación panóptica (Fuente: [27])

Siendo "*Label*" la clase a la que pertenece el píxel e "*Instance Number"* el número de instancia al que pertenece el pixel dentro de todas las instancias correspondientes a la clase a la que pertenece el píxel.

La representación visual de la información obtenida a la salida de la red quedaría de la siguiente forma:

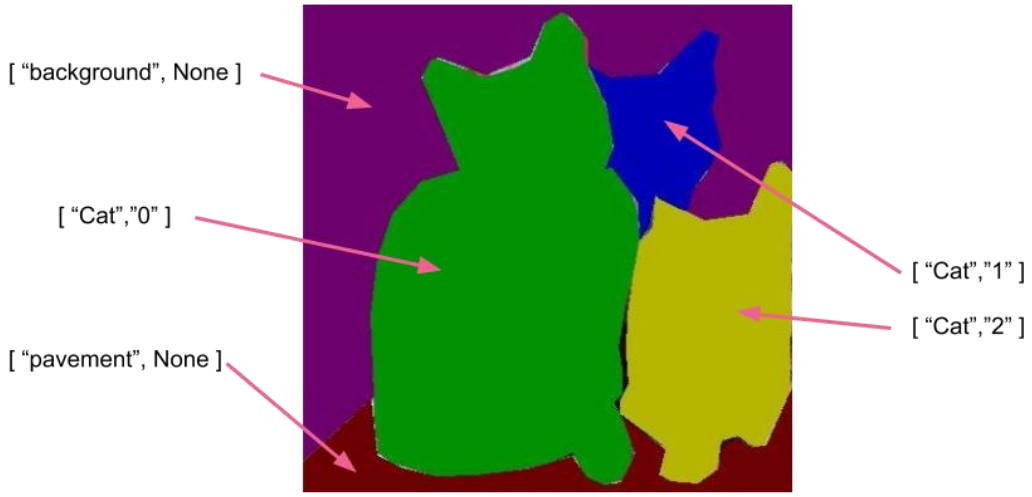


Imagen 13 - Representación visual de la salida de una red que implementa segmentación panóptica (Fuente: [27])

Ya se ha visto que las redes neuronales permiten trabajar con una gran variedad de problemas. Su principal potencia reside en su habilidad para **reconocer relaciones existentes** entre los distintos datos suministrados en el *dataset*.
Para lograr esto las redes neuronales están compuestas por unas **unidades matemáticas básicas llamadas neuronas**. Estas unidades básicas realizan una operación matemática sencilla, de tal forma que combinando cada una de las operaciones realizadas por cada una de las neuronas que componen una red, se puede llegar a obtener la solución a un problema complejo.

Cada una de estas neuronas son en esencia una pequeña función que recibe unos datos de entrada que tendrá que procesar para obtener un único dato de salida.

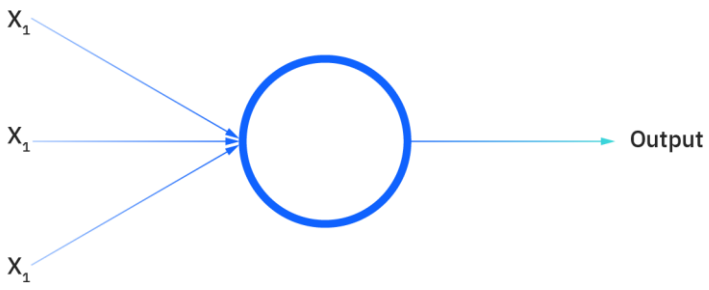


Imagen 14 - Esquema simplificado de una neurona en una red neuronal (Fuente: [23])

Este procesamiento de datos viene dado por la siguiente expresión:

$$y = \sigma((\sum_{i=0}^{n-1} x_i * w_i) + b) \quad (1)$$

Como se puede ver, la ecuación (1) consta, a su vez, de dos operaciones distintas:

$$z = (\sum_{i=0}^{n-1} x_i * w_i) + b \quad (2)$$

$$y = \sigma(z) \quad (3)$$

En la ecuación (2) se puede ver que se opera con tres tipos de parámetros: **x, w y b**. Los **parámetros x_i** Son los distintos datos de entrada que recibe una neurona, los **parámetros w_i y b** son parámetros que la propia neurona irá modificando y ajustando en la etapa de entrenamiento, según el procedimiento que se explicará más adelante, para poder llevar a cabo la tarea asignada.

A estos **parámetros w_i** se les conoce como **"pesos".** Esto se debe a que mediante estos parámetros las neuronas determinan la importancia que tienen cada una de las características en la respuesta final de la red. Por ejemplo, si continuamos con el ejemplo de si un inicio de sesión es sospechoso, podríamos tener los siguientes datos de entrada:

- x_1 = ubicación geográfica del equipo desde dónde se ha realizado el inicio de sesión.
- x_2 = marca y modelo del equipo con el que se ha iniciado sesión.
- x_3 = fecha y hora en la que se ha realizado el intento de inicio de sesión.

Si suponemos que nuestra red neuronal tiene una sola neurona tendríamos el siguiente sistema:

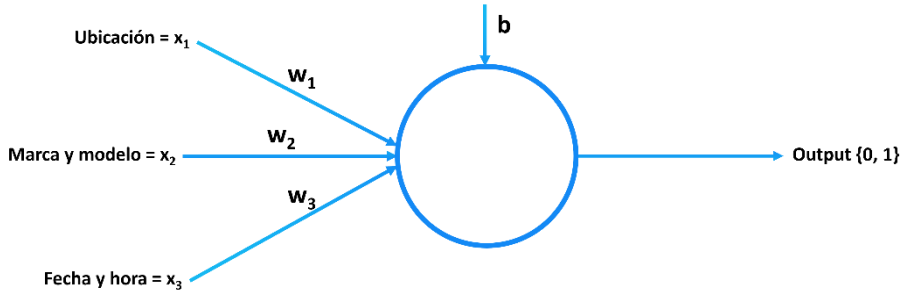


Imagen 15 - Esquema de neurona para identificar intentos de inicio de sesión maliciosos

De esta manera, la neurona podría aprender, después de ser entrenada, que la ubicación del dispositivo es el dato más relevante de los tres valores que recibe de entrada. Esto se vería reflejado en que el parámetro asociado a x_1, es decir, **w_1, tendría un valor superior** al de los otros dos parámetros, w_2 y w_3.

Además, en la *Imagen 5* vemos que w_1, w_2 y w_3 están asociados cada uno a una característica distinta mientras que el parámetro b no está asociado a ningún parámetro concreto. Esto se debe a que la función del parámetro b, llamado *bias*, consiste principalmente en ajustar la red para que la salida se ajuste lo máximo posible al *ground truth*.

Este parámetro tiene la misma función que el termino independiente en una ecuación de primer grado del tipo y = ax + b, puesto que permite desplazar la función a lo largo del eje x, lo que permite que a la salida de la red se pueda trabajar con un rango de valores mucho mayor **[12]**.

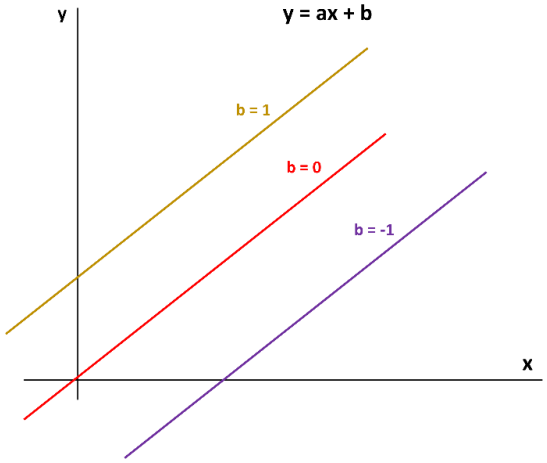


Imagen 16 - Influencia del parámetro b en la salida de una red neuronal

Por tanto, como se indicaba en la ecuación 2, la primera operación que realiza la red con los datos de entrada que se le suministran es multiplicar los pesos con los valores de entrada correspondientes y sumarle el parámetro de *bias*.

Esta es una operación lineal (solo contiene sumas y multiplicaciones) y las funciones lineales no presentan ningún tipo de curvas, sino que son líneas rectas con una pendiente determinada. Por ello, no suele ser común trabajar con operaciones lineales, ya que, si se utiliza una operación no lineal se consigue una mayor adaptación al modelo real de distribución de los datos tal como se puede ver en la *Imagen 17*.

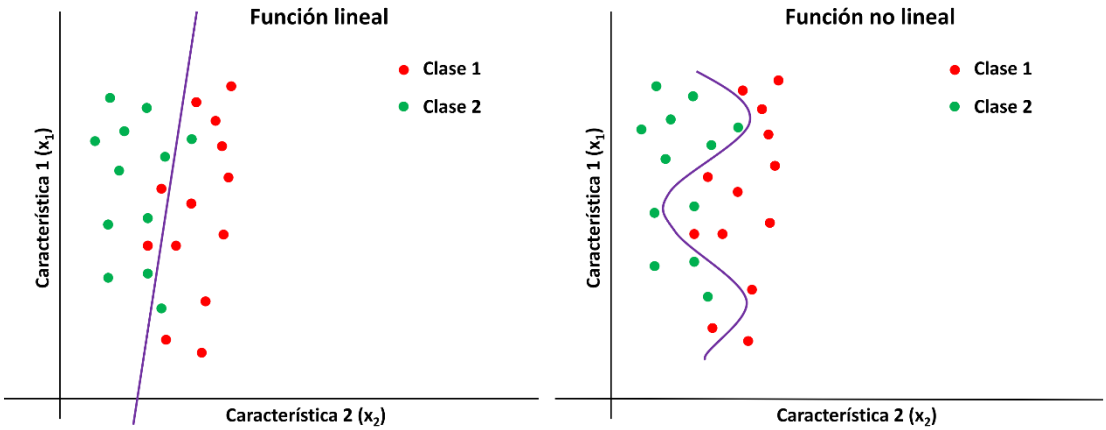


Imagen 17 - Ventajas del uso de una función no lineal para el procesamiento de datos en una neurona

La ecuación (3), también llamada "función de activación", consiste en aplicar una función llamada σ, generalmente no linear, al resultado obtenido en la ecuación (2), para que la salida de las distintas neuronas pueda ser no linear en caso de que sea necesario.

Hay varias **funciones σ** que se pueden utilizar, algunas de ellas son la siguientes:

1. Función linear

   En algunas ocasiones puede interesar que la salida de la función de activación de una neurona en concreto sea igual a su entrada. En este caso podemos utilizar una función de activación lineal que se describe de la siguiente manera:

$$\sigma(z) = z \tag{4}$$

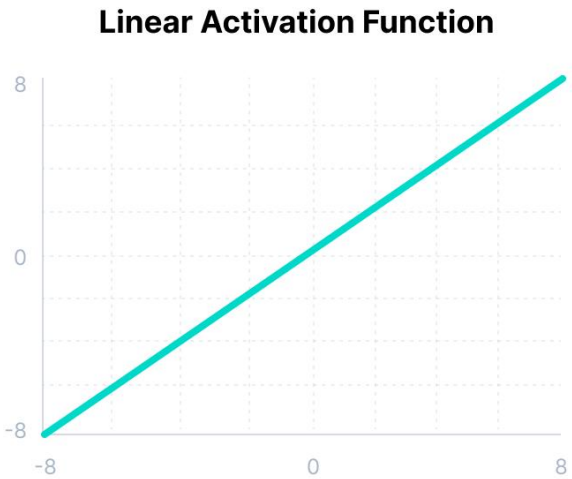


Imagen 18 - Función de activación lineal (Fuente: [24])

Por todo lo que se ha comentado anteriormente, **esta función de activación no se suele usar**.

2. Función ReLU

La función ReLU ("*Rectified Linear Unit*") es similar a la función de activación lineal. En concreto, presenta el mismo comportamiento que la función lineal en el intervalo [0, ∞), mientras que en el intervalo (-∞, 0] su valor es siempre nulo.
La expresión matemática correspondiente a esta función es la siguiente:

$$\sigma(\boldsymbol{z}) = \mathbf{max}\,(\mathbf{0}, \boldsymbol{z}) \tag{5}$$

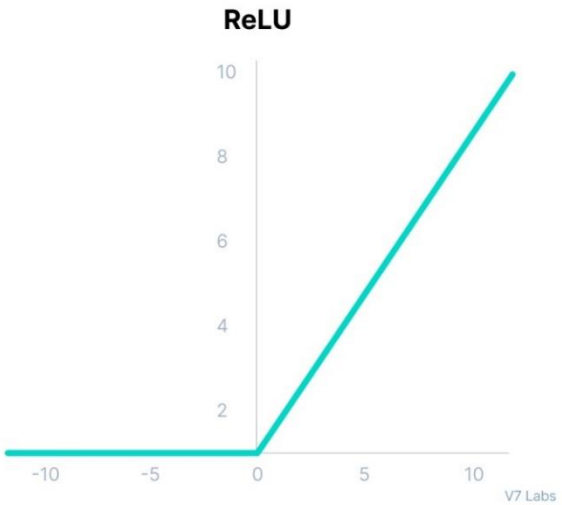


Imagen 19 - Función de activación ReLU ("Rectified Linear Unit") (Fuente: [24])

Esta función de activación permite que se "active", es decir, que tome un valor no nulo, dependiendo tanto del valor de los datos de entrada y la relación entre ellos, como del valor de los parámetros de la propia neurona.

3. Función sigmoidal

La función sigmoidal es una función no lineal en todo su dominio, está acotada entre 0 y 1 y, a diferencia de la función ReLU, es derivable en todo punto.
Se define matemáticamente de la siguiente manera:

$$\sigma(\boldsymbol{z}) = \frac{\mathbf{1}}{\mathbf{1} + \boldsymbol{e}^{-\boldsymbol{z}}} \tag{6}$$

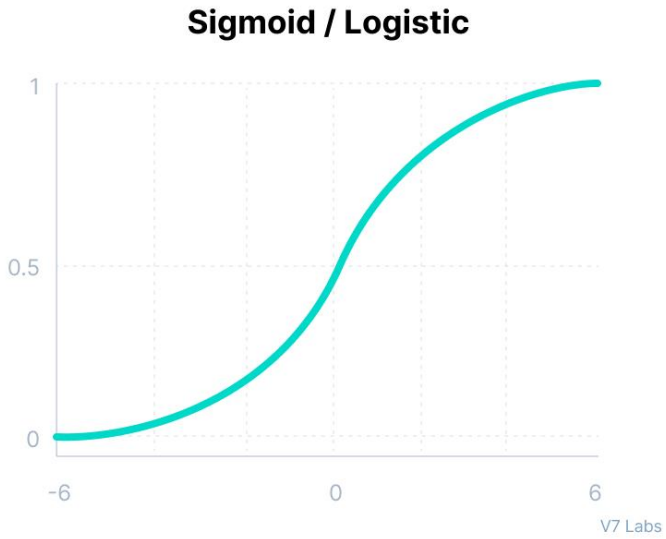


Imagen 20 - Función de activación sigmoidea (Fuente: [24])

Como se puede ver en la *Imagen 10*, la función sigmoidea devuelve valores cercanos a 1 cuando la entrada es un valor grande positivo y devuelve valores cercanos a 0 cuando la entrada es un valor pequeño negativo. Esto provoca que su valor medio a la salida esté en torno a 0,5.

4. Función tangente hiperbólica (TanH)

La función tangente hiperbólica también se puede usar como función de activación. Es una función no lineal; acotada entre -1 y 1 y es derivable en todo punto.
Su expresión matemática es la siguiente:

$$\sigma(\boldsymbol{z}) = \frac{\boldsymbol{e^{z}} - \boldsymbol{e^{-z}}}{\boldsymbol{e^{z}} + \boldsymbol{e^{-z}}} \qquad (7)$$

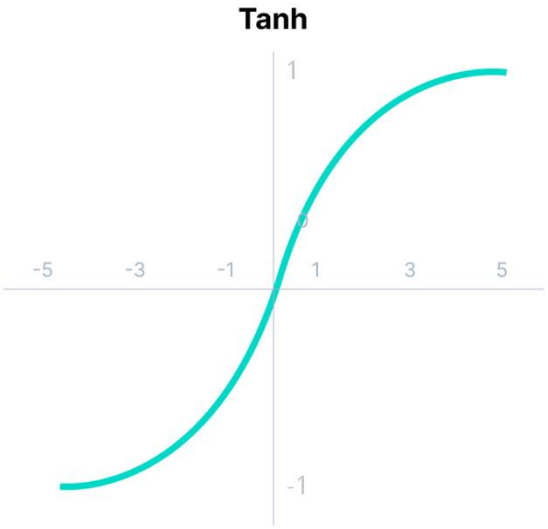


Imagen 21 - Función de activación tangente hiperbólica (Fuente: [24])

A diferencia de la función sigmoidea, esta función está centrada en 0, por lo que su valor medio en la salida será cercano a este valor. Además, toma valores de salida muy cercanos a 1 para valores de entrada muy grandes (positivos) y muy cercanos a -1 para valores de entrada muy pequeños (negativos).

5. Función "*Swish*"

La función "*Swish"* es la más reciente de todas las funciones de activación mencionadas. Fue desarrollada por investigadores de Google en 2017 como sustitución a la función ReLU. Las principales diferencias que tiene con la función ReLU es que es derivable en todo punto y que en valores cercano a cero su valor no es nulo.
Su definición matemática es la siguiente:

$$\sigma(\boldsymbol{z}) = \frac{\boldsymbol{z}}{\mathbf{1} + \boldsymbol{e}^{-\boldsymbol{z}}} \qquad (8)$$

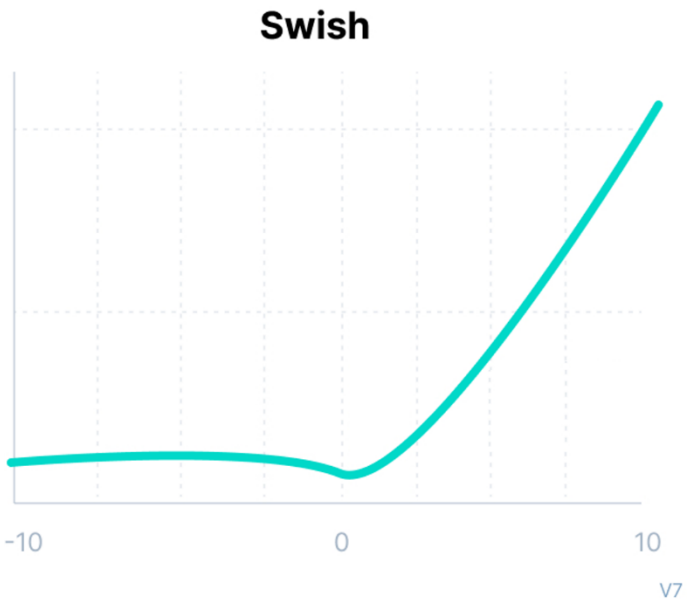


Imagen 22 - Función de activación "Swish" (Fuente: [24])

Debido a las diferencias que presenta con respecto a la función ReLU, esta función siempre obtiene resultados iguales o mejores con respecto a ReLU. Esto se debe a que los valores de entrada cercanos a 0 pueden ser relevantes, por lo que no anularlos a la salida de la función de activación suele mejorar el rendimiento del sistema. Además, en el proceso de aprendizaje de la red, que se describirá más adelante, se necesita obtener la derivada de la función de activación. Esto implica que, en caso de tener funciones con características parecidas como "*Swish*" y ReLU es preferible utilizar una función derivable en todo punto.

Todas las funciones de activación que se han visto tienen características distintas, por lo que se deberá determinar qué función se debe aplicar a cada neurona según la tarea para la que se quiera entrenar a la red.

Sin embargo, todo lo visto hasta ahora aplicado a una sola neurona no es de gran utilidad. Lo que realmente permite a las redes neuronales realizar tareas complejas imitando en cierta forma el comportamiento de un cerebro humano es la **cooperación de varias neuronas en una misma red**.

Estas neuronas que operan de forma conjunta **se organizan en varias capas**, de tal manera que **una capa recibe como datos de entrada los datos de salida de la capa anterior**.
Fundamentalmente, las capas se dividen en tres tipos:

- Capa de entrada o *input layer*: es la capa que recibe como datos los datos de entrada a la red, es decir, **no hay ninguna otra capa por delante de ella**. El número de neuronas de esta capa es igual al número de características con las que se va a entrenar a la red.

- Capa de salida o *output layer*: **es la última capa de la red** y, por tanto, la capa que genera la salida final de la red. El número de neuronas de esta capa es igual al número de elementos que contenga la salida de la red. Es decir, si la red debe devolver un valor binario (0, 1) la capa de salida solo tendrá una neurona, pero si la red debe devolver, por ejemplo, el tipo de objeto que se detectan en una imagen junto con su tamaño la capa de salida deberá tener dos neuronas.

- Capas ocultas o *hidden layers*: son todas **las capas que están en entre la capa de entrada y la capa de salida** de la red. Esto quiere decir que sus datos de entrada provienen de capas anteriores y sus datos de salida son suministrados a capas posteriores.

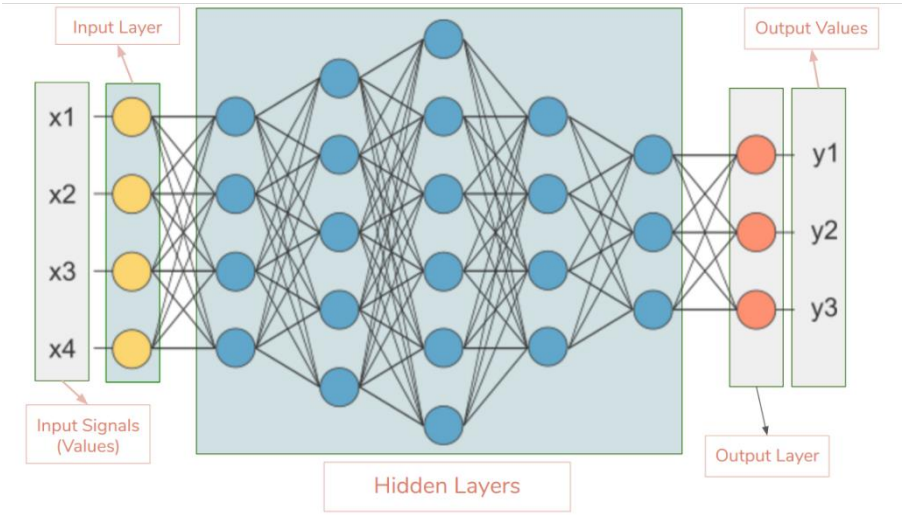


Imagen 23 - Capas de una red neuronal (Fuente: [31])

Hasta ahora se ha visto qué es una red neuronal, cuáles son sus elementos fundamentales y la importancia de trabajar con un conjunto de varias neuronas para dar solución a tareas complejas para los sistemas computacionales.

Además, ya se ha mencionado anteriormente la necesidad de entrenar a la red con un *dataset*, pero **¿cómo aprenden realmente las redes neuronales?**

El proceso de aprendizaje de una red está compuesto de dos fases fundamentales: la fase entrenamiento y la fase de validación. Ambas fases necesitan de un conjunto de datos distintos para poder llevarse a cabo, por ello el *dataset* del que se dispone para entrenar a la red debe dividirse en dos secciones, ambas compuestas por un conjunto de características y sus etiquetas correspondientes. La primera sección, conocida como "set de entrenamiento", se utilizará en la fase entrenamiento y será la más grande de las dos. La segunda, en cambio, será considerablemente más pequeña y se utilizará en la fase de validación. A esta sección se le conoce como "set de validación".

Cada una de estas dos fases llevan a cabo un papel muy distinto en el proceso de aprendizaje. La finalidad de la **fase de entrenamiento** es que **la red aprenda a realizar una determinada tarea** mediante el procesado de los datos que componen el "set de entrenamiento". Sin embargo, la red puede aprender de dos maneras distintas: memorizando o generalizando. Si en la fase de entrenamiento la red ha memorizado los datos del "set de entrenamiento", ésta conseguirá repuestas que se ajusten mucho a la respuesta correcta únicamente cuando reciba como entrada algún elemento de este set. En caso contrario, si ha sido capaz de generalizar, la respuesta de la red se ajustará bastante a la respuesta correcta independientemente de que la entrada pertenezca o no al "set de entrenamiento".

Es fundamental **garantizar que en la fase entrenamiento la red ha generalizado y no ha memorizado**, por ello en la **fase de validación** se deben le introducir a la red los datos del "set de validación" para ver si sus respuestas son buenas (la red ha generalizado) o si son malas (la red ha memorizado).

Por tanto, la fase de validación únicamente consiste en introducir a la red los datos del "set de validación" y evaluar sus respuestas. La fase de entrenamiento, en cambio, es bastante más compleja y se divide en los siguientes pasos:

1. Se introduce a la red todos los datos del "set de entrenamiento", tanto características como etiquetas.
2. La red obtiene la salida para cada entrada del conjunto de datos introducido y las compara con el *groud truth*.
3. La red ajusta los parámetros w y b de distintas neuronas para intentar obtener mejores resultados.
4. Se le vuelven a introducir a la red todos los datos del "set de entrenamiento" y se repite el bucle un número predefinido de veces.

La parte más compleja de la fase de entrenamiento reside en el paso 3, puesto que la red debe determinar qué parámetros debe modificar, así como si debe aumentar o disminuir su valor y en qué magnitud lo debe hacer. Para ello, las redes neuronales utilizan una función matemática llamada **función de coste**. Esta función describe cuánto se ajustan la salida de la red al *ground*

truth de cada entrada del "set de entrenamiento", o, en otras palabras, como de grande es el error que está cometiendo la red en este conjunto de datos.

Esta función de coste relaciona la salida obtenida y la esperada. Cómo la salida final de la red depende de las salidas que obtienen las neuronas de las distintas capas de la red, con esta función de coste se puede averiguar qué parámetros *w* y *b* se deben modificar en cada neurona para que la diferencia entre la salida esperada y la obtenida disminuya.

Por ejemplo, supongamos que nuestra función de coste es una parábola. A los parámetros w y b se les da un valor inicial aleatorio, por lo que el valor inicial a la salida de la red podría ser cualquiera.

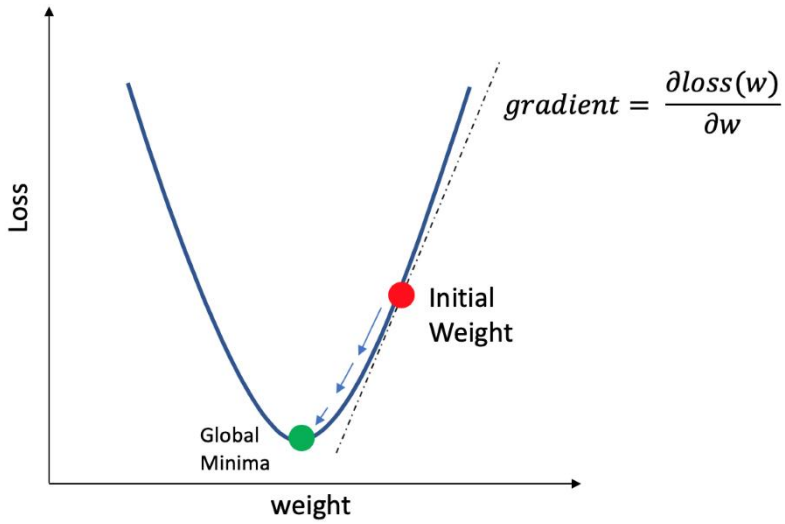


Imagen 24 - Ejemplo de función de coste (Fuente: [25])

Por tanto, el valor inicial de la función de coste podría ser el punto rojo de la *Imagen 24*, sin embargo, se puede ver que el valor de coste mínimo que se puede alcanzar con esta función de coste sería el punto verde. De esta forma, el principal objetivo de la fase de entrenamiento es ajustar los parámetros *w* y *b* de todas las neuronas de la red para que a función de coste pueda alcanzar su valor mínimo, en este caso el punto verde.

Esto se puede conseguir mediante el cálculo de derivadas parciales de la función de coste con respecto a cada uno de los parámetros w y b.
Al calcular las derivadas parciales se obtiene cómo se deben modificar los valores de los distintos parámetros para llegar al punto mínimo de la función de la que se está calculando la derivada. En este ejemplo, si nuestra red se encuentra en la situación del punto rojo, se debería disminuir el valor del parámetro w que se está estudiando para que la función de coste puede acercarse a su mínimo absoluto. Sin embargo, si el coste actual de la red estuviese situado a la izquierda del punto óptimo, sería necesario aumentar el valor de w. Este proceso se conoce como optimización.

No obstante, este procedimiento no es igual de sencillo en todos los casos, puesto que la definición de la función de coste no es única. Esto quiere decir que se pueden utilizar distintas funciones según el tipo de solución que se quiera implementar con la red, al igual que ocurre con la función de activación.

Tal como se ha visto, el entrenamiento de una red neuronal tiene como fundamento encontrar un mínimo de la función de coste, pero en ocasiones se puede llegar a un mínimo local de tal forma que el error a la salida de la red no se pueda seguir disminuyendo, pero que el valor de la función de coste no sea el óptimo.

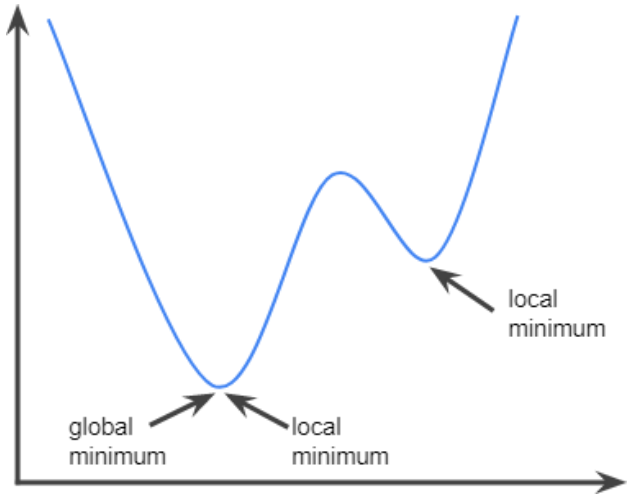


Imagen 25 - Ejemplo de función con mínimo local y mínimo global (Fuente: [26]

En la *Imagen 25* podemos ver una función con un mínimo local y un mínimo absoluto. Si durante el entrenamiento la red llega al mínimo local a la derecha de la función, se interpretará que el valor de la función de coste es óptimo y, a partir de ese momento, apenas variaría el valor de los parámetros de la red. Sin embargo, se puede observar cómo el mínimo global de la función tiene un valor muy inferior.

Por ello es muy importante seleccionar una buena función de coste según el uso que se le quiera dar a la red.

Una opción sería usar una función computacionalmente sencilla, como es la suma de diferencias de cuadrados. A esta operación también se le conoce como el cálculo del *Error Cuadrático Medio*.

$$\boldsymbol{y = \frac{1}{N}\sum_{i=1}^{N}(y - a)^2} \qquad (9)$$

Siendo *N* el número total de elementos que contiene el set de entrenamiento, *y* el *ground truth* y *a* el resultado obtenido a la salida de la red.

En el caso de la función del "Error Cuadrático Medio", para conseguir disminuir el coste de la red hay que optimizar la función $z = (y - x)^2$, cuya representación gráfica en 3D es la siguiente:

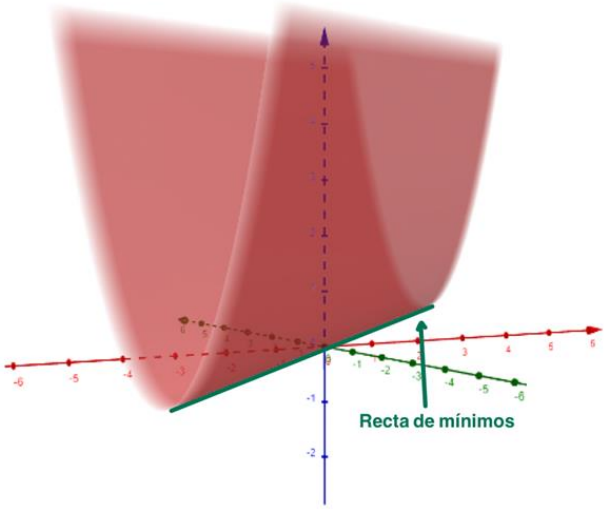


Imagen 26 - Representación gráfica de la función $z = (y-x)^2$

En la gráfica se puede ver cómo esta función tiene forma de parábola y, por tanto, tiene un único mínimo local, lo que permite que la optimización sea más sencilla.

Además del cálculo del "Error cuadrático Medio" también hay otras funciones de coste más complejas como es el caso de la función de coste logarítmica, cuya expresión matemática es la siguiente:

$$\boldsymbol{y = \frac{1}{N} \sum_{i=1}^{N} y \cdot log(a) + (1 - y) \cdot log\,(1 - a)} \qquad (10)$$

De nuevo, *N* es el número total de elementos que contiene el set de entrenamiento, *y* el *ground truth* y *a* el resultado dado por la red.

Esta función es muy útil en tareas de clasificación binaria. Como ya se ha mencionado, en los problemas de clasificación la salida de la red corresponde la probabilidad de que un objeto pertenezca a una clase determinada, por lo que el valor de *a* siempre estará comprendido entre 0 y 1. De la misma manera, el valor de *y* será 1 si un objeto pertenece a una clase determinada y 0 si no pertenece.

En el caso de la función de coste logarítmica, para conseguir disminuir el coste de la red hay que optimizar la función $z = \mathrm{y} \cdot \log(\mathrm{a}) + (1 - \mathrm{y}) \cdot \log(1 - \mathrm{a})$, cuya representación gráfica en 3D es la siguiente:

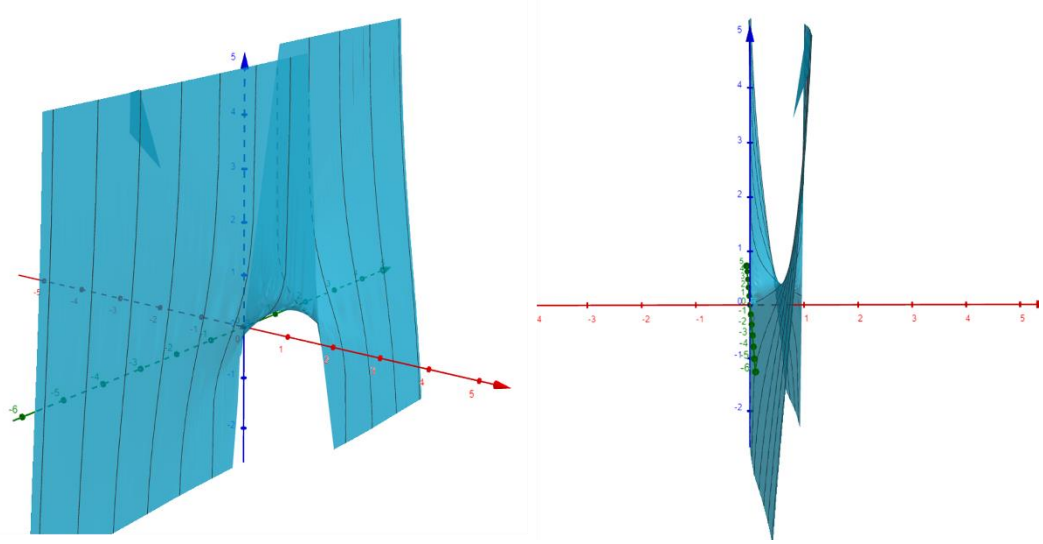


Imagen 27 - Representación gráfica de dos vistas de la función z = y·log(a) + (1-y)·log(1-a)

Sin embargo, como ya se ha mencionado no se debe considerar todo el dominio de la función. Si se representa la función limitando lo valores de x e y entre 0 y 1 queda la siguiente función:

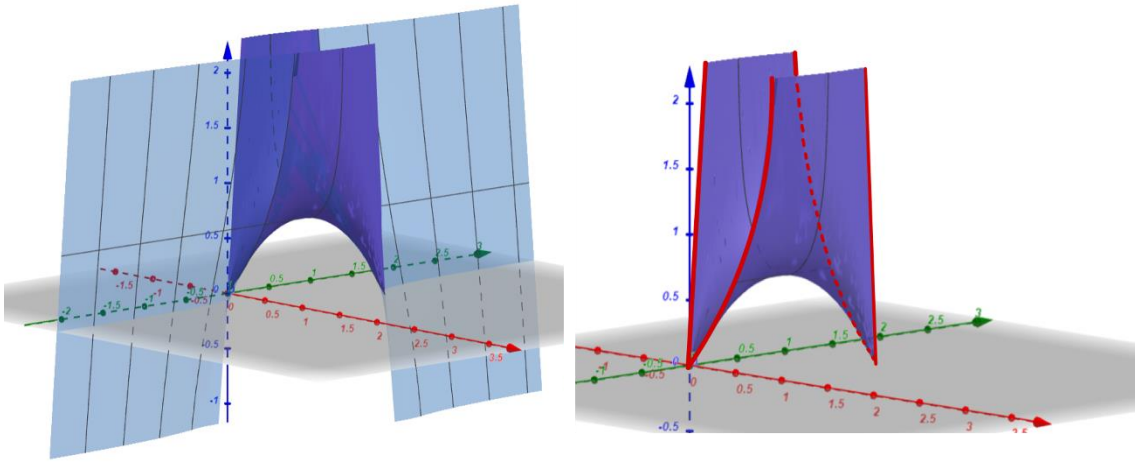


Imagen 28 - Representación gráfica de la función z = y·log(a) + (1-y)·log(1-a) con x,y ∈ [0,1] (En la izquierda la función completa en azul y la acotada en morado y a la derecha la función acotada con los valores que puede tomar la función de coste en rojo)

En la gráfica se puede ver que la función acotada tiene dos mínimos locales a la misma altura, por lo que, dependiendo del valor que tome la *y* (0 o 1) en cada caso, la función deberá tender a un mínimo u otro.

No obstante, para poder disminuir lo máximo posible el error de la red, no sólo es necesario elegir una buena función de coste, sino que también es necesario elegir un valor adecuado para la velocidad de aprendizaje o *learning rate*.

Todo el proceso descrito hasta ahora indica si se debe aumentar o disminuir el valor de los parámetros, pero no indica en qué cantidad se debe modificar. El *learning* rate es el que indica la magnitud de este cambio mediante la siguiente expresión matemática:

$$w = w + \alpha \frac{\delta J}{\delta w} \quad (8)$$

siendo α el *learning rate*, w el valor del parámetro a modificar, J la función de coste y $\delta J/\delta w$ la derivada parcial de J con respecto a w.

Un *learning* rate muy grande implica variaciones muy bruscas de los parámetros w y b lo que causaría que el valor de la función de coste oscilase de un lado a otro del punto óptimo, provocando que en algunas ocasiones este valor nunca se llegase a alcanzar. Por el contrario, en caso de que la función de coste tuviera varios mínimos relativos sería más sencillo que se pudieran evitar.

De la misma manera, si el *learning rate* es muy pequeño, llevará mucho tiempo alcanzar el mínimo absoluto de la función de coste y podría ocurrir que se llegase a un mínimo relativo y no se pudiera salir de él. En cambio, si no se entra en ningún mínimo relativo el valor de la función de coste se acabaría aproximando mucho a su valor óptimo.

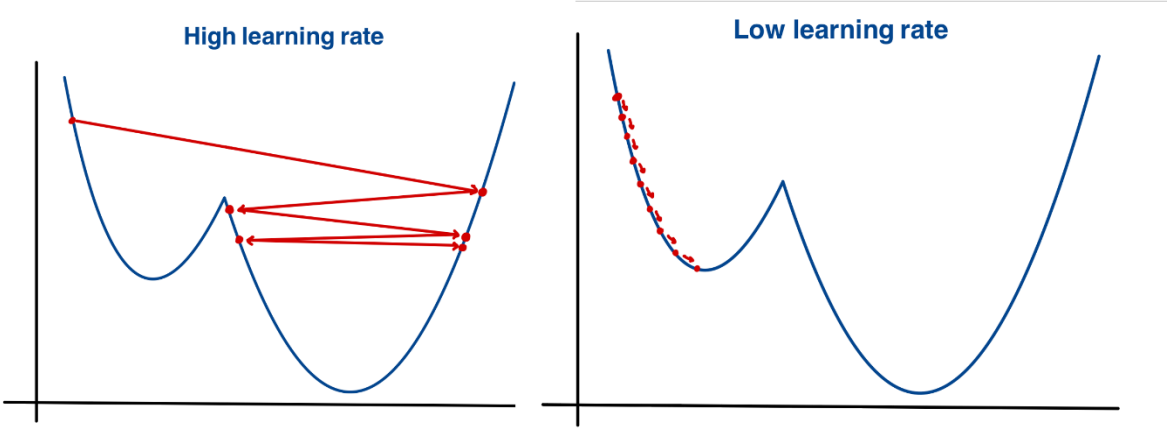


Imagen 29 - Elección de un "learnnig rate" alto (izquierda) y bajo (derecha)

2.1.1. Deep learning y redes neuronales convolucionales

Todo lo que se ha viso en el punto anterior es el funcionamiento general de las redes neuronales, sin embargo, no todas las redes son iguales. Existe una campo dentro de las redes neuronales, llamado *deep learning* o "aprendizaje profundo", que se encarga del estudio de las *redes neuronales profundas*.

Las redes neuronales profundas únicamente se diferencian de las redes neuronales clásicas por el número de capas ocultas que tiene la red, de tal forma que **una red neuronal es profunda si tiene más de tres capas ocultas** [13]**.**

Dentro del d*eep learning*" también existen varios tipos de redes neuronales profundas. Unas de las más utilizadas hoy en día son las Redes Neuronales Convolucionales. (CNN por sus siglas en inglés – *Convolutional Neural Networks*).

Las redes neuronales convolucionales se usan fundamentalmente en tareas relacionadas con c*omputer vision*, porque su arquitectura permite que tengan un muy buen desempeño en este tipo de problemas.

Esto se debe a que este tipo de redes están compuestas por dos partes claramente diferenciadas: un conjunto de capas convolucionales y una etapa final que suele ser una red neuronal f*ully connected*, pero que puede ser otro tipo de algoritmo de *machine learning*.

- Conjunto de capas convolucionales: Esta sección de la red se encarga de extraer distintas características presentes en los datos que la red recibe como entrada. Estos datos normalmente son imágenes o vídeos.
  Cada capa convolucional se centra una serie de características concretas de los datos. Para ello, cada capa convolucional está compuesta de los siguientes elementos (dado que es el caso más común, se supondrá que la entrada de la red es una imagen):

1. Cálculo de la convolución de una imagen con un filtro

   Una convolución es una operación matemática en la que un filtro se va desplazando a lo largo de una entrada [14]. De esta manera, se puede ir extrayendo información de distintas secciones de los datos de entrada para poder luego analizar de forma conjunta toda la información extraída. En el caso de las redes neuronales convolucionales este filtro suele tener el aspecto de una matriz y es esta matriz la que debe ir ajustando esta parte de la CNN para ser capaz de detectar las características de la imagen útiles para la red.

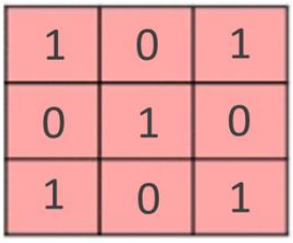


Imagen 30 - Ejemplo de filtro de una capa convolucional

Este filtro de dimensiones n x m (en este caso de 3x3) se irá desplazando a lo largo de la imagen de izquierda a derecha y de arriba a abajo cubriendo cada vez un conjunto de píxeles distinto. Cada vez que el filtro cubra un conjunto de píxeles se generará un valor numérico que será un término de la matriz obtenida como resultado de la convolución.

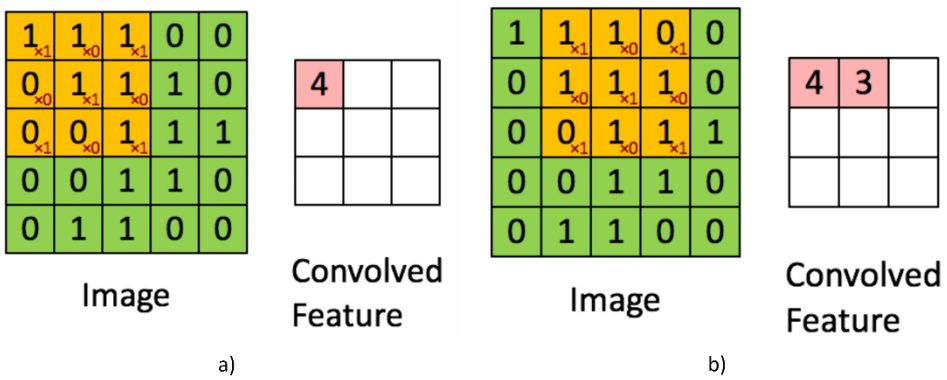


Imagen 31 – Calculo del primer (a) y el segundo (b) término de una convolución entre una matriz de entrada (en verde) y un filtro que cubre la región representada en amarillo (Fuente: [28])

2. Reducción de los datos obtenidos con la convolución o *pooling*

   El *pooling* es una técnica utilizada para reducir el tamaño de la información con la que se va a trabajar. En este caso, las matrices resultado de las convoluciones realizadas suelen tener un tamaño considerable, por lo que, para reducir el coste computacional del resto de capas de la red se elimina la información menos útil.
   Hay dos tipos principales de *pooling*: *average pooling* y *max pooling*. En ambos casos la matriz que se quiere procesar se divide en distintas regiones y se hace un cálculo sencillo con los números contenidos en esa región.

   En el *average pooling* se calcula la media de los valores contenidos en cada región, sin embargo, en el *max pooling* se escoge como resultado el valor más grande de entre todos los de esa región.

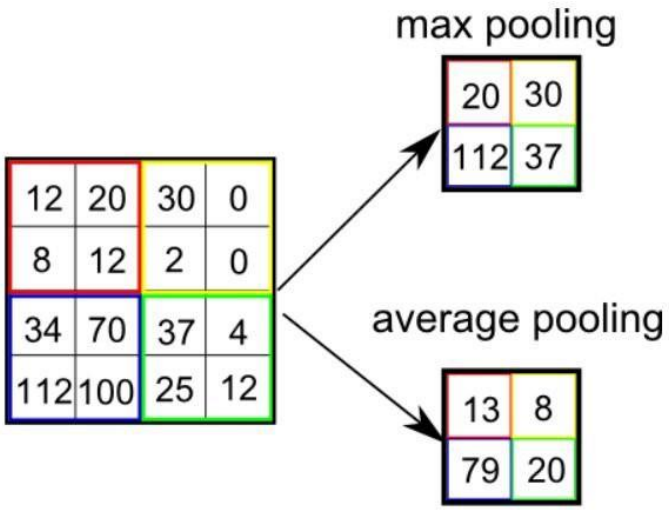


Imagen 32 - Cálculo del "average pooling" y "max pooling" (Fuente: [28])

Cada capa convolucional genera lo que se conoce como mapa de características. Según la profundidad a la que se encuentre cada capa convolucional, el significado semántico y el tamaño de su mapa de características será mayor o menor. De esta forma, a mayor profundidad de una capa convolucional, mayor significado semántico y menor tamaño del mapa de características.

Esto quiere decir que, mientras que las primeras capas convolucionales pueden detectar características más simples como los bordes presentes en la imagen, las capas más profundas pueden detectar formas concretas a partir de esos bordes y determinar a qué elemento corresponde una forma concreta (ej.: en una imagen de un perro detectar que en una región de la imagen está una de sus patas).

Por tanto, en general, cuantas más capas convolucionales haya, más precisión habrá en la extracción de los datos.

- Etapa final: Esta parte de la red puede ser cualquier tipo de algoritmo de *machine learning*.

  En muchas ocasiones, se usa una red neuronal *fully connected*. El funcionamiento de estas redes es el que ha explicado en el apartado anterior. No obstante, tiene la peculiaridad de que todas las neuronas de una capa están conectadas a todas las neuronas de la capa siguiente.

  Sin embargo, las redes *fully connected* no son el único tipo de algoritmo utilizado. Hay otros algoritmos bastante conocidos, como, por ejemplo, el algoritmo *Feature Pyramid Network* o FPN. Las FPN utilizan los mapas de características generados por las distintas capas convolucionales para crear una pirámide de características, donde los mapas de características con menor significado semántico están en la base de la pirámide.

  Partiendo del mapa de características ubicado en la parte superior de la pirámide, se realiza una operación de extrapolación (que es la operación inversa al *pooling*) para intentar generar el mapa de características correspondiente al nivel inferior de la pirámide. Esta operación se realiza para todos los niveles de la pirámide y, como consecuencia, se genera un mapa de características con un significado semántico fuerte para cada uno de los niveles de la pirámide.

  Por último, en cada nivel de la pirámide, se combinan los mapas de características generados por la CNN y los generados a partir del proceso explicado en el párrafo anterior.
  Esto permite poder trabajar con mapas con un significado semántico fuerte sin tener que renunciar a los mapas de los niveles inferiores de la pirámide (que muchas veces son los que permiten la detección de objetos pequeños).

  Este procedimiento utilizado por las FPN, al igual que el procedimiento utilizado por las redes *fully connected*, se utiliza para procesar los datos con el objetivo de resolver la tarea asignada a la red.

Se podría decir que **el conjunto de capas convolucionales se encarga de hacer el preprocesado de los datos necesario para que esta parte de la red pueda solucionar el problema planteado** (clasificación, de detección de objetos, etc.).

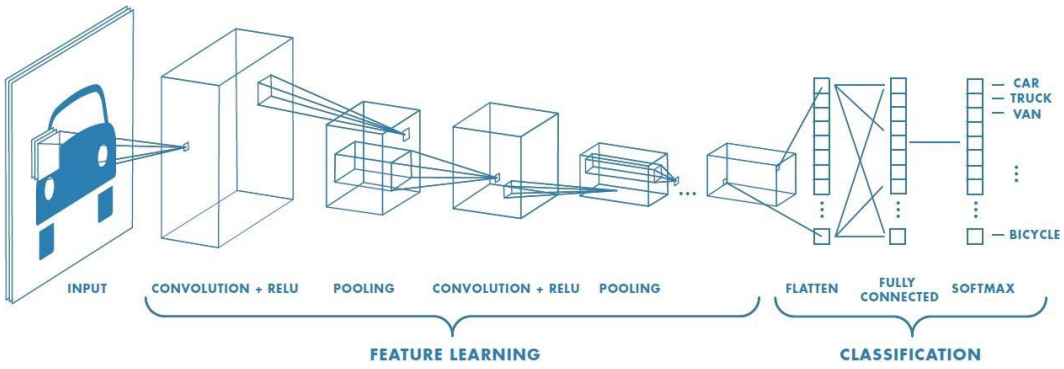


Imagen 33 - Estructura CNN (Fuente: [28])

2.5.1. Arquitecturas de las CNN [15]

La estructura que se ha visto en el apartado anterior es la estructura general de las redes neuronales convolucionales, sin embargo, no todas las CNN son iguales. Las capas convolucionales de las redes pueden tener distintos componentes y estructuras, esto es lo que define la arquitectura de una red neuronal convolucional.

Una de las arquitecturas utilizadas en tareas de c*omputer vision* es ResNeXt. Esta arquitectura está basada en otras tres arquitecturas de red distintas: VGG (*Visual Geometry Group*), Inception y ResNet. Cada una de estas tres arquitecturas sigue una filosofía distinta y ResNeXt recoge ciertas particularidades de cada una de ellas para conseguir una mejora de los resultados. Estas particularidades son las siguientes:

- Inception: Esta arquitectura de red neuronal trabaja con un concepto llamado "bloque". Un bloque consiste en un conjunto de convoluciones, con filtros que pueden tener tamaños distintos, aplicadas a un mismo conjunto de datos de entrada. Cada una de estas convoluciones recibe el nombre de "rama" del bloque. También suele haber una rama dedicada al *pooling*.

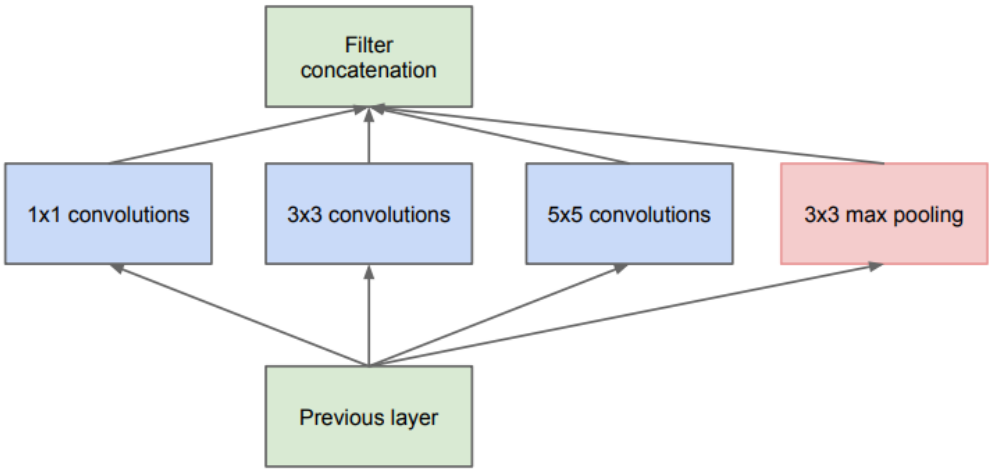


Imagen 34 - Bloque de Inception (Fuente: [15])

  En la arquitectura de red Inception cada bloque tiene una combinación de convoluciones distinta, optimizada para obtener las características de los datos de entradas que sean relevantes para solucionar el problema propuesto. ResNeXt es similar a Inception en que también utilizan distintas convoluciones dentro de un mismo bloque.

- VGG: Esta arquitectura de red también trabaja con bloques, sin embargo, a diferencia de Inception, todos sus bloques tienen una estructura similar. Además, los bloques no tienen distintas ramas, sino que las distintas convoluciones se aplican en secuencia. Esta es la filosofía que adopta ResNext puesto que de esta forma es más fácil que la red sea capaz de generalizar en vez de memorizar.

- ResNet: Esta última arquitectura también trabaja con bloques. En este caso los bloques son similares a los de VGG. Un bloque de ResNet tiene un esquema parecido al siguiente:

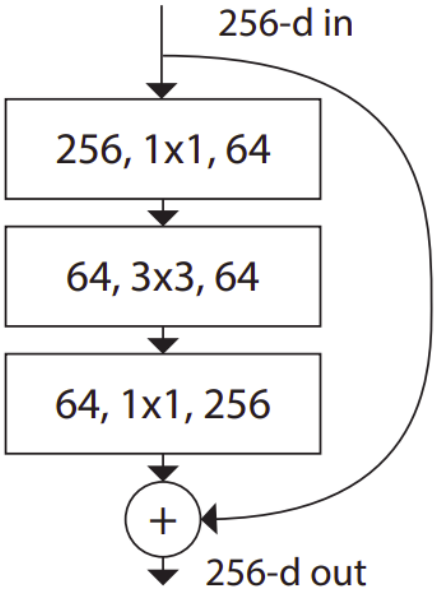


Imagen 35 - Esquema de un bloque de ResNet (Fuente: [29])

Los bloques de ResNet encapsulan tres tareas distintas: realizar convoluciones, modificar el tamaño de las matrices con las que trabaja la red (normalmente de filtro 1x1) y aplicar la técnica de *skip connections*, que consiste en pasarle a una capa la salida de otra capa no contigua como entrada.

Como ya se ha visto anteriormente, las convoluciones sirven para extraer características importantes de los datos de entrada, y cualquier red neuronal convolucional debe realizar este tipo de cálculos. Sin embargo, ninguna de las otras dos arquitecturas explicadas modifica el tamaño de las matrices de entrada, ni utiliza la técnica de *skip connections*.

Cada una de estas dos tareas aportan una ventaja distinta: la primera permite que la red neuronal sea más eficiente y más precisa en sus resultados, mientras que la segunda ayuda a que el aprendizaje sea más rápido y a que ciertas características que se recogieron en las primeras capas puedan ser tenidas en cuenta por capas posteriores sin ningún procesado entre medias [16].

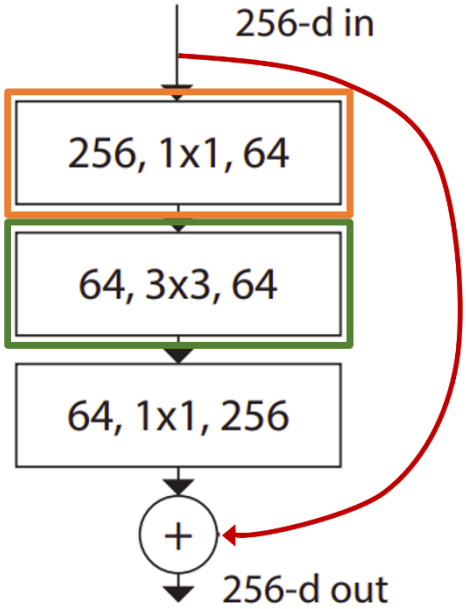


Imagen 36 - Estructura de un bloque ResNet que presenta capa de cambio de dimensiones (en naranja), de convolución 3x3 (en verde) y "skip connection" (en rojo)

Por tanto, la arquitectura de **red ResNeXt presenta: bloques con distintas ramas** (al igual que Inception), **con una estructura similar entre ellos** (como VGG) **y en los que cada uno de los bloques presenta las mismas tres funcionalidades que ResNet** (Realización de convolución, modificación del tamaño de matrices y *skip conections*) tal como se puede ver en la *Imagen 34*.

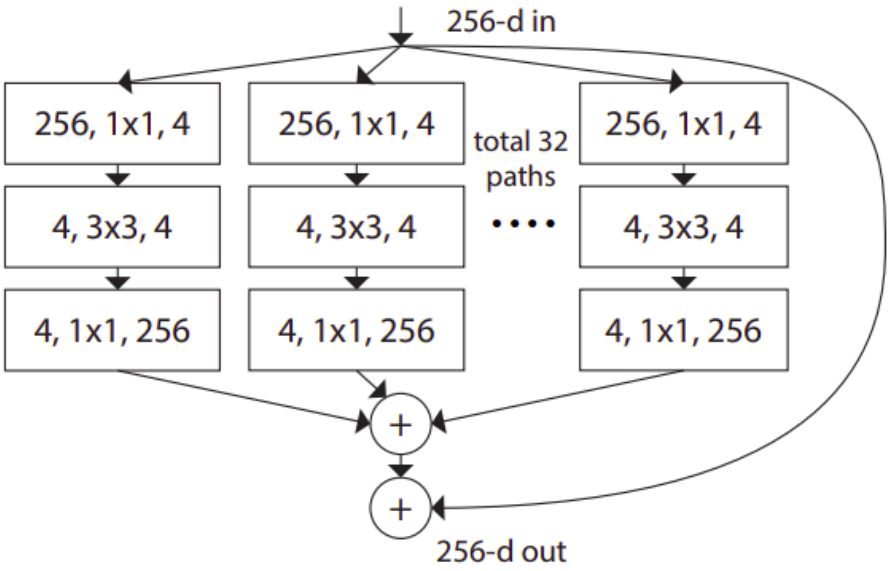


Imagen 37 - Estructura de un bloque de ResNeXt (Fuente: [29])

2.5.2. Algoritmos usados en las CNN [17]

La manera de resolver una tarea concreta usando CNNs no es única, sino que hay varios algoritmos que se pueden usar y que dan solución al problema de manera distinta. En el área de la detección de objetos cabe destacar los algoritmos R-CNN (CNNs basadas en regiones), Fast R-CNN, Faster R-CNN y Mask R-CNN.

- R-CNN (Region based CNNs)

  Como su propio nombre indica este tipo de algoritmos dividen las imágenes en regiones para poder trabajar con secciones de datos más pequeñas. Sin embargo, una imagen se puede dividir en muchas regiones y cuántas más regiones se extraigan de una imagen más lento será el algoritmo. Por eso R-CNN utiliza un método llamado "selección selectiva" que es capaz de, partiendo de muchas regiones, ir combinando las que sean similares para quedarse únicamente con 2000 regiones. **Estas 2000 regiones seleccionadas se conocen como "regiones de interés".**

  Una vez que se han seleccionado las regiones de interés, éstas se redimensionan para convertirlas en una imagen cuadrada. Esta nueva imagen se pasa a una red neuronal convolucional que extrae las características presentes en ella. Finalmente, mediante la resolución de un problema de clasificación, un algoritmo SVM es el encargado de determinar si hay algún objeto de interés en las distintas regiones procesadas.

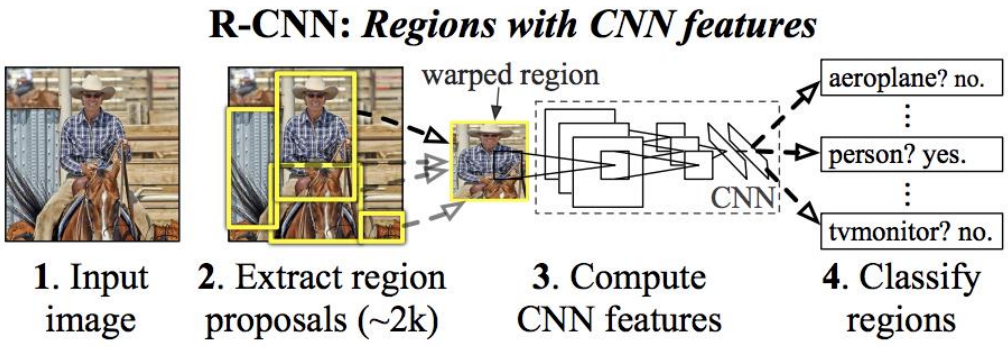


Imagen 38 - Fases de un algoritmo R-CNN (Fuente: [17])

Este algoritmo, sin embargo, tiene el inconveniente de que tanto el entrenamiento de la red como el proceso de detección de los objetos de interés presentes en una imagen llevan una gran cantidad de tiempo.

- Fast R-CNN

  Este algoritmo es prácticamente igual a R-CNN, pero difiere en el momento en el que se determinan las regiones de interés.

El proceso que se lleva a cabo es el siguiente:

1. Se le pasa a la CNN la imagen completa que se quiere procesar.
2. La CNN saca el mapa de características de la imagen.
3. Con el mapa de características obtenido se determinan las distintas regiones de interés de la imagen mediante de método de "selección selectiva".
4. Cada una de estas regiones se transforma para generar una imagen cuadrada y, usando la técnica de "*pooling*", se redimensiona para ajustar su tamaño a la entrada de la red que la va a procesar.
5. Las regiones redimensionadas se pasan a una red neuronal "*Fully Connected*". Esta red es la encargada de determinar si el objeto contenido en una región es de interés y, en su caso, la clase a la que pertenece.

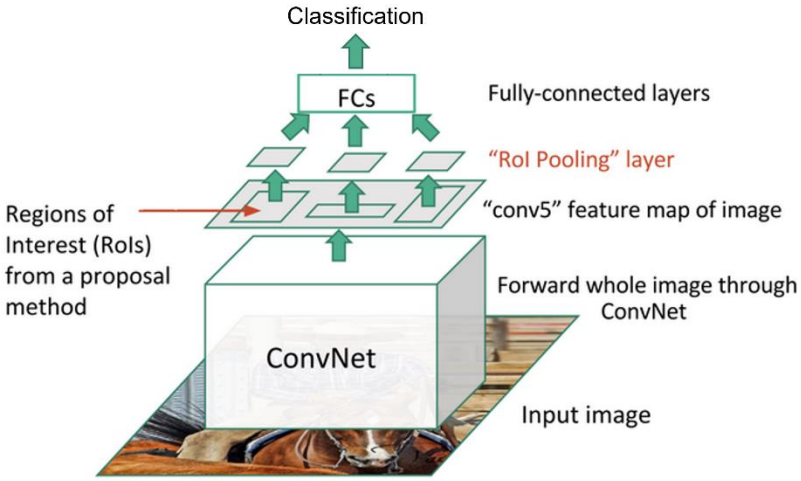


Imagen 39 - Fases del algoritmo Fast R-CNN

De este modo, mientras que en R-CNN las regiones de interés se determinan antes de que la CNN procese los datos, el algoritmo Fast R-CNN lo hace justo después. Esto permite que la CNN solo tenga que ejecutarse una vez y que, por tanto, el algoritmo Fast R-CNN sea considerablemente más rápido que el algoritmo R-CNN.

- Faster R-CNN

Este algoritmo también se basa en su antecesor (Fast R-CNN en este caso). En este caso la diferencia viene dada por la forma en la que se calculan las regiones de interés.

Como ya se ha mencionado, el algoritmo Fast R-CNN usa el método "selección selectiva" para determinar las regiones de interés, sin embargo, el algoritmo Faster R-CNN utiliza una red neuronal adicional para esta tarea.

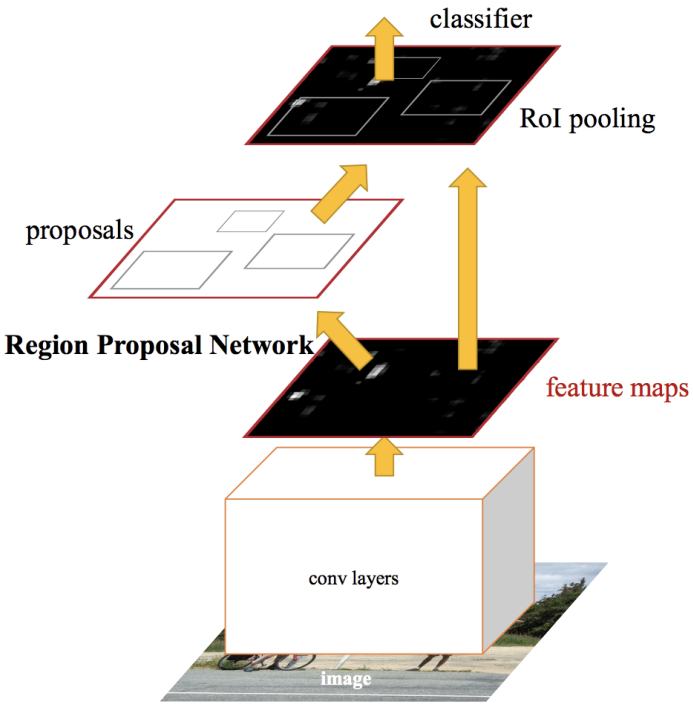


Imagen 40 - Estructura del algoritmo Faster R-CNN (Fuente: [17])

Esto permite que las redes que implementen el algoritmo Faster R-CNN tengan la capacidad de aprender a proponer las regiones de interés óptimas, mientras que con el método "selección selectiva" no es posible llevar a cabo ningún aprendizaje debido a que este método es estático.

- Mask R-CNN

Este algoritmo es, en esencia, el mismo que Faster R-CNN, puesto que se implementa de la misma manera. La única diferencia que hay entre estos dos algoritmos consiste en que con Mask R-CNN se obtiene a la salida una máscara del objeto detectado, además de la clase a la que pertenece y su "*Bounding Box*".

3. Especificaciones y restricciones de diseño

El objetivo principal de este proyecto es el desarrollo de un sistema basado en redes neuronales que sea capaz de detectar y realizar un conteo de los rodaballos y grupos de rodaballos que contengan las imágenes que se le pasen como entrada.

3.1. Especificaciones de diseño

- El sistema diseñado utiliza redes neuronales convolucionales para implementar una solución con los métodos de la rama de conocimiento c*omputer vision*, perteneciente a la Inteligencia Artificial.
- El sistema devuelve una representación gráfica de los rodaballos y grupos de rodaballos detectados, junto a un valor numérico correspondiente al número de rodaballos y grupos de rodaballos detectados.
- El sistema desarrollado parte de un entrenamiento previo para detección de objetos. Por lo tanto, se ha hecho uso de la técnica de t*ransfer learning*.
- El sistema no ha sido entrenado para trabajar con imágenes que no se correspondan con imágenes de rodaballos en un tanque de una piscifactoría.

3.2. Especificaciones técnicas

Para el desarrollo de este proyecto se han utilizado las siguientes herramientas software:

- Python 3.7 (entrenamiento de la red) y Python 3.9.13 de 64 bits (formateo de datos)

  Lenguaje de programación de alto nivel interpretado, de tipado dinámico y que busca tener una sintaxis fácilmente legible. Este lenguaje soporta la orientación a objetos, pero también se puede utilizar para una programación imperativa.

- Torch 1.10.0

  Librería de código abierto orientada al aprendizaje automático que facilita tanto el proceso de creación de redes neuronales como la implementación del proceso de aprendizaje correspondiente.

- CUDA para Torch 1.10.0

  CUDA ("*Compute Unified Device Arquitecture*") es una plataforma de computación en paralelo desarrollada por NVIDIA que permite codificar algoritmos para ejecutarlos en un GPU de NVIDIA. Esto permite aumentar la rapidez de la ejecución de operaciones vectoriales, algo que es muy útil en las redes neuronales debido a que gran parte de las operaciones realizadas son de este tipo.

- Google Colab

  Es un producto de Google Research que permite a cualquier usuario escribir y ejecutar código escrito en Python en un navegador. El código se ejecuta en máquinas virtuales alojadas en los servidores de Google. Por este motivo, se desconoce el hardware específico de los equipos que ejecutan el código de entrenamiento de la red, a excepción de la GPU que es una GPU Tesla P100.

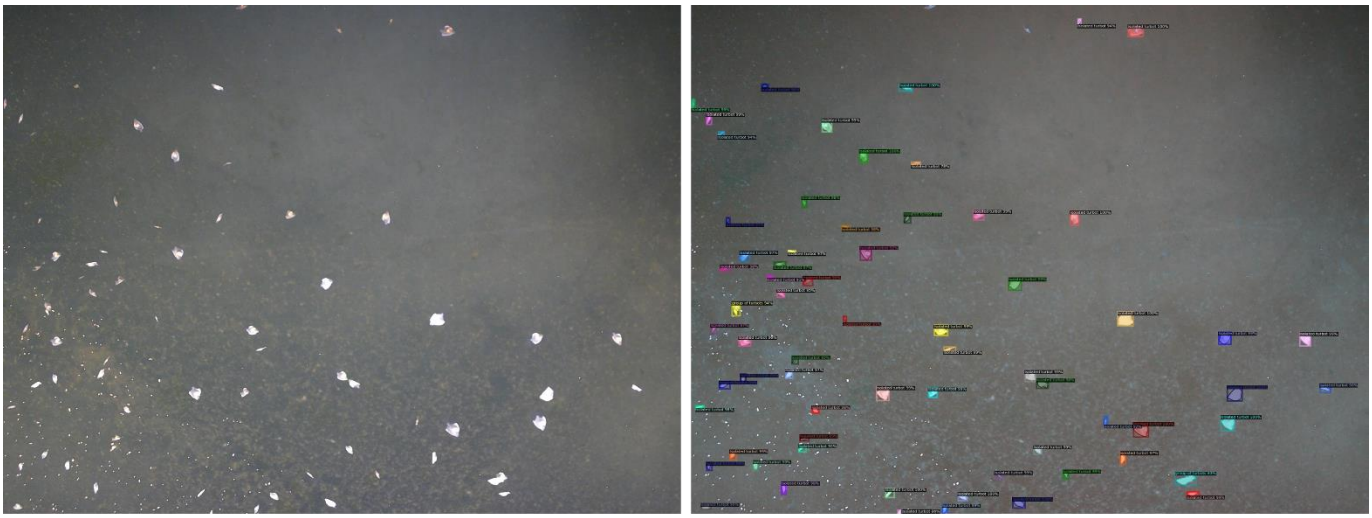
3.3. Restricciones de diseño

- La cantidad de datos disponibles para la fase de entrenamiento es factor especialmente limitante, ya que tener una gran cantidad de datos para entrenamiento permite que a la salida de la red se obtengan resultados mucho mejores.
- El tiempo disponible tanto para el etiquetado como para la realización de las pruebas de entrenamiento del sistema ha sido limitado, lo que ha provocado que el número de pruebas realizadas también haya sido limitado.
- Se ha utilizado una versión gratuita de Google Colab, lo que ha supuesto que el tiempo de uso de la GPU asignada por Google fuese limitado.
- La aplicación necesita del hardware y software mencionados anteriormente para su desarrollo y despliegue.
- El etiquetado de las imágenes no ha sido realizado por personal profesional, por lo que los objetos más complicados de etiquetar pueden haber sido etiquetados erróneamente.

4. Descripción de la solución propuesta

En este proyecto se ha desarrollado un sistema capaz de realizar un conteo de rodaballos en tanques de piscifactorías. Para ello, se ha implementado un algoritmo de segmentación de instancia. Este tipo de algoritmos, como se ha visto anteriormente, devuelven la ubicación, la clase de los distintos objetos detectados en la imagen y una máscara binaria de cada objeto detectado.

Este sistema recibe como entrada una imagen de una sección de un tanque y a la salida, gracias a la información obtenida de la red, es capaz de indicar qué objetos han sido detectados como rodaballos.



a) b)

Imagen 41 - Ejemplo de imagen introducida a la entrada (a) y de imagen obtenida a la salida de la red (b)

En tareas de *computer vision*, como es la segmentación semántica, las redes neuronales convolucionales suelen obtener muy buenos resultados. Por ello, se ha trabajado con una CNN publicada por la empresa Meta que usa un algoritmo Mask R-CNN (*Region based Convolutional Neural Network*) y una arquitectura ResNeXt.

Esta red neuronal sigue el modelo de aprendizaje supervisado, esto quiere decir que para entrenar la red se necesitan tanto características como etiquetas. Las características se obtienen a partir de las imágenes que se le pasan como entrada a la red. Las etiquetas, sin embargo, en este caso se han tenido que generar manualmente.

Esta ha sido la primera fase del proyecto.

Al terminar esta primera fase, ya se disponían de todos los datos necesarios para entrenar la red, sin embargo, el formato de los datos no era el adecuado. Por esta razón la segunda fase del proyecto ha consistido en reformatear los datos obtenidos en el etiquetado de las imágenes.

Por último, la tercera fase del proyecto ha consistido en entrenar la red convolucional con un total de 157 imágenes (128 imágenes para el set de entrenamiento y 29 para el de validación) de las cuales 60 han sido etiquetadas en el contexto de este proyecto.

Fase 1: Etiquetado de las imágenes

La función principal del etiquetado es indicarle a la red cuál es la respuesta correspondiente a cada entrada. Por lo tanto, en un algoritmo de segmentación semántica el etiquetado contiene tres tipos de datos para cada objeto de interés presente en la imagen:

- El conjunto de puntos que componen el borde de un objeto
- Su "*Bounding Box*", es decir, el rectángulo más pequeño que contiene a todo el objeto
- La clase a la que pertenece. En este proyecto se han determinado 2 clases (rodaballo aislado y grupo de rodaballo) y a cada una de estas clases se le han asignado varias etiquetas en función de la seguridad de la respuesta dada.

En el desarrollo del proyecto el etiquetado se ha llevado a cabo mediante una aplicación codificada en Matlab que no ha sido desarrollada en este proyecto.

Esta aplicación recibe como entrada un vídeo, segmenta todos los objetos de uno de sus fotogramas y va mostrando objeto a objeto para que el usuario asigne una etiqueta.

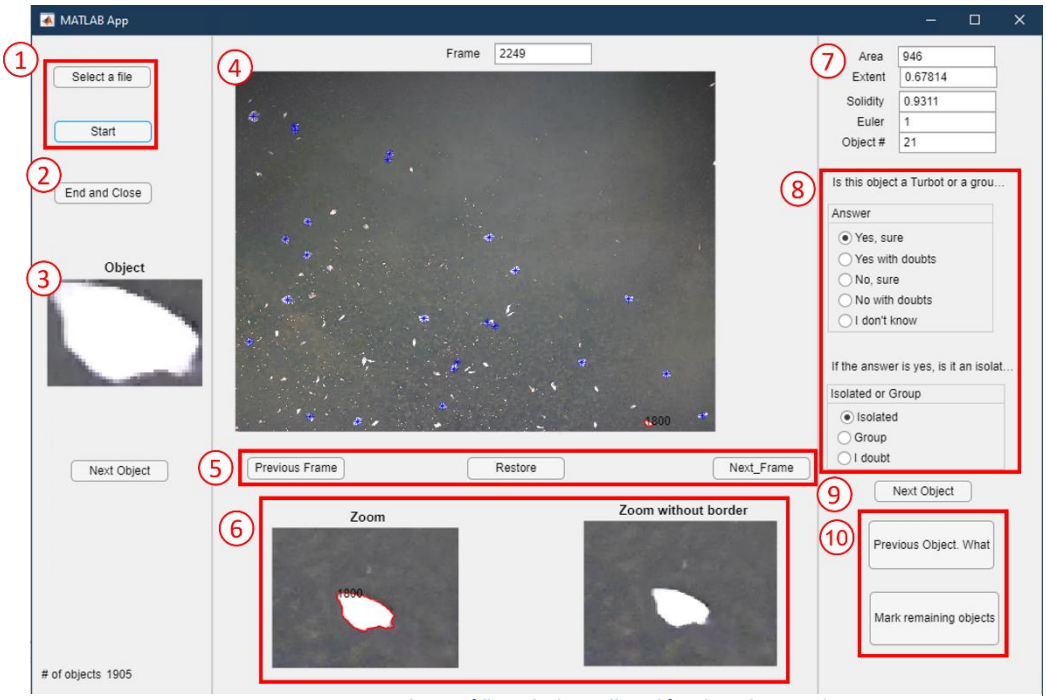


Imagen 42 - Interfaz gráfica de la aplicación de etiquetado

Las etiquetas que se pueden asignar son las siguientes:

- Es, seguro, un rodaballo o grupo de rodaballos
- Se cree, sin estar seguro del todo, que es un rodaballo o un grupo de rodaballos
- No se sabe si es o no es un rodaballo o grupo de rodaballos

- Se cree, sin estar seguro del todo, que no es un rodaballo o un grupo de rodaballos
- Seguro que no es un rodaballo ni grupo de rodaballos

Además, si se seleccionan algunas de las dos primeras opciones, se debe indicar si el objeto que se está etiquetando es un rodaballo, un grupo de rodaballos o si no se sabe. Por último, en caso de que el objeto se etiquete como grupo de rodaballos se debe seleccionar el centro de cada uno de los ellos en el recuadro "Object".

Por tanto, lo primero que se debe hacer es seleccionar el fichero de vídeo que se quiere etiquetar ①. La aplicación selecciona el fotograma que se encuentre a la mitad del video y lo muestra en el recuadro central ④.

A continuación, se debe iniciar el proceso de etiquetado ①. En ese momento se procesa el fotograma seleccionado y se segmentan todos los objetos que contiene. Algunos de estos objetos pueden ser rodaballos, sin embargo, otros también pueden ser comida, burbujas u otro tipo de elementos que no son de interés. Por esta razón es necesario etiquetar las imágenes que se van a usar para entrenar a la red: para poder indicarle qué elementos de cada imagen son rodaballos o grupos de rodaballos.

Al segmentar los distintos objetos de la imagen el programa los va numerando automáticamente de arriba abajo y de izquierda a derecha, de tal forma que los números más bajos corresponden a los objetos más cercanos a la esquina superior izquierda mientras que los mayores números son asignados a los objetos cercanos a la esquina inferior derecha de la imagen. Sin embargo, los objetos no se van mostrando en este orden, sino que la aplicación ordena los elementos segmentados de mayor a menor tamaño. Esto permite etiquetar primero los objetos que tengan más probabilidad de ser rodaballos y poder etiquetar todos los objetos restantes como "No es un rodaballo. Seguro" cuando éstos ya sean demasiado pequeños para ser peces ⑩.

Cada uno de estos objetos segmentados se muestra en tres recuadros distintos de la aplicación:

- El recuadro "Object" ③: este recuadro muestra el objeto aumentado al máximo, de tal forma que justamente se vea el objeto entero.
- Los recuadros "Zoom" y "Zoom without border" ⑥: estos recuadros muestran una imagen aumentada del objeto en la que también se puede observar parte del entorno que lo rodea. Ambos recuadros muestran la misma imagen con la diferencia de que en el recuadro "Zoom" se indica cuál es el objeto de interés, rodeándolo con una línea roja.
  Sin embargo, cada uno de estos recuadros tiene una función distinta:
  - El recuadro "Zoom": permite saber qué objeto se está etiquetando cuando en este recuadro se muestra más de un objeto.
  - El recuadro "Zoom without border": permite tener una vista más clara del objeto de interés, ya que en ocasiones la línea roja mostrada en el recuadro "Zoom" dificulta determinar la clase del objeto etiquetado.

En ciertas ocasiones, como ocurre en la *Imagen 42*, es sencillo determinar la etiqueta que se le debe asignar a un objeto, sin embargo, otras veces es complicado distinguir un rodaballo de otros

elementos, o incluso saber cuántos rodaballos forman un grupo. Por ello, la aplicación permite avanzar y retroceder en los fotogramas del vídeo para poder realizar un etiquetado más preciso al ver el objeto en movimiento ⑤.

Según se van etiquetando cada uno de los objetos, mostrando el siguiente objeto ⑨. Sin embargo, si alguna etiqueta se ha asignado de forma errónea y se quiere modificar, se puede retroceder al anterior objeto para volver a etiquetarlo. De esta forma, se le puede asignar una etiqueta nueva y corregir el etiquetado del objeto ⑩.

Al terminar de etiquetar todos los objetos de deben guardar todos los datos obtenidos en un fichero de extensión .mat. Este proceso de guardado de datos también se debe realizar si se quiere interrumpir el proceso de etiquetado para retomarlo más adelante ②.

Este fichero en el que se vuelcan los datos del etiquetado almacena 7 variables distintas. Algunas de estas variables contienen un solo valor numérico, mientras que, otras contienen estructuras matriciales más complejas.

Las variables almacenadas en los ficheros .mat son las siguientes:

- iFrameS: almacena el número del fotograma que se ha etiquetado.

- nS: número total de objetos segmentados por el programa. Este es, por tanto, el número total de objetos que se deben etiquetar.

- countObject: número del último objeto etiquetado. Este dato es de utilidad cuando en un momento dado se interrumpe el etiquetado y posteriormente se quiere retomar.

- LS (matriz de etiquetas): matriz del mismo tamaño que el fotograma que se está etiquetando en la que se indican los píxeles que pertenecen a cada objeto. Para ello, a todos los píxeles que pertenecen al fondo de la imagen se les asigna el valor 0, ya los píxeles que pertenecen a un objeto segmentado se les asigna el número correspondiente a ese objeto (según la numeración de arriba a abajo y de izquierda a derecha mencionada anteriormente).

1	1	1	0	0
1	1	0	0	3
1	0	0	3	3
0	0	0	0	3
2	2	0	0	0

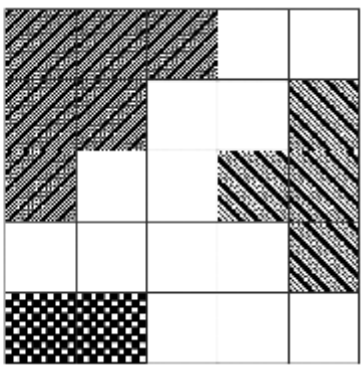


Imagen 43 - Ejemplo de matriz de etiquetas

- BS: vector que contiene los puntos que forman el borde de cada uno de los objetos segmentados por el programa. Esta variable es la que tiene al estructura más compleja, ya que cada uno de los elementos de este vector es una matriz de dos columnas que contiene la coordenada x y la coordenada y de todos los puntos que forman el borde de un objeto.

- mrProps: matriz de 11 filas que contiene tres bloques de información para cada uno de los objetos segmentados. En esta matriz los objetos están ordenados de mayor a menor tamaño:
    - Fila 1: número de objeto que asigna el programa según el orden que ya se ha visto anteriormente (de arriba abajo y de izquierda a derecha).
    - Filas 2-10: características morfológicas del objeto mostradas en la interfaz gráfica del programa ⑦ (*Imagen 42).*
    - Fila 11: etiqueta asignada al objeto. Hay 9 etiquetas distintas, cada una representada por un valor numérico. Esta asignación es de la siguiente manera:

        Etiqueta = 1 – No se sabe si es o no es un rodaballo o grupo de rodaballos.

        Etiqueta = 2 – Seguro que no es un rodaballo ni grupo de rodaballos.

        Etiqueta = 3 – Se cree, sin estar seguro del todo, que no es un rodaballo o un grupo de rodaballos.

        Etiqueta = 4 – Es un rodaballo o un grupo de rodaballos seguro. Dentro de esta opción se cree que es un rodaballo aislado.

        Etiqueta = 5 – Es un rodaballo o un grupo de rodaballos seguro. Dentro de esta opción se cree que es un grupo de rodaballos.

        Etiqueta = 6 – Es un rodaballo o un grupo de rodaballos seguro. Pero no se sabe si es un grupo o un rodaballo aislado.

        Etiqueta = 7 – Se cree, sin estar seguro del todo, que es un rodaballo o grupo de rodaballos. Dentro de esta opción se cree que es un rodaballo aislado y no un grupo.

        Etiqueta = 8 – Se cree, sin estar seguro del todo, que es un rodaballo o grupo de rodaballos. Dentro de esta opción se cree que es un grupo de rodaballos y no un rodaballo aislado.

        Etiqueta = 9 – Se cree, sin estar seguro del todo, que es un rodaballo o grupo de rodaballos, sin embargo, no se sabe decir si es un grupo o un rodaballo aislado.

- turbotCentroidS: matriz de 3 columnas que contiene para cada rodaballo presente en la imagen (si el rodaballo pertenece a un grupo se muestra una fila por cada uno de los peces pertenecientes al grupo):
    - Número de objeto según la numeración de izquierda a derecha y de arriba abajo.

- Si el objeto se ha etiquetado como rodaballo aislado: coordenada x del centroide del objeto.
  Si el objeto se ha etiquetado como grupo de rodaballos: coordenada x del punto seleccionado manualmente.

- Si el objeto se ha etiquetado como rodaballo aislado: coordenada y del centroide del objeto.
  Si el objeto se ha etiquetado como grupo de rodaballos: coordenada y del punto seleccionado manualmente.

Una vez recogida esta información **para cada fotograma segmentado** se deben formatear los datos a un formato que entienda la red neuronal.

Fase 2: Formateo de datos

La red neuronal se ha entrenado con las imágenes que contiene la base de datos COCO ("*Common Object in Context*") [18]. Esta base de datos tiene definido un formato de datos para los datos del etiquetado asociados a las imágenes que contiene conocido como COCO JSON ("*JavaScript Object Notation*") [19]. La red trabaja con una versión reducida de datos en este formato, pero también acepta datos con el formato COCO JSON oficial, ya que, para obtener los datos en el formato reducido simplemente debe procesar los campos que le interesan y eliminar los que no.

El formato de datos COCO JSON se basa, como su propio nombre indica, en el formato de datos JSON [20].

Un formato de datos no es otra cosa que la representación de la información en una estructura determinada.

En el caso del formato JSON se utiliza una estructura de datos de tipo clave-valor. Esto quiere decir que cada dato almacena en formato JSON debe tener una clave asociada. Por ejemplo, si se quiere almacenar el nombre de una persona en formato JSON, el dato se debería almacenar de una manera similar a la siguiente:

```
"nombre" : "Carlos"
```

Imagen 44 - Ejemplo par clave-valor JSON

Donde "nombre" es la clave asociada al valor "Carlos".

La existencia de una clave para cada valor almacenado es de gran utilidad, ya que permite obtener un dato concreto si se conoce la clave. En este caso, si se tiene un fichero con extensión .json con la asociación clave-valor mostrada más arriba, se podría solicitar recuperar el valor asociado a la clave "nombre" y se recibiría el valor "Carlos".

En este formato, las claves siempre deben ser cadenas de caracteres (y, por tanto, deben ir entre comillas), sin embargo, para los valores se permite usar otros tipos de datos como números enteros, decimales o booleanos (que pueden tomar el valor de True - verdadero - o False - falso).

No obstante, en un documento JSON no sólo se trabaja con parejas de clave valor tan sencillas como la que se ha mostrado en el ejemplo anterior. En JSON hay dos estructuras de datos complejas: los objetos y los *arrays*.

Los objetos son un conjunto de parejas de clave-valor que se pueden tratar como un solo dato. Se representan por medio de las llaves ({}) Por ejemplo, si se quieren almacenar los datos de una persona, se deberá guardar información como su nombre, su edad, su nacionalidad y un medio de contacto. Este último campo puede estar compuesto por un solo teléfono, un teléfono y un correo o incluso varios teléfonos y varios correos. Por tanto, el campo "medios de contacto" podría estar formado, a su vez, por varias parejas clave-valor, pero sigue siendo una unidad de información en sí misma. Este es el concepto de objeto en JSON.

La representación en formato JSON de todo lo descrito en el párrafo anterior quedaría de la siguiente manera:

```
{ "nombre" : "Carlos",
"apellido" : "Gómez",
"edad" : 43,
"medio de contacto" : {
    "tel1" : 917654309,
    "tel2" : 676549872,
    "email" : "cgomez@gmail.com"
    }
}
```

Imagen 45 – Ejemplo de caso de uso de un objeto en JSON

Los *arrays*, en cambio, están formados por un conjunto de valores que pertenecen a una misma clave y se representan por medio de los corchetes ([]). Por ejemplo, si una persona tiene como medio de contacto varios teléfonos, se pueden asociar todos los teléfonos a una misma clave de esta manera:

```
{ "nombre" : "Carlos",
"apellido" : "Gómez",
"edad" : 43,
"medio de contacto" : {
    "tel" : [917654309, 676549872],
    "email" : "cgomez@gmail.com"
    }
}
```

Imagen 46 - Ejemplo de caso de uso de un array en JSON

Esta estructura de datos JSON basada en parejas de clave-valor, objetos y *arrays* es la que utiliza el formato COCO JSON. Esto quiere decir que JSON y COCO JSON no presentan ninguna diferencia en cuánto a la forma de representar los datos, sino que su diferencia fundamental reside en que COCO JSON tiene un esquema. La función de este esquema es definir las parejas clave-valor que deben aparecer en el documento, y los tipos de datos que de cada uno de esos valores.

Este esquema se define de la siguiente manera:

```
{
   "info"              : info,
   "images"            : [image],
   "annotations"       : [annotation],
   "licenses"          : [license],
}

info{
   "year"              : int,
   "version"           : str,
   "description"       : str,
   "contributor"       : str,
   "url"               : str,
   "date_created"      : datetime,
}

image{
   "id"                : int,
   "width"             : int,
   "height"            : int,
   "file_name"         : str,
   "license"           : int,
   "flickr_url"        : str,
   "coco_url"          : str,
   "date_captured"     : datetime,
}

license{
   "id"                : int,
   "name"              : str,
   "url"               : str,
}
```

Imagen 47 - Esquema del formato COCO JSON (I) (Fuente: [19])

Como se puede ver en la *Imagen 47* el documento JSON tiene 4 claves principales: “info”, “licences”, “images” annotations”.

```
annotation{
   "id"                 : int,
   "image_id"           : int,
   "category_id"        : int,
   "segmentation"       : RLE or [polygon],
   "area"               : float,
   "bbox"               : [x,y,width,height],
   "iscrowd"            : 0 or 1,
}

categories[{
   "id"                 : int,
   "name"               : str,
   "supercategory"      : str,
}]
```

Imagen 48 - Esquema del formato COCO JSON (II) (Fuente: [19])

La clave “info” tiene como valor un objeto que contiene información básica sobre el documento JSON al que pertenece (fecha en el que se creó, versión del documento, descripción,…).

La clave “licences” tiene asociada un array de objetos del tipo “licence”. Con este tipo de objetos se busca definir una seria de licencias para luego indicar qué licencia tiene cada una de las imágenes que se han etiquetado.

La clave “images” tienen como valor un array de objetos “image”. Este tipo de datos tiene como objetivo proporcionar información sobre las imágenes que se han etiquetado. Los campos más importantes de este tipo de objetos son el campo “id” (su importancia se verá más adelante) y el campo “file_name”. Este último campo suele ser utilizado para poder descargar u obtener la imagen que se ha etiquetado. Por ejemplo, si el valor “file_name” es la url de la imagen, el programa que está procesando el fichero COCO JSON puede descargársela si es necesario. De la misma manera, si el campo “file_name” contiene el *path* donde está ubicada la imagen, ésta también se podrá obtener fácilmente. Los campos “Flickr_url” y “coco_url” no son obligatorios, y en este proyecto no se ha trabajado con ellas.

Por último, está la clave “annotations”, que es la más importante de todas. Contiene objetos del tipo “annotation” y contiene la información de cada uno de los objetos etiquetados en todas las imágenes indicadas en el array “images”. Por tanto, se puede ver fácilmente que cada anotación debe estar asociada a una imagen en concreto: esta es la función de la clave “image_id”. El valor de este campo será el valor del campo “id” de la imagen a la que pertenezca esta anotación. Por ejemplo, si una imagen llamada “image1.jpg” tiene asociado un “id” : 2, todas las anotaciones que pertenezcan a esa imagen deben tener el valor 2 en la clave “image_id” (“image_id” : 2).

Además de este campo, el resto de campos relevantes son: “segmentation”, “bbox” y “category_id”.

"Segmentation" contiene un array de los puntos que forman el borde del objeto al que pertenece la anotación. Este array tiene la siguiente estructura: [coor x_1, coor y_1, coor x_2, coor y_2, …].

"Bbox" contiene la *bounding box* del objeto correspondiente (como ya se ha comentado anteriormente una *bounding box* es el rectángulo más pequeño que es capaz de contener a todo el objeto en su interior). Su valor es un array que tiene la siguiente estructura: [coor superior izquierda x, coor superior izquierda y, ancho, alto].

Por último, el campo "category_id" contiene la etiqueta del objeto. En el documento COCO JSON hay otra clave llamada "categories" que contiene el nombre de todas las categorías existentes en ese documento y el valor numérico que se les ha asociado a cada una de ellas. El campo "category_id" de "annotation" contiene el valor numérico de la categoría a la que pertenece un objeto. De esta forma, si en un documento COCO JSON existe la categoría persona (id=1) y la categoría perro (id=2) y una anotación pertenece a un objeto de tipo perro, el valor de su clave "category_id" será 2.

De estos cuatro campos que componen el formato JSON, "info" y "licences" contienen información más general del documento. Por ello, para rellenarlos no se necesita ningún dato de los obtenidos con la aplicación de etiquetado de imágenes.

Sin embargo, para completar los campos "images" y "annotations" sí es necesario leer y formatear ciertos datos de los ficheros .mat generados durante el etiquetado de imágenes.

Para ello, se ha desarrollado un programa en Python que lee los ficheros .mat contenidos en un fichero, obtiene los datos necesarios para construir un fichero COCO JSON y los formatea debidamente.

En los campos "info" y "licences" casi toda la información que se debía introducir se ha escrito manualmente en el código debido a la imposibilidad de adquirir la información necesaria de una forma distinta.

```
#Info Section
info_dict = {
        "year": "2022",
        "version": "1",
        "description": "Turbot labeled images",
        "contributor": "Maria Castillo",
        "url": "",
        "date_created": str(datetime.datetime.now())
        }
```

Imagen 49 - Código de creación de la sección "info" en un documento COCO JSON

```
#license section
lisence_dict = [
    {
        "id": 1,
        "url": "https://creativecommons.org/licenses/by-nc-nd/4.0/",
        "name": "Attribution required, no commercial use and no derivative works"
    }
]
```

Imagen 50 - Código de creación de la sección "licences" en un documento COCO JSON

Sin embargo, para completar los campos “images” y “annotations”, ha sido necesario escribir un código más complejo para formatear los datos correctamente.

Para hacer el formateo de datos más sencillo se han creado dos directorios distintos. En el primero de ellos se han almacenado todas las imágenes etiquetadas, mientras que en el segundo se han almacenado los ficheros .mat generados para cada imagen contenida en el primer directorio.

En el caso del campo “image” se han ido obteniendo todas las imágenes en el primer directorio creado y se han ido completando los objetos correspondientes a cada una de ellas. Para ello, se han guardado en un array los nombres de los ficheros ubicados en este primer directorio que tuvieran como extensión .jpg.

Los campos de los distintos objetos “image” se han ido completando de la siguiente manera:

- Clave “id”: estos campos simplemente se han ido completando con números enteros consecutivos.

- Clave “file_name”: estos campos se han completado con cada uno de los nombres de los ficheros .jpg obtenidos.

- Clave “width”: para obtener el valor de esta clave ha sido necesario abrir cada una de las imágenes contenidas en el directorio de las imágenes etiquetadas para poder obtener el ancho de la imagen (en píxeles).

- Clave “heigth”: para obtener el valor de esta clave también ha sido necesario abrir cada una de las imágenes etiquetadas para poder obtener el alto de la imagen (en píxeles).

- Clave “licence”: a este campo se le ha dado el mismo valor para todas las imágenes, ya que su uso se debe limitar únicamente a este proyecto.

- Clave “date_captured”: este campo no es de interés para este proyecto, por lo que el valor asignado en todas la imágenes el “null” (es decir, sin contenido).

Además, en una variable del programa guarda la asociación <nombre de imagen – id de imagen> para cada imagen de la que se crea un objeto “image”. Esta variable permite determinar el campo “image_id” que se debe asignar a cada objeto “annotation”.

De la misma manera, para el campo “annotations” se ha obtenido un array con todos los nombres de los ficheros con extensión .mat contenidos en el otro directorio creado.

Partiendo de la información contenida en los ficheros .mat obtenidos, el formateo de datos se ha realizado de la siguiente manera:

- Campo “image_id”: como se ha comentado anteriormente, este dato se obtiene gracias a la variable que almacena la asociación <nombre de imagen – id de imagen> para cada imagen registrada en el campo “images”. Esto implica que para obtener el id de la imagen a la que pertenece una anotación, se debe obtener antes su nombre. Esto es sencillo, ya que el nombre de la imagen que pertenece a cada fichero .mat se puede obtener a partir del nombre del fichero .mat que tiene asociado. En concreto, el nombre de los ficheros .mat es la concatenación del nombre de la imagen que tienen asociada y "_c.mat”.

- Campo “category_id”: este campo tiene una relación directa con la etiqueta que se le ha asignado al objeto etiquetado. Concretamente, la etiqueta se debe obtener de la matriz “mrProps” del fichero .mat. Sin embargo, no tiene por qué haber una correspondencia directa entre el valor numérico de la etiqueta obtenido del fichero .mat y el id de la categoría asociado a ese tipo de objeto. Es decir, en un fichero .mat el valor 4 puede representar la etiqueta “Rodaballo aislado. Seguro.” mientras que, en el documento COCO JSON el tipo de objeto “rodaballo aislado” puede tener asociado el id de categoría 1.

- Campo “area”: este campo indica el área (en píxeles) del objeto etiquetado. En la matriz “mrProps” de los ficheros .mat la segunda fila contiene esta característica morfológica del objeto. Por tanto, solo es necesario obtener este dato y asignárselo como valor a esta clave.

- Campo “segmentation”: este campo está contiene los puntos que forman el borde del objeto etiquetado. Como ya se ha visto anteriormente el formato de este campo es el siguiente: [coor x_1, coor y_1, coor x_2, coor y_2, …]. En los ficheros .mat el vector columna “BS” contiene, en cada uno de sus elementos, otra matriz de dos columnas. Una de las columnas contiene la coordenada x mientas que la otra contiene la coordenada y de todos los puntos que formar el borden de un objeto etiquetado. Por tanto, el proceso que se ha seguido para formatear las coordenadas al formato COCO JSON es el siguiente:

```
obtener un elemento de la matriz BS
para cada fila de ese elemento
    para cada columna de esa fila
        concatenar valor obtenido a un array
```

Imagen 51 - Pseudocódigo del formateo del campo "segmentation"

- Campo “bbox”: este campo contiene la *bounding box* del objeto etiquetado. La estructura de este campo es [coor superior izquierda x, coor superior izquierda y, ancho, alto]. Sabiendo que la coordenada superior izquierda x se corresponde con la coordenada x más

pequeña y que la coordenada superior izquierda y con la coordenada y más pequeña, el formateo de los datos se puede hacer de la siguiente manera:

```
para cada fila de la matriz BS[i]
    para cada columnna de la matriz BS[i]
        si es la columna 1 (coordenada y)
            si el valor obtenido es menor que el valor min guardado anteriormente para coordenada y
                sobreescribir valor min coordenada y
            si el valor obtenido es mayor que el valor max guardado anteriormente para coordenada y
                sobreescribir valor max coordenada y
        si es la columna 2 (coordenada x)
            si el valor obtenido es mayor que el valor guardado anteriormente para coordenada x
                sobreescribir valor min coordenada x
            si el valor obtenido es mayor que el valor max guardado anteriormente para coordenada x
                sobreescribir valor max coordenada x

generar bbox con [min x, min y, max x-min x, max y-min y]
```

Imagen 52 – Pseudocódigo del formateo del campo "bbox"

- Campo "is_crowd": este campo solo puede tomar los valores 0 o 1. El valor 1 indica que el objeto etiquetado es un conjunto de muchos objetos que no pueden o quieren ser contados y el valor 0 indica el caso contrario. Por ejemplo, si el objeto etiquetado fuese una solo persona, el valor de "isCrowd" sería 0; pero si el objeto etiquetado fuese una muchedumbre en un estadio, el valor de "isCrowd" sería 1.

Fase 3: Entrenamiento de la red

Una vez que los datos de las imágenes etiquetadas se han obtenido en formato COCO JSON, ya se puede entrenar a la red neuronal. La red utilizada es una red neuronal convolucional programada por la empresa META llamada "Detectron2". Esta red tiene una arquitectura ResNeXt de 101 capas convolucionales, conectada a una FPN y usa el algoritmo Mask-RCNN.

A esta red hay que introducirle como datos de entrada las imágenes que se hayan etiquetado, junto con los datos de etiquetado en formato COCO JSON generados en la fase 2 del proyecto.

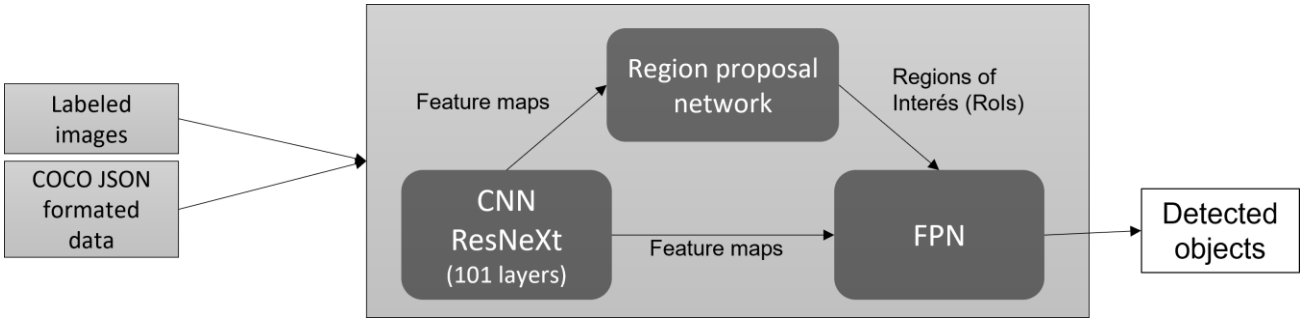


Imagen 53 – Arquitectura simplificada de la red Detectron2

Esta red neuronal es capaz de resolver una gran variedad de tareas, como la segmentación semántica, la segmentación panóptica o el *human pose estimation* (la detección de la postura que tiene un cuerpo humano en una imagen).

El objetivo de este proyecto es estimar la cantidad de peces presentes en un tanque de una piscifactoría partiendo de una imagen de uno de estos tanques. Sin embargo, para contar el número de peces presentes en una imagen primero hay que detectar los peces presentes en ella. Por ello, en este proyecto se hará uso de la funcionalidad de segmentación semántica que ofrece la red "Detectron2".

Para llevar a cabo esta tarea se ha partido de un código de ejemplo que se ha personalizado y modificado para obtener los datos de interés para el proyecto [21]. Los pasos implementados en el código de ejemplo que realiza una segmentación semántica de las imágenes son los siguientes:

1. Instalar los paquetes de Torch y Detectron2 necesarios
2. Importar los paquetes que se van a usar
3. Registrar en la red el *dataset* de entrenamiento y el *dataset* de validación (tanto las imágenes con las que se ha trabajado como los datos de las etiquetas)
4. Comprobar que los datos del etiquetado del set de entrenamiento son correctos representando alguno de ellos gráficamente
5. Configurar la arquitectura de red que se va a usar
6. Configurar los parámetros que se van a usar en el entrenamiento de la red (*learning rate*, número de iteraciones,...)
7. Entrenar la red con el set de entrenamiento
8. Definir el nivel de confianza de la red a partir del que la red determina qué detecciones de objetos se consideran como válidas
9. Introducirle a la red los datos del set de validación
10. Representar gráficamente las respuestas obtenidas para el set de validación
11. Obtener la matriz de confusión, que permite analizar el desempeño de la red
12. Representar gráficamente la matriz de confusión con el código de colores asociado a este tipo de gráficos

Este código de ejemplo se ha tenido que analizar en profundidad para poder entender su funcionalidad. Además, se ha ido comentando la funcionalidad de cada sección del código según se iba comprendiendo. Por último, se han realizado las modificaciones necesarias para adaptar esta funcionalidad al caso de uso concreto de este proyecto. Estas modificaciones llevadas a cabo son las siguientes:

- En el paso 3, el registro de los *datasets* se ha tenido que modificar, ya que los datos del etiquetado usados en el ejemplo no estaban en formato COCO JSON y se debían reformatear antes de introducírselos a la red.
  A partir del registro de estos *datasets*, se le ha tenido que indicar a la red que se debe trabajar con los nuevos *datasets*, en vez de con lo que se trabaja en el ejemplo.

- En el paso 4 se han representado gráficamente los datos de etiquetado de las 157 imágenes perteneciente a los sets de entrenamiento y validación. Esto ha permitido poder descartar algunas imágenes que estaban mal etiquetadas.
  Además, se han guardado las imágenes que contienen los datos de etiquetado representados gráficamente para poder comparar los resultados obtenidos de la red con el *ground truth* en algunas imágenes específicas.

- En el paso 10 se han guardado las imágenes obtenidas como salida de la red. Es decir, las imágenes del set de validación con las detecciones de objetos que ha obtenido la red representadas gráficamente.
  Además, se ha obtenido, para cada imagen del set de validación, el número de rodaballos aislados y grupos de rodaballos etiquetados y estos números se han comparado con el número de rodaballos aislados y grupos de rodaballos detectados por la red.

  Estos datos no se pueden utilizar para evaluar el desempeño de la red, ya que, que el número de rodaballos etiquetados y detectados en una imagen sea el mismo no quiere decir que se hayan detectados todos los rodaballos (si se detecta, como rodaballo, un elemento que no es un pez y, a la vez, no se detecta uno de los rodaballos etiquetados, el número de peces detectados y etiquetados sería el mismo, pero no todas las detecciones serían correctas).
  Sin embargo, estos datos sí permiten obtener otro tipo de información, por ejemplo, en qué casos el número de rodaballos detectados es superior o bastante inferior al real. Estos casos son interesantes de analizar, ya que comparando el *ground truth* y la salida obtenida de la red se puede ver qué objetos se está confundiendo con rodaballos y qué rodaballos no se están detectando.

- En el paso 11 se ha modificado la manera de analizar el desempeño de la red. El código de ejemplo utiliza el parámetro de la precisión media (AP por sus siglas en inglés - "Average Precision"), sin embargo, se ha decidido trabajar con una matriz de confusión. Las matrices de confusión permiten determinar con más precisión qué tipo de errores está cometiendo la red (por ejemplo, si se deja muchos elementos de una clase sin detectar o si suele confundir los elementos se una clase con los de otra).
  El código necesario para generar la matriz de confusión se ha obtenido de la página de foros de programación "Stack Overflow" [22]. Este código se ha analizado para poder comprender su funcionamiento y se han tenido que corregir algunos errores detectados.

  Para poder determinar cuándo un objeto ha sido detectado por la red este código trabaja con el concepto de IoU ("*Intersection over Union*). Este concepto tiene como principal función dar un criterio numérico para determinar cuándo la detección de un objeto es válida aunque la *bounding box* del objeto detectado no sea exactamente la misma a la del objeto etiquetado.

La definición matemática de este concepto se ilustra en la siguiente imagen:

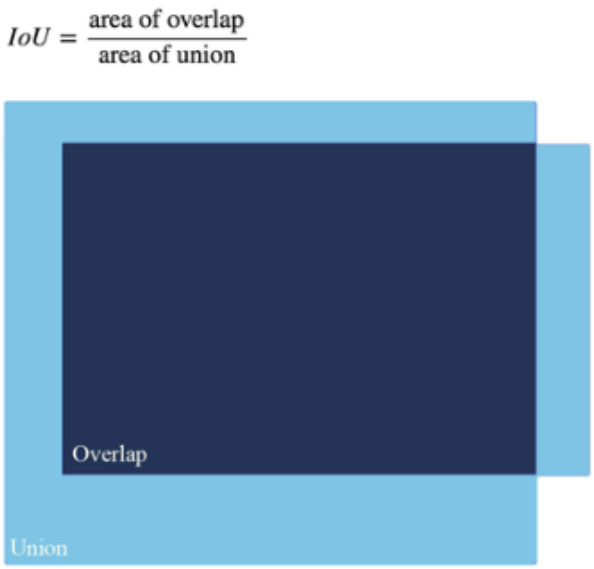


Imagen 54 - Definición del parámetro IoU ("Intersection over Union) (Fuente: [33])

Si las segmentaciones coinciden exactamente, la intersección y la unión son iguales por los que el valor de IoU será 1. Mientras que si no coincide ni un pixel, no habrá intersección y el valor de IoU será 0.
De esta forma, el pseudocódigo de esta parte del programa es el siguiente:

```
para cada imagen del set de validación
    obtener las predicciones de la red para esta imagen
    acualizar el valor de la matriz de confusión
```

Imagen 56 - Pseudocódigo para la función general del cálculo de la matriz de confusión

```
para cada etiqueta del set de validación
    para cada objeto detectado en una imagen
        calcular IoU entre las bounding boxes del objeto etiquetado y el objeto detectado
        si el IoU es mayor a 0.5
            si aún no ha habido una coincidencia de la etiqueta con otro objeto detectado
                obtener la clase del objeto etiquetado y del objeto detectado
                actualizar la matriz de confusión
                marcar el objeto detectado como emperajado con una etiqueta
                marcar la etiqueta como detectada
para cada etiqueta no detectada
    actualizar la matriz de confusión
para cada objeto detectado no emparejado con una etiqueta
    actualizar la matriz de confusión
```

Imagen 55 - Pseudocódigo de la función para actualizar la matriz de confusión

- Por último, se ha utilizado la librería de "*matplotlib*" para representar y dar formato a la matriz de confusión obtenida en el paso anterior. También se le ha aplicado un código de colores de tal forma que los valores más altos se correspondan con tonos más oscuros. Esto permite interpretar la información contenida en la matriz de forma más visual, tal como se explica más adelante.

5. Resultados

Durante el desarrollo del proceso se han ido realizando distintas pruebas. En cada una de ellas se ha entrenado a la red neuronal "Detectron2" con distintos valores para los parámetros *learning rate* y "número de iteraciones".

Como ya se ha mencionado anteriormente, el *learning rate* define lo rápido o lento que aprende una red. Es muy importante seleccionar un valor óptimo del *learning rate* ya que, tanto escoger un *learning rate* demasiado alto, como escoger un *learning rate* demasiado bajo, afecta negativamente a los resultados obtenidos de la red.

El número de iteraciones, en cambio, define el número de veces que se le introduce a la red el *dataset* de entrenamiento para ir disminuyendo el valor de la función de coste. En este caso, pasa algo similar a lo que ocurre con el *learning rate*: un número demasiado bajo del número de iteraciones puede provocar que el error cometido por la red no disminuya lo suficiente, mientras que un número de iteraciones demasiado alto puede hacer que el error alcance su valor mínimo antes de llegar al número de iteraciones indicado y después comience a aumentar.

Para ir evaluando el desempeño de la red con las distintas combinaciones de parámetros seleccionadas se han utilizado tres conceptos distintos: la matriz de confusión, la precisión y el *recall*.

Matriz de confusión

La matriz de confusión es una herramienta que muestra de forma visual el desempeño de un algoritmo de clasificación. Para entender la matriz de clasificación, primero se deben entender varios conceptos adicionales:

- Falso positivo (FP – "*False positive*"): este caso se da cuando un algoritmo de clasificación detecta que un elemento es de una clase cuando este no lo es. En el caso de este proyecto equivaldría a decir que la red neuronal ha detectado como un pez algo que realmente no es un pez.

- Falso negativo (FN – "*False negative*"): este caso se da cuando no se detecta un elemento que pertenece una clase. En el caso de este proyecto sería el caso en el que algún rodaballo presente en la imagen no se detectara como tal.

- Verdadero positivo (TP – "*True positive*"): este caso se da cuando un elemento de una clase se detecta como tal, es decir, cuando hay un rodaballo en la imagen y la red lo detecta correctamente.

- Verdadero negativo(TN – "*True negative*"): este caso se da cuando un elemento que no pertenece a una clase no se detecta como perteneciente a esa clase. En este proyecto sería el caso de que un trozo de comida no se detectara como rodaballo.
  Cuando hay varias clases, un verdadero negativo también se da en el caso en el que un elemento de una clase se detecte como de esa clase y no como de otra clase distinta. Por ejemplo, si tenemos un grupo de rodaballos en una imagen y la red detecta este grupo correctamente, desde el punto de la clase "grupo de rodaballos" este resultado es un verdadero positivo, mientras que desde el punto de vista de la clase "rodaballo aislado" es

un verdadero negativo, porque se ha detectado correctamente que ese elemento no pertenece a la clase “rodaballo aislado”.

La matriz de confusión para un algoritmo de clasificación binaria tiene la siguiente estructura:

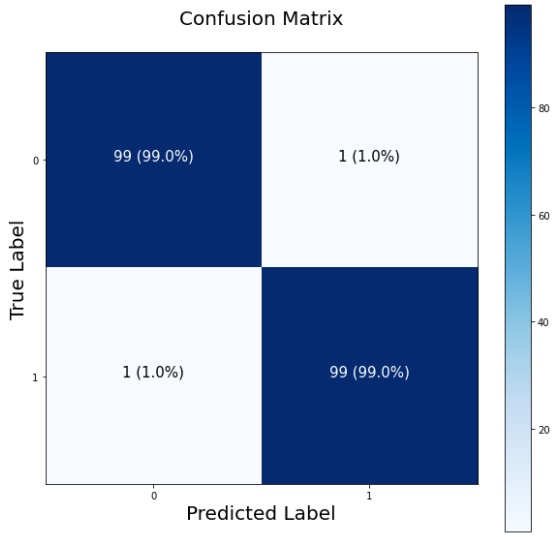


Imagen 57 - Ejemplo de matriz de confusión binaria (Fuente: [32])

Como se puede observar en la *Imagen 54*, en el eje horizontal se indican las etiquetas que el algoritmo de clasificación ha predicho, mientras que en el eje vertical se indica el *ground truth*.

Por tanto, es fácil apreciar que el cuadrado superior izquierdo de la matriz indica el porcentaje de “True Negative” que ha habido, ya que los elementos representados en ese cuadrado de la matriz son aquellos que estaban etiquetados con la etiqueta 0, y el algoritmo les ha asignado la etiqueta correctamente.

De forma similar, el cuadrado inferior derecho indica el porcentaje de “True Positive” que ha habido.

Por otro lado, el cuadrado superior derecho indica los “False positive” (la etiqueta predicha es 1, pero la real es 0), mientras que el cuadrado inferior izquierdo representa el porcentaje de “False negative” (la etiqueta predicha es 0, pero la real es 1).

Esta estructura de TN, FP en la fila superior y FN, TP en la inferior se sigue manteniendo en las matrices de confusión multi-clase.

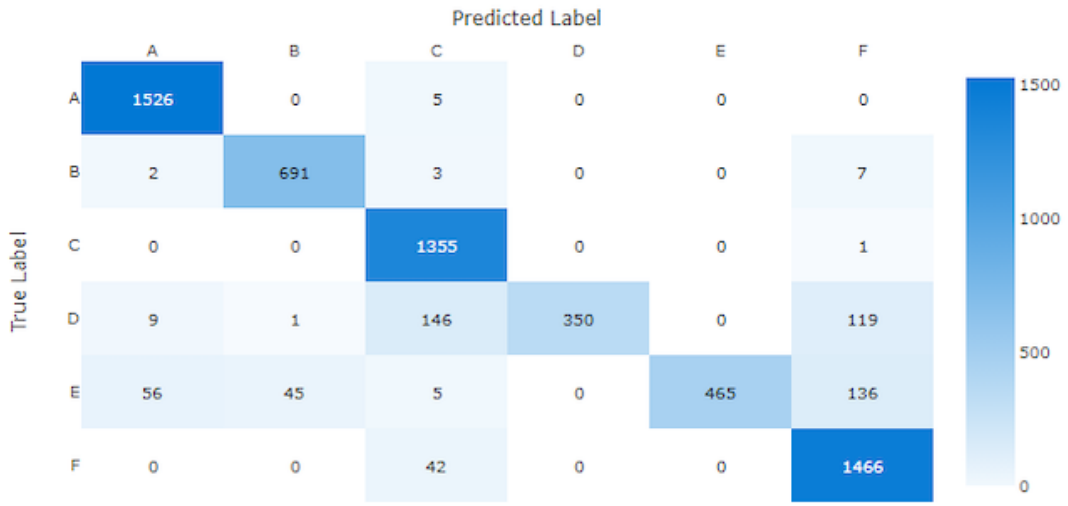


Imagen 58 - Ejemplo de matriz de confusión (Fuente: [30])

Como se puede ver en la *Imagen* 55, la estructura de las matrices de confusión multi-clase difieren de la estructura de las matrices binarias en que los FP y los FN pueden pertenecer a distintas clases. Por ejemplo, si se toma como referencia la clase "B" se puede ver que hay 1 FP con la clase "D" y 45 con la clase "E" (son elementos que se han detectado como pertenecientes a la case "B" cuando realmente pertenecen a otras clases). Además , hay 2 FN con la clase "A", 3 con la "C" y 7 con la "F" (son elementos que pertenecen a la clase "B", pero que se han detectado como pertenecientes a otras clases).

Gracias a estas herramientas se podrá realizar el estudio de los TP, TN, FP y FN que se han obtenido de la red en cada una de las pruebas. Esta herramienta, además, es especialmente útil para análisis de datos de este tipo, ya que se suele usar una escala de colores para identificar rápidamente donde se concentran la mayor parte de los resultados. **Cuanto más oscuros sean los recuadros de la diagonal de la matriz (que son los TP), mejor será el desempeño de la red**.

Precisión

La precisión es otra forma de medir el desempeño de la red. En este caso, **este parámetro indica, de todos los elementos asignados a una clase, cuántos de ellos son verdaderamente de esa clase**. Es decir, en este proyecto, la precisión mide cuántos de los rodaballos asilados detectados son realmente rodaballos aislados y cuántos de los grupos de rodaballos detectados son realmente grupos de rodaballos.

Matemáticamente la precisión se calcula de la siguiente manera:

$$Precision = \frac{TP}{TP + FP}$$

Imagen 59 - Fórmula para el cálculo de la precisión (Fuente: [33])

Recall

El *recall* es la tercera forma de evaluar el desempeño de la red que se va a usar en este proyecto. Este parámetro **evalúa, de todos los elementos pertenecientes a una clase, cuántos han sido detectados por el algoritmo**. En este proyecto, evaluaría cuántos rodaballos aislados y grupos de rodaballos no han sido detectados por la red. El *recall* se define de la siguiente manera:

$$Recall = \frac{TP}{TP + FN}$$

Imagen 60 - Fórmula para el cálculo del recall (Fuente: [33])

Ya se ha comentado que un buen desempeño de la red viene dado por una matriz cuya diagonal contiene los valores más altos. Del mismo modo, si se quiere evaluar la red mediante el valor del *recall* y la precisión, se necesita que ambos parámetros tengan valores altos para poder determinar que el desempeño de un algoritmo es bueno. Esto se debe a que la precisión y el *recall* miden aspectos distintos dentro del desempaño de la red. Si alguno de los dos parámetros tiene un valor bajo ocurre lo siguiente:

- *Recall* alto con una precisión baja: esto indica que el algoritmo está detectando a casi todos los elementos que pertenecen a una clase. Sin embargo, también indica que, muchos elementos que no pertenecen a una clase se están clasificando mal.

- Precisión alta con un *recall* bajo: esto quiere decir que casi todos los elementos detectados por el algoritmo están bien clasificados, pero que, al mismo tiempo, el algoritmo se deja muchos elementos sin detectar.

5.1. Análisis de resultados

En el desarrollo de este proyecto se han realizado varias pruebas. En estas pruebas se ha ido modificando el valor del LR (*learning rate*), el número de las iteraciones del entrenamiento y el umbral de confianza para, de esta manera, poder determinar con qué parámetros se conseguía un mejor desempeño de la red. Cada entrenamiento ha tenido una duración de entre una hora y media y dos horas y media, dependiendo del número de iteraciones seleccionado.

Las pruebas realizadas han sido las siguientes:

1. LR = 0.0025 y num_iteraciones = 3000

   El programa con el que se ha estado trabajando proporciona varias gráficas que permiten ver distintos datos de interés referentes a la fase de entrenamiento de la red. En este caso, solo tres de las gráficas son de interés.

La primera gráfica de interés representa la disminución del error de la red según se van incrementando el número de iteraciones realizadas. Esta disminución del error es el resultado del proceso de optimización del valor de la función de coste explicado en el capítulo 2.

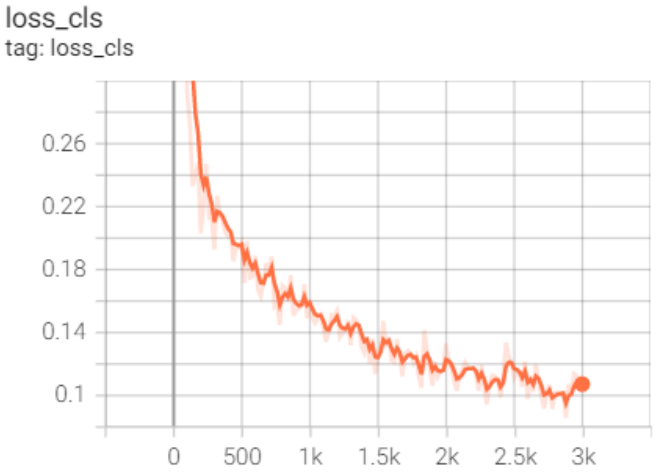


Imagen 61 - Representación de la disminución del error en función de las iteraciones de la fase de entrenamiento (Prueba 1)

La segunda gráfica, en cambio, representa el incremento de la precisión de la red como resultado de la disminución del error mostrada en la gráfica anterior. Como se puede ver, la precisión final es muy alta (≈ 96%), sin embargo, esta precisión no se debe confundir con la precisión utilizada para evaluar los resultados obtenidos por la red, ya que esa precisión se debe calcular a partir de los datos del set de validación y distingue los aciertos según la clase que se esté evaluando mientras que esta gráfica no.

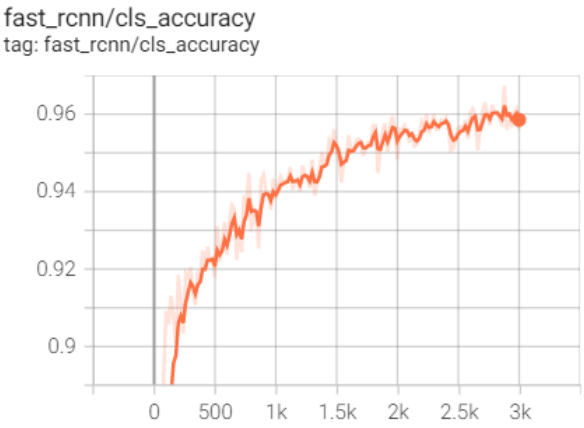


Imagen 62 - Representación de la variación en la precisión en función de las iteraciones de la fase de entrenamiento (Prueba 1)

La última gráfica de interés representa la disminución de la tasa de falsos negativos como resultado de la disminución del error cometido por la red. Como se puede ver en la fórmula del *recall* (*Imagen 57)* cuántos menos falsos negativos haya, mayor será el valor del *recall*. Por ello, esta gráfica representa, de forma indirecta, la disminución del *recall* a lo largo de las iteraciones realizadas. Igual que ocurría con la gráfica anterior, este valor de *recall* no se debe confundir con el valor de *recall* que sirve para analizar el desempeño de la red.

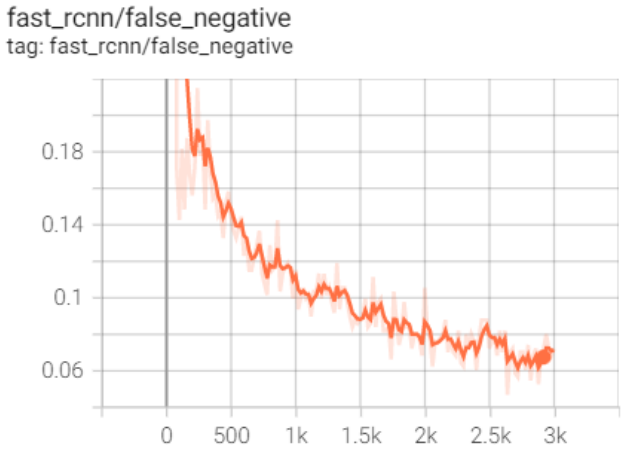


Imagen 63 - Representación de la variación de la tasa de falsos negativos (FN) en función de las iteraciones de la fase de entrenamiento (Prueba 1)

Estas tres gráficas muestran que el comportamiento de la red es el esperado, puesto que el valor del error de la red y la tasa de los falsos negativos van disminuyendo y el valor de la precisión va aumentando según avanzan las iteraciones. Esto en principio indica que la red ha ido aprendiendo, sin embargo, no se debe olvidar que la red podría estar memorizando los datos y puede no haber sido capaz de generalizarlos. Además, también hay que determinar si la red ha aprendido lo suficiente, o si necesita de un mayor número de iteraciones para disminuir aún más su error.

Para ello se va a analizar la matriz de confusión generada a partir de los datos obtenidos con el set de validación. Como ya se ha mencionado antes, también es interesante ver cómo afecta el parámetro de umbral de confianza al desempeño de la red. El umbral de confianza define a partir de qué nivel de confianza se da una respuesta por válida. Un umbral de confianza bajo permitirá que se detecten más rodaballos (y, por tanto, que el *recall* aumente), sin embargo, también es más probable que se clasifiquen como rodaballos o grupos de rodaballos objetos que no son de ninguna de las dos clases (y, por tanto, que la precisión disminuya).

Por este motivo, se ha decidido calcular dos matrices de confusión, una para un umbral de confianza del 0.5 y otra para un umbral de confianza del 0.7.

Umbral de confianza = 0.5

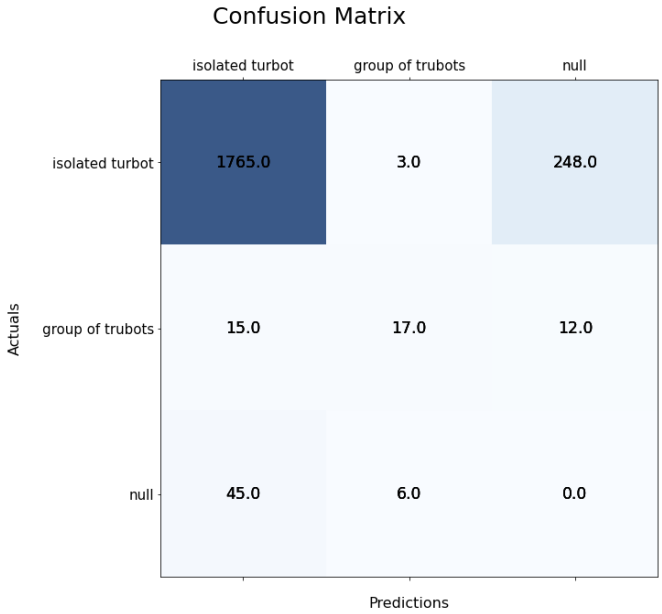


Imagen 64 - Matriz de confusión de la prueba 1 (umbral de confianza = 0.5)

Como se puede ver en la matriz de confusión de la *Imagen 64* hay muchos más rodaballos asilados que grupos de rodaballos en las imágenes del set de validación. Por ello, se ha decidido normalizar la matriz de confusión para que el código de colores sea más visual. La normalización se ha llevado a cabo sumando todos los datos de una misma fila y dividiendo todos sus términos por el valor de la suma obtenido.

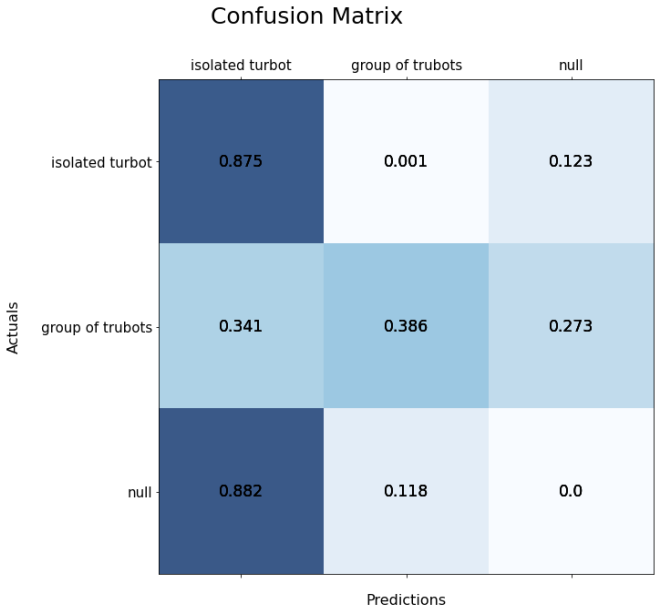


Imagen 65- Matriz de confusión de la prueba 1 normalizada (umbral de confianza = 0.5)

El problema que se quiere abordar en este proyecto no es de clasificación, sino que es de detección de objetos. Por ello, se han añadido unas etiquetas llamadas “null”. Estas etiquetas permiten obtener información muy útil. La etiqueta “null” del eje horizontal indica cuántos rodaballos y grupos de rodaballos no se han detectado (no se les ha asignado ninguna etiqueta en la salida de la red). En cambio, la etiqueta “null” del eje vertical indica cuántos objetos que no son peces se han detectado como rodaballos o grupos de rodaballos.

Con esta información contenida en la matriz de confusión podemos sacar el valor de la precisión y el *recall* obtenidos a la salida de la red:

$$\boldsymbol{Precisión_{rodaballo\ aislado} = \frac{TP}{TP + FP} = \frac{1765}{1765 + 15 + 45} = 0.967} \quad (11)$$

$$\boldsymbol{Precisión_{grupo\ de\ rodaballos} = \frac{TP}{TP + FP} = \frac{17}{3 + 17 + 6} = 0.654} \quad (12)$$

$$\boldsymbol{Recall_{rodaballo\ aislado} = \frac{TP}{TP + FN} = \frac{1765}{1765 + 3 + 248} = 0.875} \quad (13)$$

$$\boldsymbol{Recall_{grupos\ de\ rodaballos} = \frac{TP}{TP + FN} = \frac{17}{15 + 17 + 12} = 0.386} \quad (14)$$

Umbral de confianza = 0.7

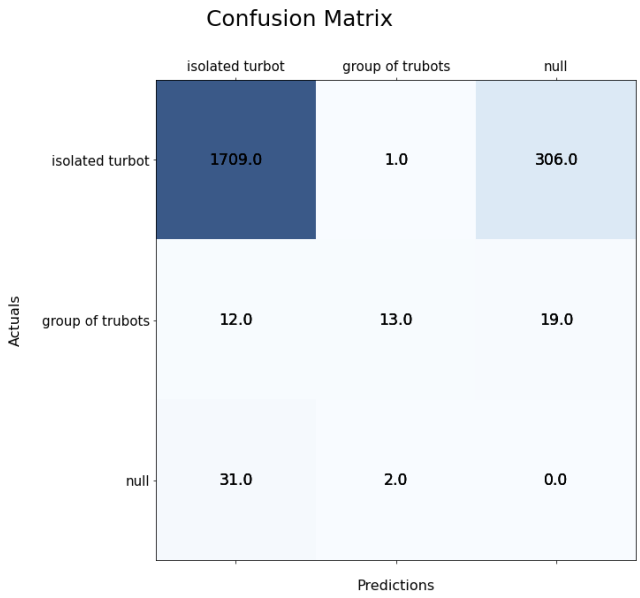


Imagen 66 - Matriz de confusión de la prueba 1 (umbral de confianza = 0.7)

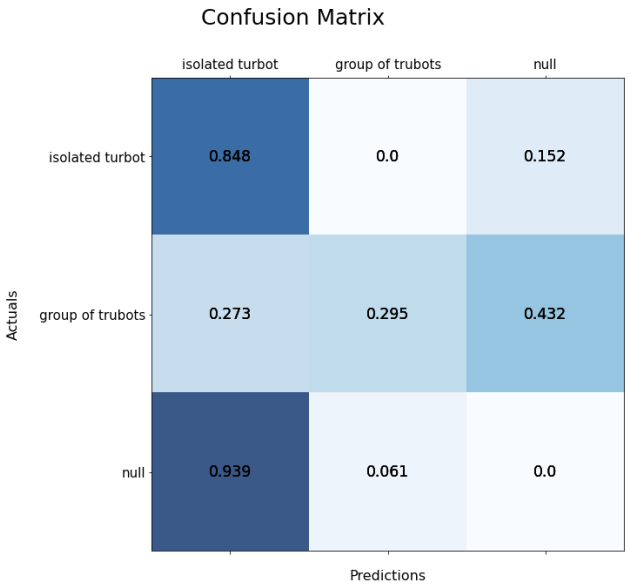


Imagen 67 - Matriz de confusión de la prueba 1 normalizada (umbral de confianza = 0.7)

Los valores de la precisión y el *recall* para esta segunda matriz de confusión son los siguientes:

$$\boldsymbol{Precisión_{rodaballo\ aislado} = \frac{TP}{TP + FP} = \frac{1709}{1709 + 12 + 31} = 0.975} \quad (15)$$

$$\boldsymbol{Precisión_{grupo\ de\ rodaballos} = \frac{TP}{TP + FP} = \frac{13}{1 + 13 + 2} = 0.812} \quad (16)$$

$$\boldsymbol{Recall_{rodaballo\ aislado} = \frac{TP}{TP + FN} = \frac{1709}{1709 + 1 + 306} = 0.848} \quad (17)$$

$$\boldsymbol{Recall_{grupos\ de\ rodaballos} = \frac{TP}{TP + FN} = \frac{13}{12 + 13 + 19} = 0.295} \quad (18)$$

Se puede ver que la red obtiene resultados muy buenos en cuanto a la detección de rodaballos aislados, es decir, que se deja un porcentaje muy bajo de ellos sin detectar y que de los que detecta la gran mayoría están clasificados correctamente.

Sin embargo, con la clase de grupo de rodaballos no ocurre lo mismo. La precisión, sí tiene un valor muy bueno, pero el *recall* tiene un valor considerablemente bajo.

Por tanto, esto indica que el principal problema que presenta este modelo de la red es que no detecta muchos de los grupos rodaballos presentes en las imágenes.

Por otro lado, se puede ver que al modificar el valor del umbral de confianza la precisión mejora notablemente en ambas clases, mientras que el *recall* empeora considerablemente. Este resultado es lógico ya que al aumentar el umbral de confianza lo más probable es que los

objetos detectados estén clasificados correctamente, pero que, a la vez, el sistema se deje muchos peces sin detectar.

2. LR = 0.0025 y num_iteraciones = 5000

Para esta configuración de la red neuronal se han obtenido las siguientes gráficas:

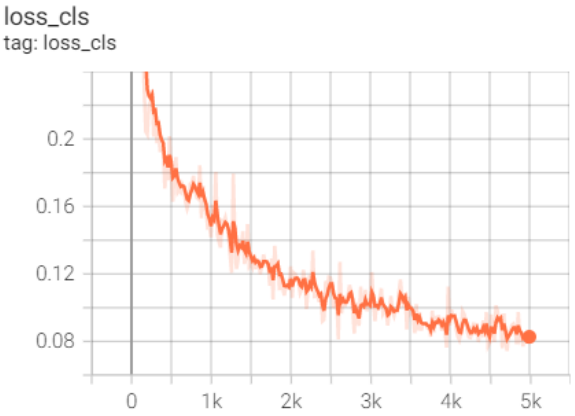


Imagen 69 - Representación de la disminución del error en función de las iteraciones de la fase de entrenamiento (Prueba 2)

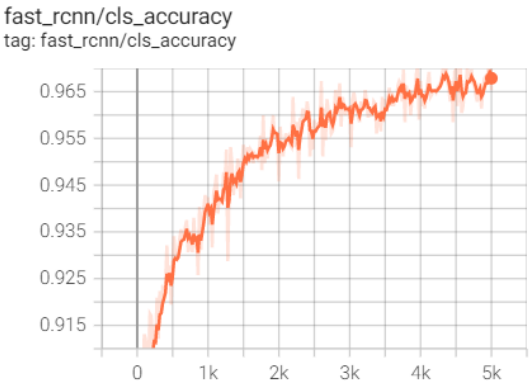


Imagen 68 - Representación de la variación en la precisión en función de las iteraciones de la fase de entrenamiento (Prueba 2)

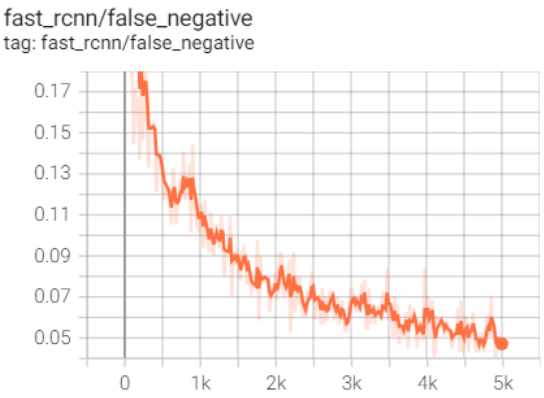


Imagen 70 - Representación de la variación de la tasa de falsos negativos (FN) en función de las iteraciones de la fase de entrenamiento (Prueba 2)

Se puede observar que, al haber incrementado el número de iteraciones, la red ha conseguido mejorar las tres métricas mostradas en las gráficas, ya que el error y la tasa de FN han disminuido mientras que la precisión ha aumentado su valor.

A continuación, se muestran las matrices de confusión correspondientes a los umbrales de confianza 0.5 y 0.7.

Umbral de confianza = 0.5

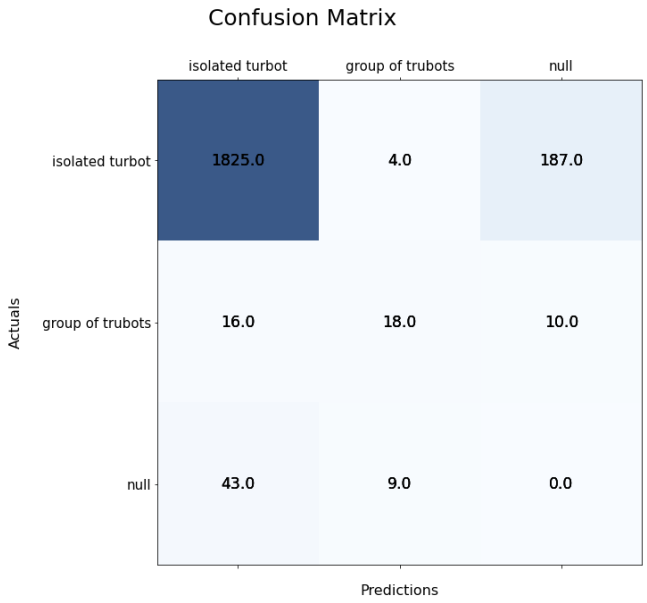


Imagen 71 - Matriz de confusión de la matriz prueba 2 (umbral de confianza = 0.5)

La matriz de confusión normalizada queda de la siguiente manera:

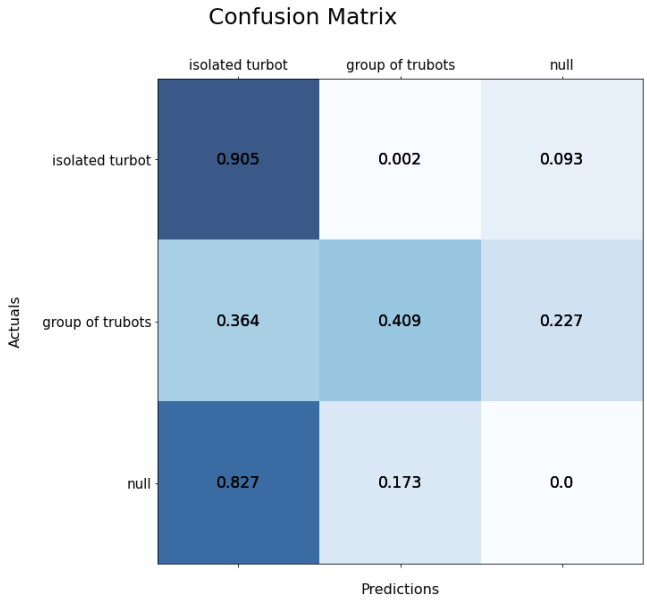


Imagen 72 - Matriz de confusión de la matriz prueba 2 normalizada (umbral de confianza = 0.5)

Para este segundo modelo de la red, los valores de precisión y *recall* obtenidos son los siguientes:

$$\boldsymbol{Precisión_{rodaballo\ aislado} = \frac{TP}{TP + FP} = \frac{1825}{1825 + 16 + 43} = 0.969} \quad (19)$$

$$\boldsymbol{Precisión_{grupo\ de\ rodaballos} = \frac{TP}{TP + FP} = \frac{18}{4 + 18 + 9} = 0.58} \quad (20)$$

$$\boldsymbol{Recall_{rodaballo\ aislado} = \frac{TP}{TP + FN} = \frac{1825}{1825 + 4 + 187} = 0.905} \quad (21)$$

$$\boldsymbol{Recall_{grupos\ de\ rodaballos} = \frac{TP}{TP + FN} = \frac{18}{16 + 18 + 10} = 0.409} \quad (22)$$

Umbral de confianza = 0.7

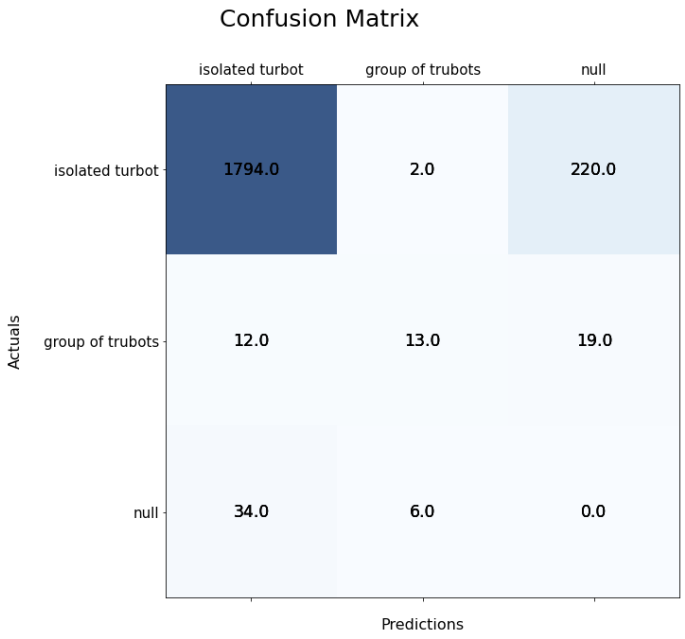


Imagen 73 - Matriz de confusión de la matriz prueba 2 (umbral de confianza = 0.7)

En este segundo caso, la matriz normalizada es la siguiente:

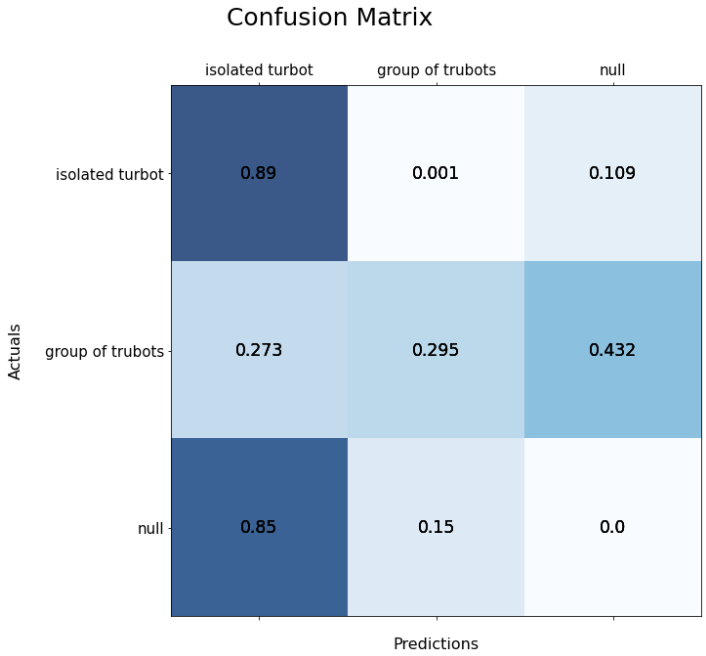


Imagen 74 - Matriz de confusión de la prueba 2 normalizada (umbral de confianza = 0.7)

Al incrementar el umbral de confianza, los valores de precisión y *recall* quedan de la siguiente manera:

$$\boldsymbol{Precisión_{rodaballo\ aislado} = \frac{TP}{TP + FP} = \frac{1794}{1794 + 12 + 34} = 0.975} \quad (23)$$

$$\boldsymbol{Precisión_{grupo\ de\ rodaballos} = \frac{TP}{TP + FP} = \frac{13}{2 + 13 + 6} = 0.619} \quad (24)$$

$$\boldsymbol{Recall_{rodaballo\ aislado} = \frac{TP}{TP + FN} = \frac{1794}{1794 + 2 + 220} = 0.89} \quad (25)$$

$$\boldsymbol{Recall_{grupos\ de\ rodaballos} = \frac{TP}{TP + FN} = \frac{13}{12 + 13 + 19} = 0.29} \quad (26)$$

Al igual que en la prueba anterior, en la clase "rodaballo aislado" se obtienen muy buenos resultados, mientras que en la clase "grupo de rodaballo" los resultados obtenidos son peores, sobre todo si se toma el valor del *recall*.

Sin embargo, sí se puede observar una mejora en los valores del *recall* de ambas clases y en el valor de precisión en la clase "rodaballo aislado". Por otro lado, también se puede observar que el valor de precisión en los grupos de rodaballos ha empeorado considerablemente. Esto se debe a que, con este modelo, la red detecta un mayor número de rodaballos aislados y de grupos de rodaballos, pero tiende a confundir con más frecuencia objetos que no son peces con grupos de rodaballos.

Además, se puede ver que, según lo previsto, en esta segunda prueba también aumenta la precisión y disminuye el *recall* al aumentar el umbral de confianza.

3. LR = 0.0025 y num_iteraciones = 5500

En esta tercera prueba se han obtenido las siguientes gráficas:

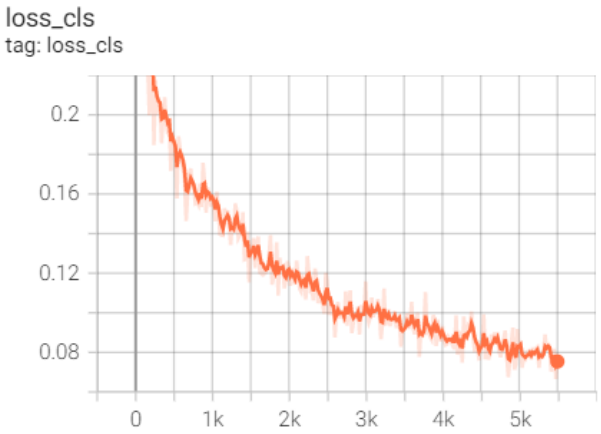


Imagen 76 - Representación de la disminución del error en función de las iteraciones de la fase de entrenamiento (Prueba 3)

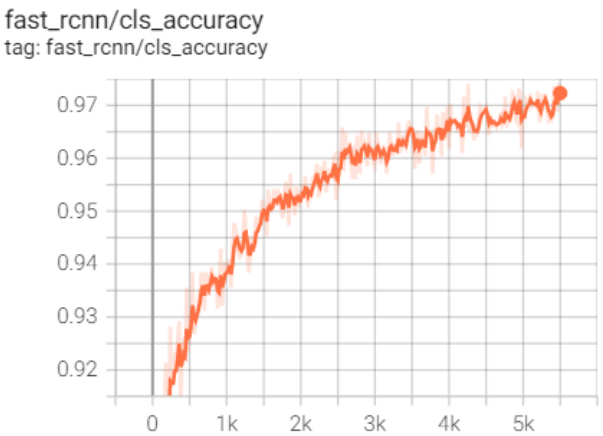


Imagen 75 - Representación de la variación de la precisión en función de las iteraciones de la fase de entrenamiento (Prueba 3)

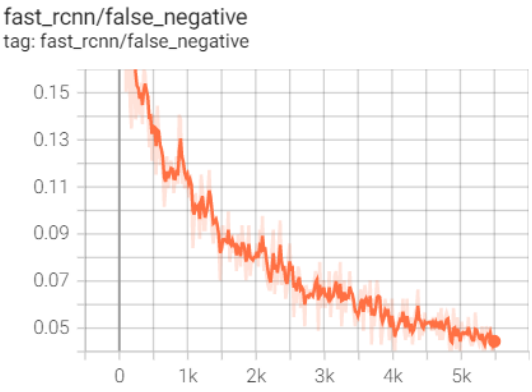


Imagen 77 - Representación de la variación de la tasa de falsos negativos (FN) en función de las iteraciones de la fase de entrenamiento (Prueba 3)

En esta ocasión se puede ver una ligera mejora de las tres métricas mostradas en las gráficas. Sin embargo, al haber dejado el mismo *learning rate* de la prueba anterior y haber aumentado muy poco el número iteraciones (de 5000 a 5500), los valores se han mantenido muy similares.

A continuación se muestran las matrices de confusión correspondientes a los umbrales de confianza 0.5 y 0.7.

Umbral de confianza = 0.5

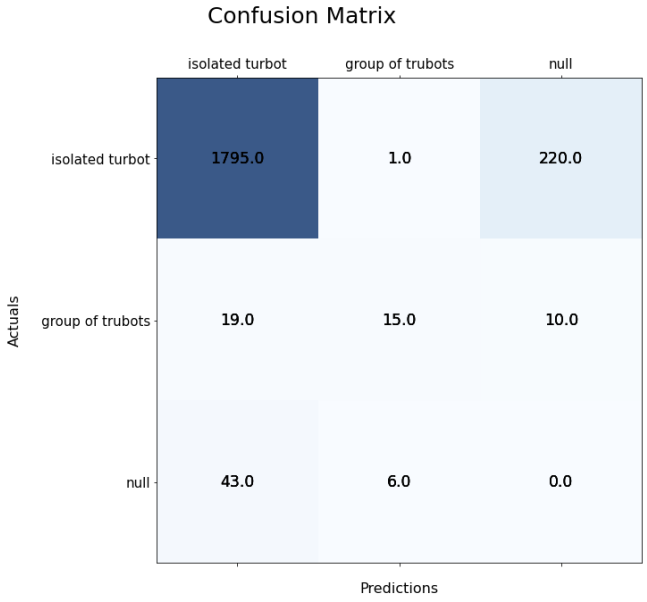


Imagen 78 - Matriz de confusión de la prueba 3 (umbral de confianza = 0.5)

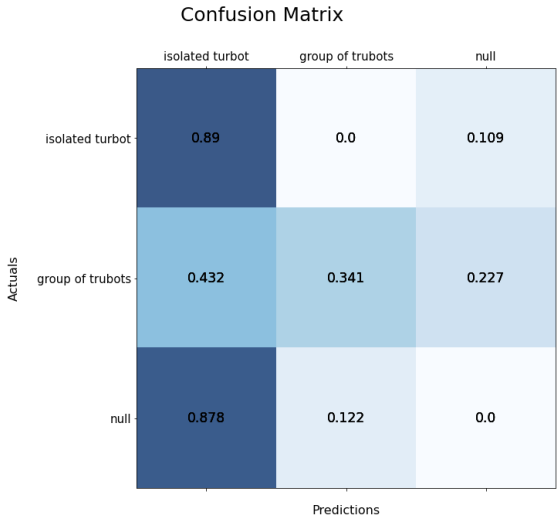


Imagen 79 - Matriz de confusión de la prueba 3 normalizada (umbral de confianza = 0.5))

El modelo de esta segunda prueba tiene los siguientes valores de precisión y *recall*:

$$\boldsymbol{Precisión_{rodaballo\ aislado} = \frac{TP}{TP + FP} = \frac{1795}{1795 + 19 + 43} = 0.967} \quad (27)$$

$$\boldsymbol{Precisión_{grupo\ de\ rodaballos} = \frac{TP}{TP + FP} = \frac{15}{1 + 15 + 6} = 0.682} \quad (28)$$

$$\boldsymbol{Recall_{rodaballo\ aislado} = \frac{TP}{TP + FN} = \frac{1795}{1795 + 1 + 220} = 0.89} \quad (29)$$

$$\boldsymbol{Recall_{grupos\ de\ rodaballos} = \frac{TP}{TP + FN} = \frac{15}{19 + 15 + 10} = 0.341} \quad (30)$$

Umbral de confianza = 0.7

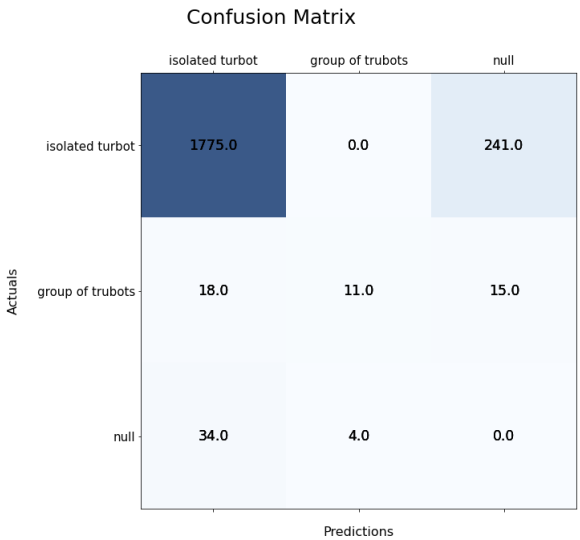


Imagen 80 - Matriz de confusión de la prueba 3 (umbral de confianza = 0.7)

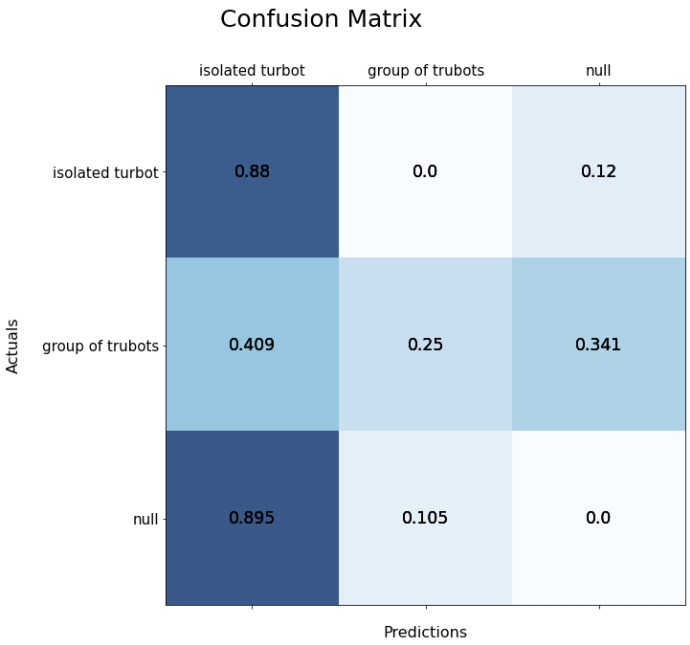


Imagen 81 - Matriz de confusión de la prueba 3 normalizada (umbral de confianza = 0.7)

Al aumentar el umbral de confianza de 0.5 a 0.7, los cálculos de la precisión y el *recall* que dan de la siguiente manera:

$$\boldsymbol{Precisión_{rodaballo\ aislado} = \frac{TP}{TP + FP} = \frac{1775}{1775 + 18 + 34} = 0.972} \quad (31)$$

$$\boldsymbol{Precisión_{grupo\ de\ rodaballos} = \frac{TP}{TP + FP} = \frac{11}{11 + 4} = 0.733} \quad (32)$$

$$\boldsymbol{Recall_{rodaballo\ aislado} = \frac{TP}{TP + FN} = \frac{1775}{1775 + 241} = 0.88} \quad (33)$$

$$\boldsymbol{Recall_{grupos\ de\ rodaballos} = \frac{TP}{TP + FN} = \frac{11}{18 + 11 + 15} = 0.25} \quad (34)$$

Al igual que en las pruebas anteriores, la red es capaz de clasificar los rodaballos aislados con buenos resultados. Sin embargo, se deja muchos grupos de rodaballos sin detectar.

Comparado con la prueba anterior, se puede observar muy poca variación en los resultados. Sin embargo, cabe destacar que en la clase "grupo de rodaballos" el valor del recall ha empeorado, mientras que el de la precisión ha disminuido. Esto quiere decir que este modelo tiene una tasa de acierto mayor en los objetos que etiqueta como "grupo de rodaballo", pero que, a la misma vez, se deja más grupos de rodaballos sin detectar.

Además, al igual que en pruebas anteriores, al aumentar el umbral de confianza los valores de precisión mejoran y los del *recall* disminuye, lo cual es coherente con la definición de "umbral de confianza".

4. LR = 0.01 y num_iteraciones = 5500

Para esta cuarta configuración de la red se han obtenido las siguientes gráficas:

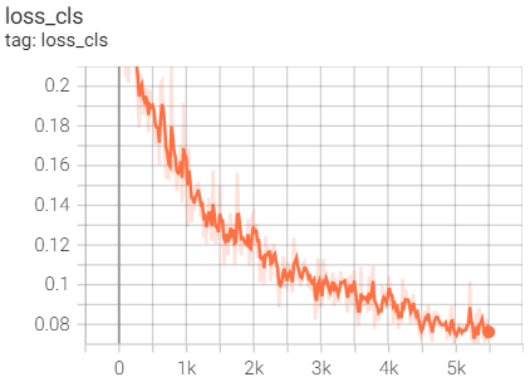


Imagen 83 - Representación de la disminución del error en función de las iteraciones de la fase de entrenamiento (Prueba 4)

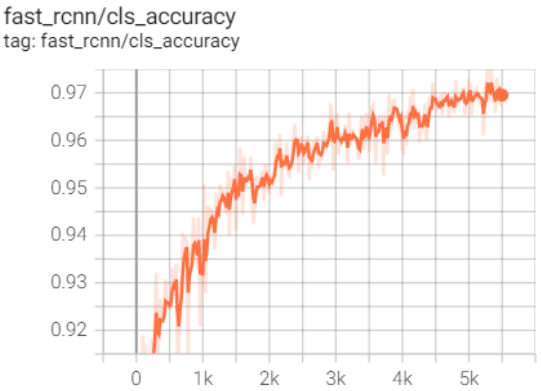


Imagen 84 - Representación de la variación de la precisión en función de las iteraciones de la fase de entrenamiento (Prueba 4)

Imagen 82 - Representación de la variación de la tasa de falsos negativos (FN) en función de las iteraciones de la fase de entrenamiento (Prueba 4)

Las tres métricas mostradas en las tres gráficas siguen presentando valores muy buenos. Sin embargo, a pesar de haber aumentado considerablemente el valor del *learning rate* (de 0.0025 a 0.01) los resultados han variado muy poco.

Las matrices de confusión obtenidas con los datos del set de validación son las siguientes:

Umbral de confianza = 0.5

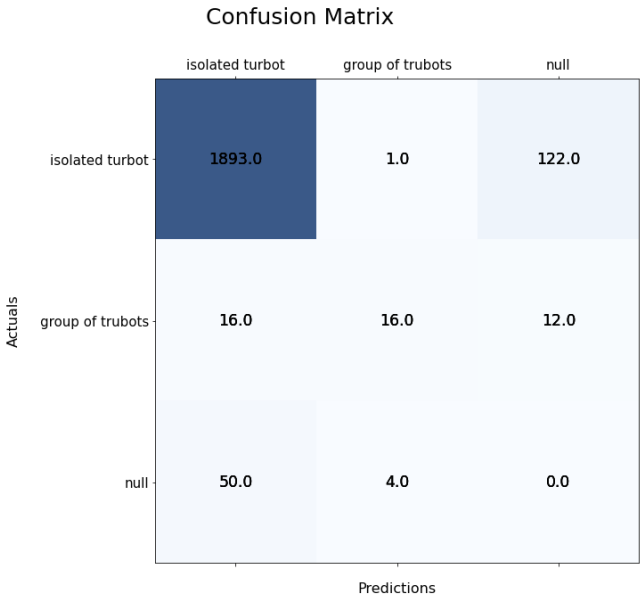


Imagen 85 - Matriz de confusión de la prueba 4 (umbral de confianza = 0.5)

La matriz normalizada queda de la siguiente manera:

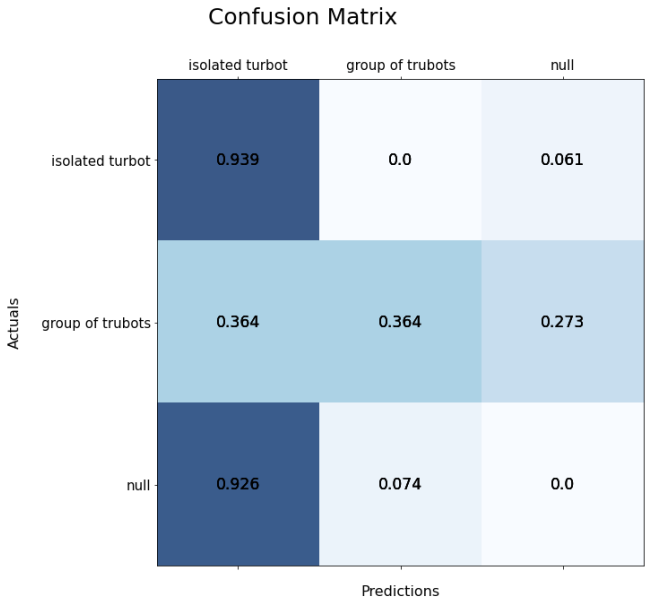


Imagen 86 - Matriz de confusión de la prueba 4 normalizada (umbral de confianza = 0.5)

A partir de esta matriz de confusión se obtienen los siguientes valores de precisión y *recall*:

$$\boldsymbol{Precisión_{rodaballo\ aislado} = \frac{TP}{TP + FP} = \frac{1893}{1893 + 16 + 50} = 0.966} \quad (35)$$

$$\boldsymbol{Precisión_{grupo\ de\ rodaballos} = \frac{TP}{TP + FP} = \frac{16}{1 + 16 + 4} = 0.762} \quad (36)$$

$$\boldsymbol{Recall_{rodaballo\ aislado} = \frac{TP}{TP + FN} = \frac{1893}{1893 + 1 + 122} = 0.939} \quad (37)$$

$$\boldsymbol{Recall_{grupos\ de\ rodaballos} = \frac{TP}{TP + FN} = \frac{16}{16 + 16 + 12} = 0.364} \quad (38)$$

Umbral de confianza = 0.7

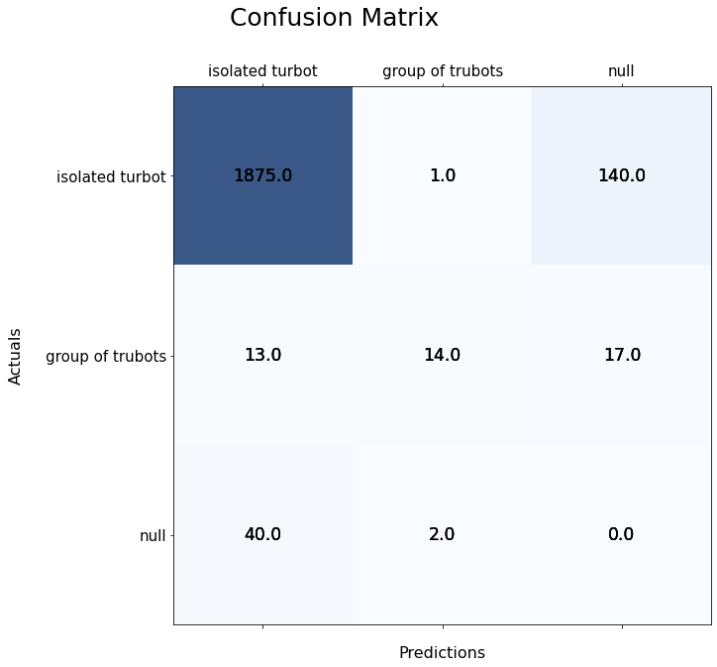


Imagen 87 - Matriz de confusión de la prueba 4 (umbral de confianza = 0.7)

Con el cambio de umbral de confianza la matriz normalizada presenta los siguientes resultados:

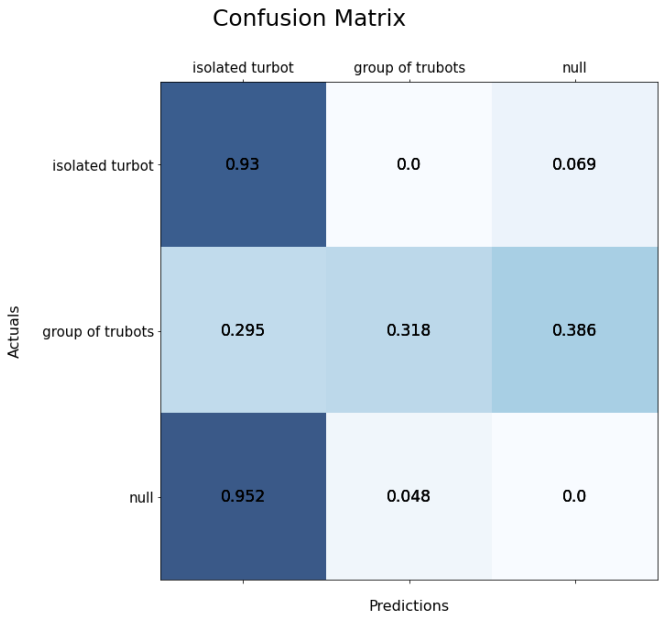


Imagen 88 - Matriz de confusión de la prueba 4 normalizada (umbral de confianza = 0.7)

Con los datos obtenidos de esta segunda matriz de confianza, los valores de precisión y *recall* obtenidos son los siguientes:

$$\boldsymbol{Precisión_{rodaballo\ aislado} = \frac{TP}{TP + FP} = \frac{1875}{1875 + 13 + 40} = 0.973} \quad (39)$$

$$\boldsymbol{Precisión_{grupo\ de\ rodaballos} = \frac{TP}{TP + FP} = \frac{14}{1 + 14 + 2} = 0.824} \quad (40)$$

$$\boldsymbol{Recall_{rodaballo\ aislado} = \frac{TP}{TP + FN} = \frac{1875}{1875 + 1 + 140} = 0.93} \quad (41)$$

$$\boldsymbol{Recall_{grupos\ de\ rodaballos} = \frac{TP}{TP + FN} = \frac{14}{13 + 14 + 17} = 0.318} \quad (42)$$

En este modelo, también se puede ver que, se obtienen mejores resultados en la precisión y peores en el *recall* al seleccionar el nivel de confianza de 0.7.

Además, se puede ver cómo este modelo, con un umbral de confianza de 0.5, en general, obtiene resultados mejores que los de los modelos de las pruebas 2 y 3. No obstante, cabe destacar que en el *recall* de la clase “grupo de rodaballos”, este modelo empeora su valor con

respecto a las pruebas 1 y 2. Esto se debe principalmente a que la red detecta los grupos de rodaballos como rodaballos aislados con más frecuencia.

5. LR = 0.02 y num_iteraciones = 5500

La gráficas obtenidas al entrenar la red con estos parámetros son las siguientes:

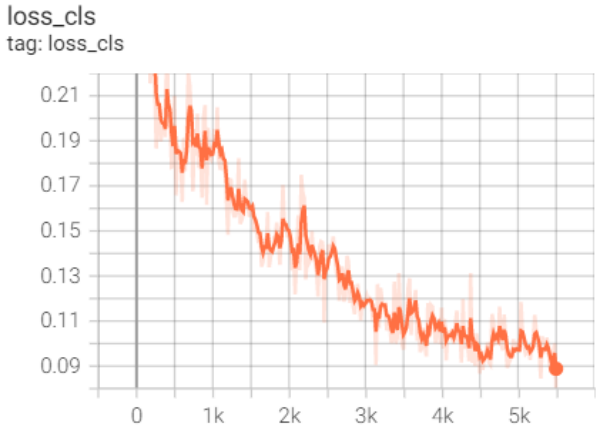


Imagen 90 - Representación de la disminución del error en función de las iteraciones de la fase de entrenamiento (Prueba 5)

Imagen 91 - Representación de la variación de la precisión en función de las iteraciones de la fase de entrenamiento (Prueba 5)

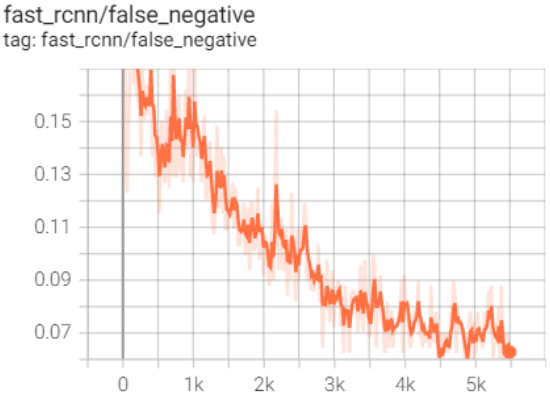


Imagen 89 - Representación de la variación de la tasa de falsos negativos (FN) en función de las iteraciones de la fase de entrenamiento (Prueba 5)

En las gráficas obtenidas a partir de este modelo se puede ver el efecto que produce tener un *learning rate* demasiado alto. Como se puede observar, los tres parámetros mostrados en las gráficas varían su valor de forma mucho más brusca según va aumentando el número de iteraciones realizadas. Esto provoca que haya un empeoramiento del valor final de estos tres parámetros en comparación con los valores obtenidos en la prueba anterior (entre este modelo y el anterior, únicamente se ha aumentado el valor del *learning rate* de 0.01 a 0.02).

Por otro lado, las matrices de confusión obtenidas mediante el estudio de los datos del set de validación son las siguientes:

Umbral de confianza = 0.5

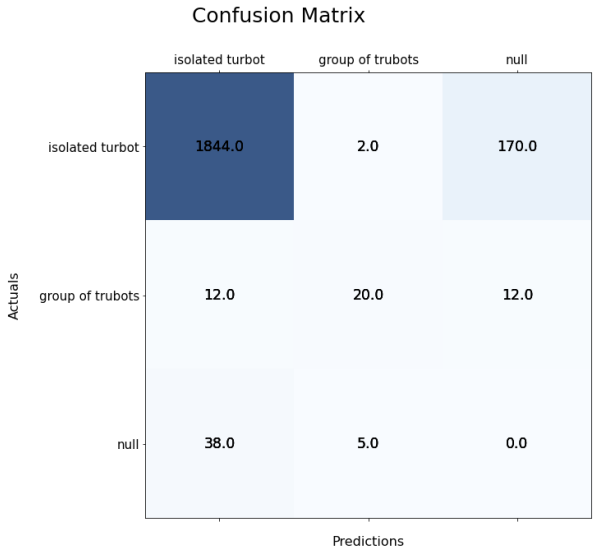


Imagen 92 - Matriz de confusión de la prueba 5 (umbral de confianza = 0.5)

Si se normaliza la matriz de confusión, se obtienen los siguientes datos:

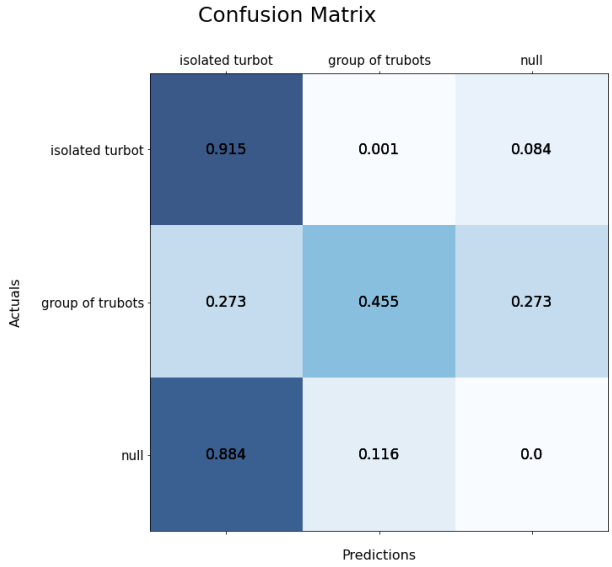


Imagen 93 - Matriz de confusión de la prueba 5 normalizada (umbral de confianza = 0.5)

Con estos datos, la precisión y el *recall* quedan de la siguiente manera:

$$\boldsymbol{Precisión_{rodaballo\ aislado} = \frac{TP}{TP + FP} = \frac{1844}{1844 + 12 + 38} = 0.974} \quad (43)$$

$$\boldsymbol{Precisión_{grupo\ de\ rodaballos} = \frac{TP}{TP + FP} = \frac{20}{2 + 20 + 5} = 0.74} \quad (44)$$

$$\boldsymbol{Recall_{rodaballo\ aislado} = \frac{TP}{TP + FN} = \frac{1844}{1844 + 2 + 170} = 0.915} \quad (45)$$

$$\boldsymbol{Recall_{grupos\ de\ rodaballos} = \frac{TP}{TP + FN} = \frac{20}{12 + 20 + 12} = 0.455} \quad (46)$$

Umbral de confianza = 0.7

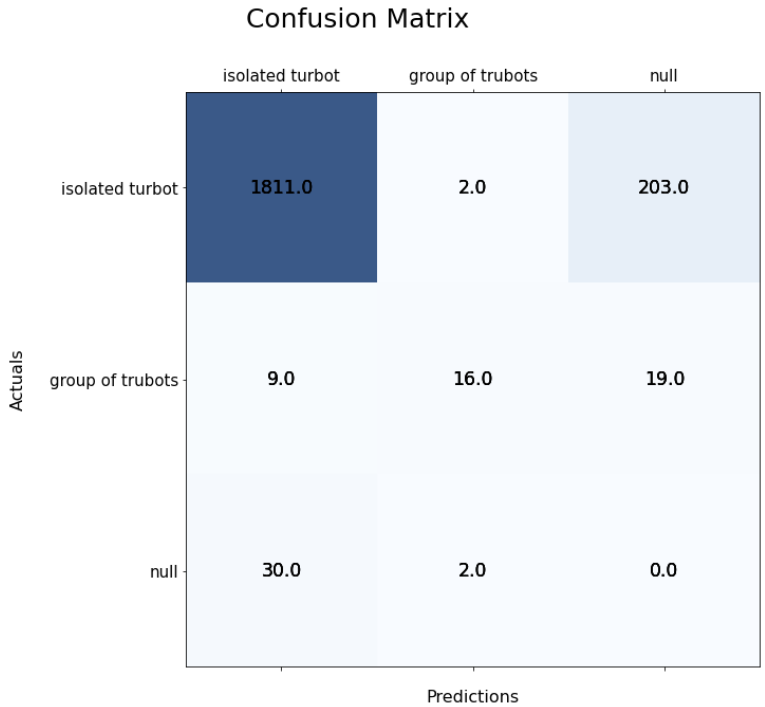


Imagen 94 - Matriz de confusión de la prueba 5 (umbral de confianza = 0.7)

La matriz normalizada obtenida al aumentar el umbral de confianza es la siguiente:

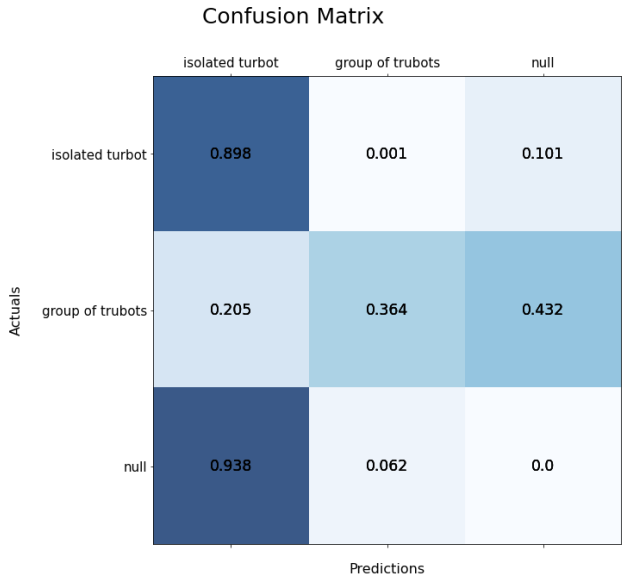


Imagen 95 - Matriz de confusión de la prueba 5 normalizada (umbral de confianza = 0.7)

El cálculo de los valores de la precisión y el *recall* a partir de esta matriz queda de la siguiente manera:

$$\boldsymbol{Precisión_{rodaballo\ aislado} = \frac{TP}{TP + FP} = \frac{1811}{1811 + 9 + 30} = 0.979} \quad (47)$$

$$\boldsymbol{Precisión_{grupo\ de\ rodaballos} = \frac{TP}{TP + FP} = \frac{16}{2 + 16 + 2} = 0.8} \quad (48)$$

$$\boldsymbol{Recall_{rodaballo\ aislado} = \frac{TP}{TP + FN} = \frac{1811}{1811 + 2 + 203} = 0.898} \quad (49)$$

$$\boldsymbol{Recall_{grupos\ de\ rodaballos} = \frac{TP}{TP + FN} = \frac{16}{9 + 16 + 19} = 0.364} \quad (50)$$

En este modelo se puede ver que la precisión de ambas clases y el *recall* de la clase "rodaballo aislado" tienen valores muy similares a los obtenidos en la prueba anterior. Estos tres valores son, en términos generales, peores que los obtenidos en la prueba 4, y algo mejores a los obtenidos en las pruebas anteriores.

Sin embargo, se puede ver cómo el valor del *recall* en la clase "grupo de rodaballos" es considerablemente mayor a la obtenida en el resto de pruebas. Esto quiere decir, que es el modelo de red que menos grupos de rodaballos se deja sin detectar.

En esta ocasión, los datos también siguen variando de la misma forma al aumentar el nivel de confianza (mayor precisión y menor *recall* en ambas clases).

6. LR = 0.01 y num_iteraciones = 6000

Para este último modelo de red la gráficas obtenidas en la fase de entrenamiento se muestran a continuación:

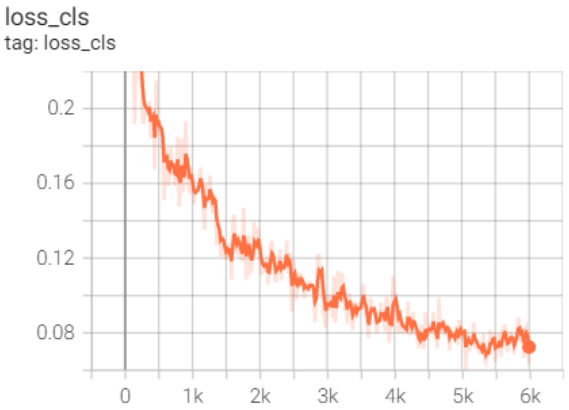


Imagen 96 - Representación de la disminución del error en función de las iteraciones de la fase de entrenamiento (Prueba 6)

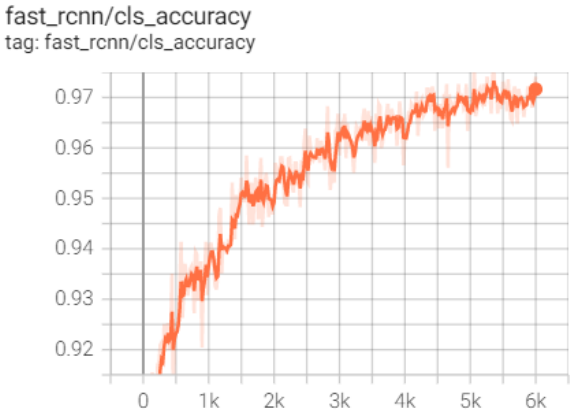


Imagen 97 - Representación de la variación de la precisión en función de las iteraciones de la fase de entrenamiento (Prueba 6)

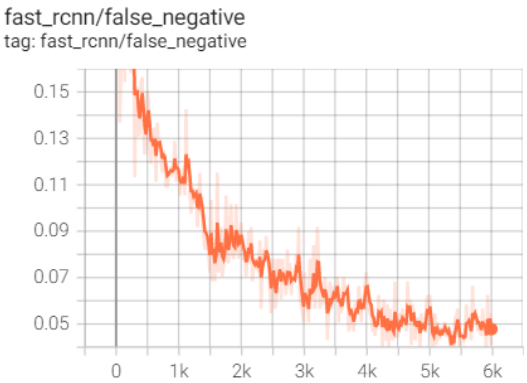


Imagen 98 - Representación de la variación de la tasa de falsos negativos (FN) en función de las iteraciones de la fase de entrenamiento (Prueba 6)

Los parámetros mostrados en estas tres gráficas muestran un valor final muy similar al obtenido en las gráficas de la prueba 4 (por lo que, son mejores que los de la prueba 5), a pesar de que en este modelo el entrenamiento haya tenido 1000 iteraciones más.

Sin embargo, se debe determinar qué modelo tiene mejores resultados analizando las matrices de confianza obtenidas con los datos del set de validación:

Umbral de confianza = 0.5

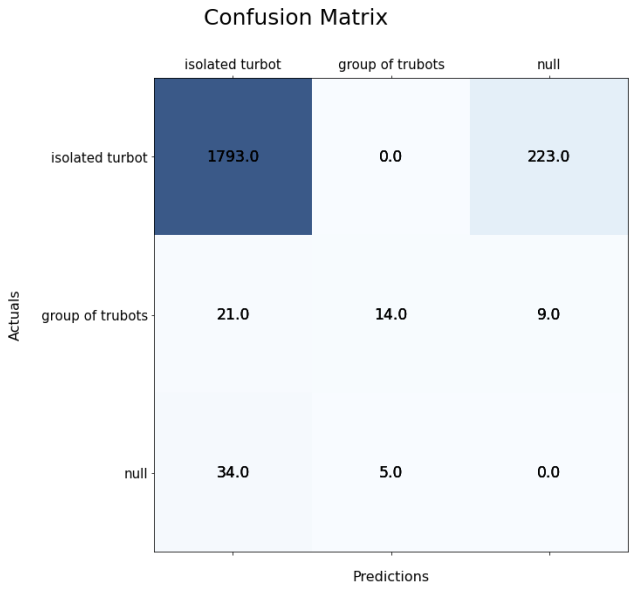


Imagen 99 - Matriz de confusión de la prueba 6 (umbral de confianza = 0.5)

La matriz normalizada obtenida a partir de la anterior matriz es la siguiente:

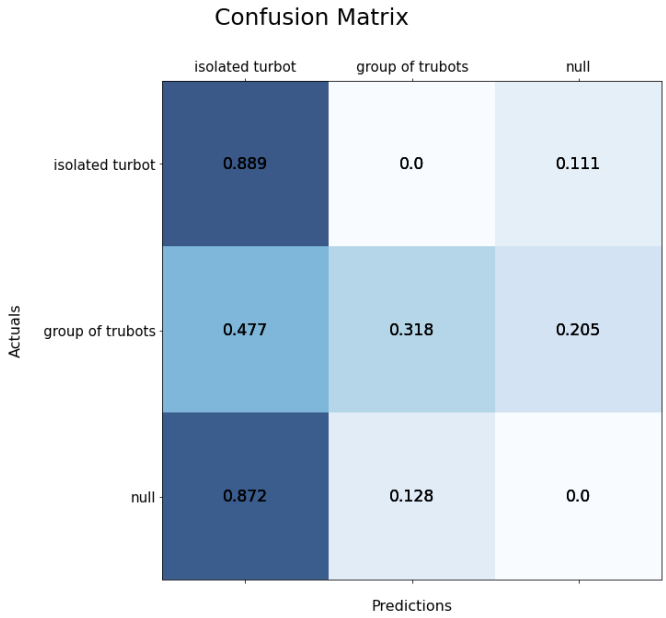


Imagen 100 - Matriz de confusión de la prueba 6 normalizada (umbral de confianza = 0.5)

Por tanto, se obtienen los siguientes valores de precisión y *recall*:

$$\boldsymbol{Precisión_{rodaballo\ aislado} = \frac{TP}{TP + FP} = \frac{1793}{1793 + 21 + 34} = 0.97} \quad (51)$$

$$\boldsymbol{Precisión_{grupo\ de\ rodaballos} = \frac{TP}{TP + FP} = \frac{14}{14 + 5} = 0.73} \quad (52)$$

$$\boldsymbol{Recall_{rodaballo\ aislado} = \frac{TP}{TP + FN} = \frac{1793}{1793 + 223} = 0.889} \quad (53)$$

$$\boldsymbol{Recall_{grupos\ de\ rodaballos} = \frac{TP}{TP + FN} = \frac{14}{21 + 14 + 9} = 0.318} \quad (54)$$

Umbral de confianza = 0.7

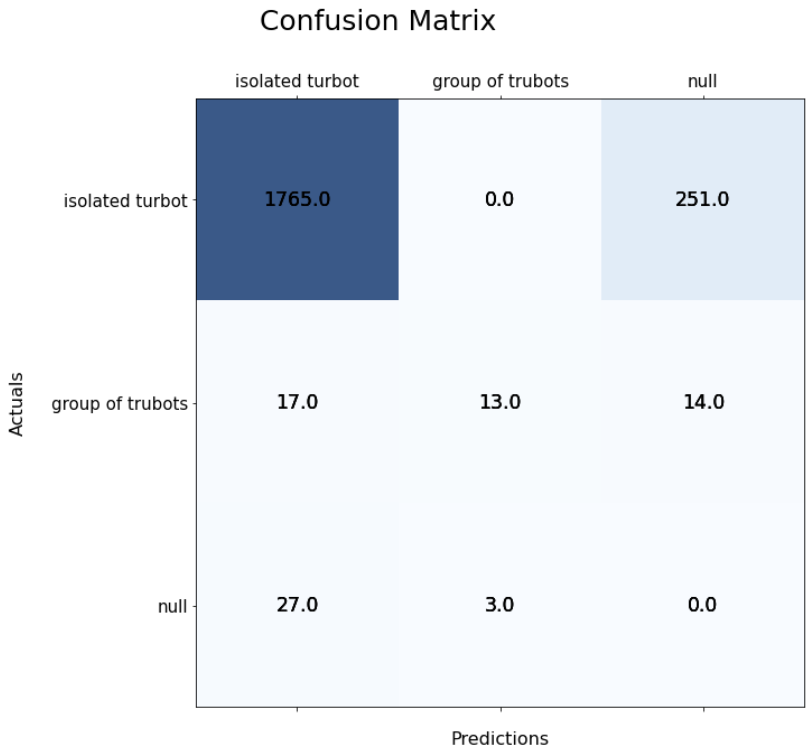


Imagen 101 -Matriz de confusión de la prueba 6 (umbral de confianza = 0.7)

La matriz normalizada resultante es la siguiente:

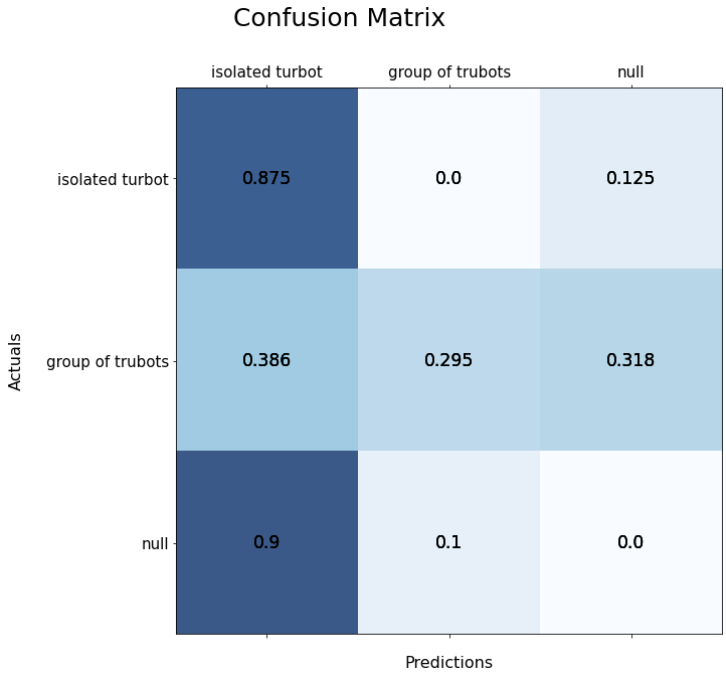


Imagen 102 - Matriz de confusión de la prueba 6 normalizada (umbral de confianza = 0.7)

Partiendo de los resultados de esta matriz se obtienen los siguientes valores de precisión y *recall*:

$$\boldsymbol{Precisión_{rodaballo\ aislado} = \frac{TP}{TP + FP} = \frac{1765}{1765 + 17 + 27} = 0.976} \quad (55)$$

$$\boldsymbol{Precisión_{grupo\ de\ rodaballos} = \frac{TP}{TP + FP} = \frac{13}{13 + 3} = 0.813} \quad (56)$$

$$\boldsymbol{Recall_{rodaballo\ aislado} = \frac{TP}{TP + FN} = \frac{1765}{1765 + 251} = 0.875} \quad (57)$$

$$\boldsymbol{Recall_{grupos\ de\ rodaballos} = \frac{TP}{TP + FN} = \frac{13}{17 + 13 + 14} = 0.295} \quad (58)$$

Este modelo presenta, en términos generales, perores resultados que los obtenidos en las dos pruebas anteriores, independientemente del umbral de confianza seleccionado.

Esto podría deberse a que al incrementar el número iteraciones la red ha comenzado a memorizar, y, por tanto, ha generalizado peor.

5.2. Resumen de los resultados obtenidos

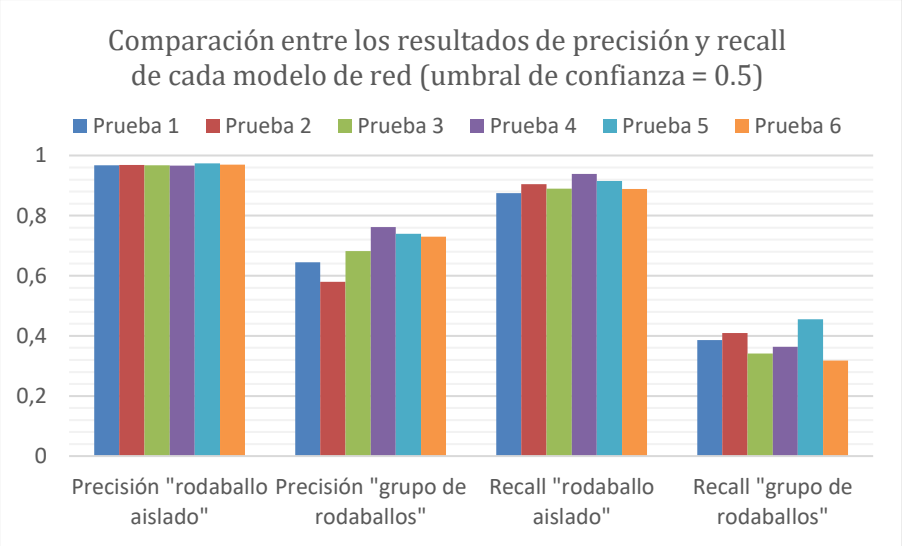
En las seis pruebas realizadas se han recogido dos bloques de datos.

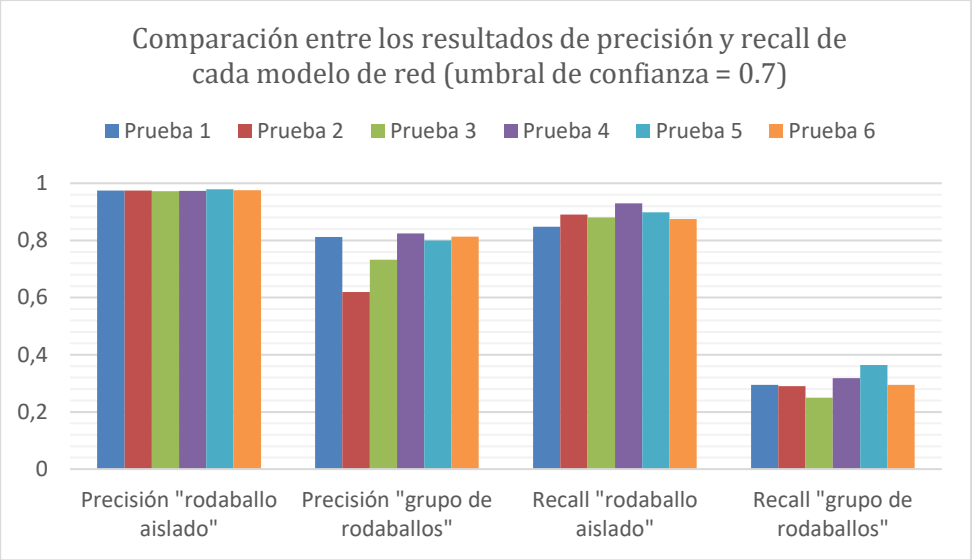
Los datos que componen el primer bloque son la precisión, el error y la tasa de falsos negativos a la que ha conseguido llegar la red al finalizar la fase de entrenamiento. En general, los valores de estos datos han sido muy similares en todas la pruebas realizadas, de hecho, la única prueba que presenta mayores diferencias en este bloque de datos es la prueba 1. Esto se debe a que la diferencia en el valor del número de iteraciones y el *learning rate* entre la prueba 1 y la prueba 2 es mucho mayor que entre las otras pruebas (el número de iteraciones se aumenta en 2000).

En los datos recogidos en estas gráficas, cabe destacar que se pueden apreciar ciertos aspectos de interés. En primer lugar, en la prueba 5 se puede ver que la variación en los parámetros de las gráficas es mucho más brusca debido a que el *learning rate* toma el valor más alto de los contemplados en las seis pruebas realizadas. Esto permite detectar cuando el entrenamiento de la red se está viendo afectado negativamente por un valor de *learning rate* demasiado alto. En segundo lugar, gracias a estas gráficas se puede determinar el número de iteraciones máximo a partir del cual los resultados de la red ya no siguen mejorando. En el caso de este proyecto se puede ver que en la prueba 6 se obtienen resultados muy similares a los de la prueba 4 a pesar de haber aumentado el número de iteraciones en 1000.

Además, estos datos del primer bloque permiten detectar ciertos casos en los que, en la fase de entrenamiento, la red no ha sido capaz de obtener buenos resultados (por ejemplo, porque ha obtenido un error final demasiado alto o una precisión final demasiado baja).

Los datos del segundo bloque, sin embargo, sirven para determinar si la red ha sido capaz de generalizar y aprender a partir de los datos del set de validación. Estos datos están compuestos por el *recall* y la precisión obtenidas en cada una de las clases. Esta información se muestra resumida en los siguientes gráficos.





En estos gráficos se puede ver que la precisión obtenida en la clase “rodaballo aislado” toma valores muy altos en todos los modelos con los que se ha trabajado. Por el contrario, en el *recall* obtenido en la clase “grupo de rodaballos” es, con diferencia, el parámetro que tiene el valor más bajo.

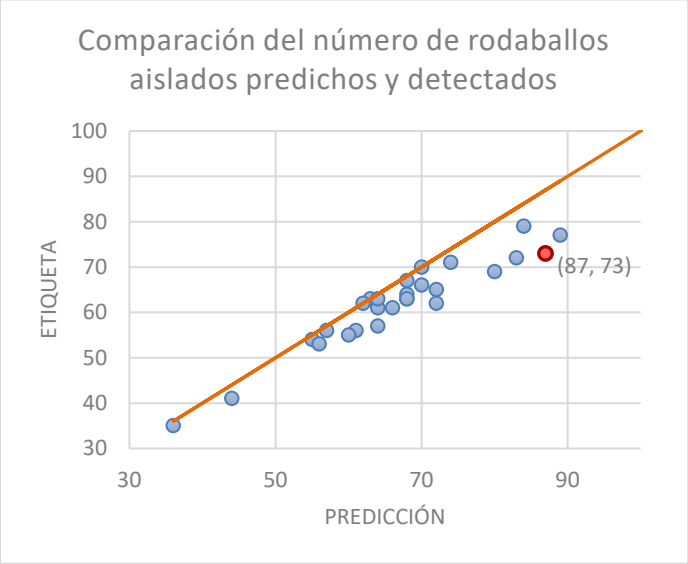
Además, como ya se ha ido mencionando, el valor del umbral de confianza influye de tal manera que cuanto menor sea el umbral seleccionado, mayor será el *recall* y menor la precisión. No obstante, dado que la precisión es bastante alta en todos los casos, los modelos evaluados con un umbral de confianza de 0.5 obtienen mejores resultados.

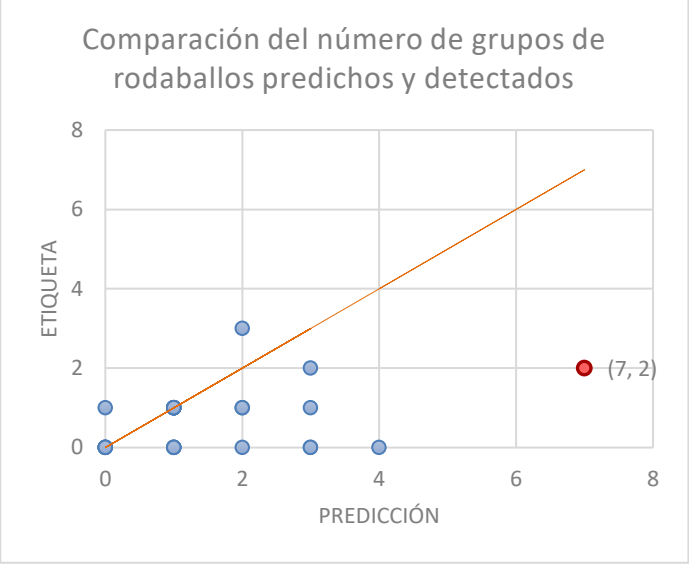
Dentro de estos modelos, los que presentan mejores resultados son el 4 y el 5. En la clase “rodaballo aislado” ambos tienen resultados similares, sin embargo, en la clase “grupo de rodaballo” el modelo 4 tiene mayor precisión y menor *recall* que el 5. Sin embargo, en esta clase, la diferencia en la precisión entre ambos modelos es menor que la diferencia en el *recall*. Por ello, se determina que **el mejor modelo de los seis modelos analizados es el obtenido en la prueba 5**.

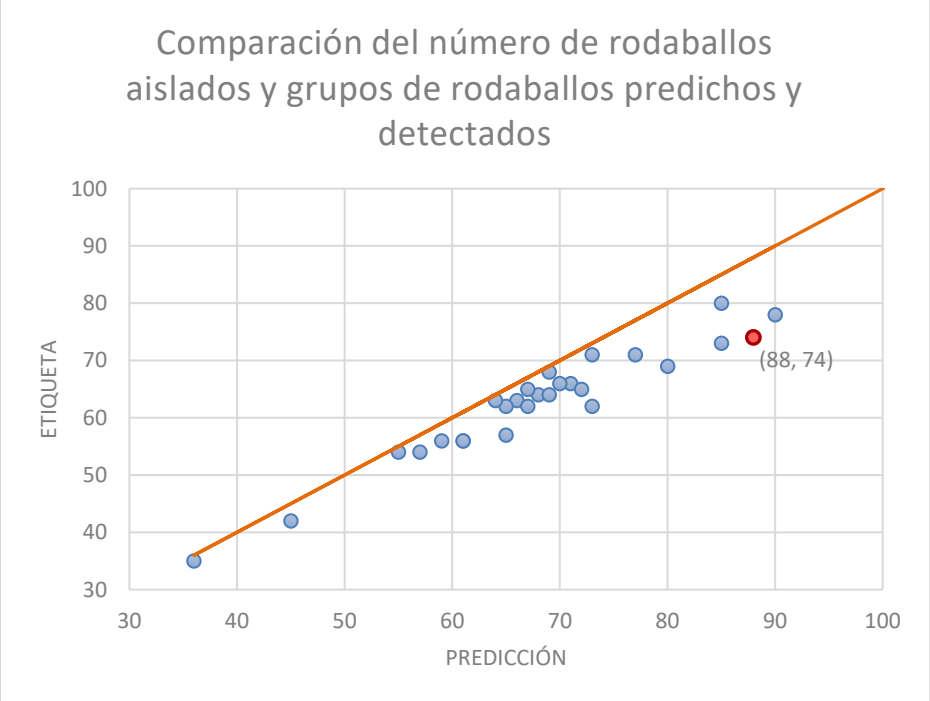
Una vez seleccionado el mejor modelo, se ha evaluado la relación existente entre los valores predichos por la red y las etiquetas asignadas a las imágenes del set de validación. Para ello, los datos obtenidos se han representado en tres gráficos distintos y se ha marcado (en rojo) el punto más alejado de la recta que representa el número real de rodaballos.

En el gráfico que representa el número de grupos de rodaballos detectados se puede ver que el número de grupos de rodaballos etiquetados es muy pequeño. Esto se debe principalmente a que tanto en el set de validación como en el set de entrenamiento se ha trabajado con imágenes con poca densidad de rodaballos, y por tanto, con pocos grupos. Como consecuencia, en la fase de entrenamiento, la red ha tenido pocos ejemplos que le hayan

permitido aprender a detectar grupos de rodaballos, sin embargo, por el mismo motivo esta carencia no ha afectado de forma determinante a los resultados obtenidos por la red.







Los resultados representados en estos tres gráficos confirman las conclusiones obtenidas mediante el análisis de la matriz de confusión del modelo seleccionado: aunque la red tenga más dificultad al detectar a los grupos de rodaballos presentes en las imágenes, la salida de la red se ajusta bastante al número de peces etiquetados en cada imagen.

6. Presupuesto

A continuación, se detalla el precio de cada uno de los recursos que se han necesitado a lo largo del desarrollo de este proyecto. Estos recursos pueden ser tanto materiales, como de mano de obra.

La fase de investigación ha tenido una duración de 1 mes y medio, 20 días/mes, 3,5 h/día.

La fase de desarrollo ha tenido una duración de 3 meses, 20 días/mes y 3,5 h/día.

Tabla 1 - Presupuesto general

Activo	Precio unitario (€)	Unidades	Precio total (€)
Personal de investigación / hora	25	1	1.750
Personal de desarrollo / hora	25	1	5.250
GPU Tesla P100	150	1	150
Ordenador de sobremesa MSI	75	1	75
Pantalla	10	1	10
Licencia Windows 10 Pro	259	1	259
Licencia Matlab de uso académico	425	1	450
Total presupuesto			**7944**

Tabla 2 - Tabla de amortizaciones de equipos informáticos

Activo	Precio total (€)	Amortización
GPU Tesla P100	1.999	150
Ordenador de sobremesa MSI	994	75
Pantalla	130	10

A parte del software de Matlab, el resto de programas utilizados durante el desarrollo del proyecto son de libre uso, por lo que no se han incluido en el presupuesto.

7. Conclusiones

La creación de las redes neuronales ha supuesto una gran revolución en el área del análisis de datos, ya que ha permitido a los ordenadores realizar tareas que, antes del nacimiento de la inteligencia artificial, eran sumamente complejas.

En concreto, las redes neuronales convolucionales han demostrado tener una gran capacidad para extraer datos contenidos en información visual, como las imágenes y los vídeos.

En este proyecto se ha utilizado una red neuronal para detectar rodaballos y grupos de rodaballos contenidos en tanques de piscifactorías partiendo de imágenes captadas con cámaras situadas en la parte superior de los mismos.

Para ello, el proyecto se ha dividido en tres fases: se ha llevado a cabo el etiquetado de las imágenes obtenidas con las cámaras ubicadas en la piscifactoría, se ha realizado un reformateo de los datos obtenidos en el etiquetado y se ha entrenado a una red neuronal convolucional desarrollada por la empresa Meta con los datos obtenidos en el reformateo.

En el desarrollo del proyecto se ha trabajado con dos clases de objetos: “rodaballo aislado” y “grupo de rodaballo”

Los resultados obtenidos con el modelo de red 5 (modelo que presenta los mejores resultados) son satisfactorios especialmente para la detección de objetos de la clase “rodaballo aislado”. Con la clase “grupo de rodaballo” los resultados obtenidos con peores sobre todo si se tiene red ha tenido más dificultades. El problema principal reside en que la red no es capaz de detectar algunos de los objetos pertenecientes a esta clase. Sin embargo, también se puede observar que una buena parte de los grupos de rodaballos no detectados como tal, sí se detectan como rodaballos aislados. Esto es comprensible, ya que hay grupos en los que uno de los peces tapa casi por completo a los otros y los rodaballos aislados pueden tomar formas muy diversas.

Por tanto, se puede concluir que tiene un valor de precisión muy bueno en ambas clases y es capaz de detectar una gran parte de los peces contenidos en las imágenes que recibe como entrada, aunque en ocasiones la clasificación realizada no sea correcta.

Por último, cabe destacar que, tal como se ha mencionado al inicio de este proyecto, este sistema es de gran utilidad para que las piscifactorías puedan llevar un conteo de los ejemplares de los que disponen de cada momento, sin tener que utilizar métodos invasivos que requieran de intervención humana.

Las líneas de trabajo futuras propuestas para esta investigación son fundamentalmente tres. En primer lugar se propone realizar el etiquetado de más imágenes que contengan rodaballos para poder entrenar a la red con una mayor cantidad de datos. En segundo lugar, se propone la utilización del método de la disminución del valor del *learning rate* en un rango determinado. Este método propone hacer variar el valor del *learning rate* en un rango establecido para poder determinar el valor óptimo que debe tomar este parámetro en el sistema desarrollado. Por último, se propone mejorar la segmentación realizada por el programa de etiquetado para que no haya etiquetas que puedan inducir a error a la red.

8. Bibliografía

[1] Y. H. Y. D. Daoliang Li, «Nonintrusive methods for biomass estimation in aquaculture with emphasis on fish: a review,» Reviews in Aquaculture, 2019.

[2] Z. M. F. P. L. W. Y. H. Z. W. T. C. H. L. Y. Z. Daoliang Li, «Automatic counting methods in aquaculture: A review,» Wiley Periodical LLC, 2020.

[3] Oxford University Press, «Oxford Reference,» 2022. [En línea]. Available: https://www.oxfordreference.com/view/10.1093/oi/authority.20110803095426960. [Último acceso: 7 Junio 2022].

[4] M. G. R. G. Valliappa Lakshmanan, Machine Learning for Computer Vision, O'Reilly, 2021.

[5] AlexSoft, «Semi-Supervised Learning, Explained with Examples,» AlexSoft, 18 Marzo 2022. [En línea]. Available: https://www.altexsoft.com/blog/semi-supervised-learning/. [Último acceso: 7 Junio 2022].

[6] IBM Cloud Education, «Machine Learning,» IBM, 15 Julio 2020. [En línea]. Available: https://www.ibm.com/cloud/learn/machine-learning. [Último acceso: 7 Junio 2022].

[7] J. M. Carew, «Reinforcement Learning,» TechTarget, Marzo 2021. [En línea]. Available: https://www.techtarget.com/searchenterpriseai/definition/reinforcement-learning#:~:text=Reinforcement%20learning%20is%20a%20machine,learn%20through%20trial%20and%20error.. [Último acceso: 7 Junio 2022].

[8] J. Hoare, «What is a Decision Tree?,» DisplayR Blog, [En línea]. Available: https://www.displayr.com/what-is-a-decision-tree/. [Último acceso: 7 Junio 2022].

[9] T. Yiu, «Understanding Random Forest,» Towards Data Science, 12 Junio 2019. [En línea]. Available: https://towardsdatascience.com/understanding-random-forest-58381e0602d2. [Último acceso: 7 Junio 2022].

[10] A. Raj, «Unlocking the True Power of Support Vector Regression,» Towards Data Science, 3 Octubre 2020. [En línea]. Available: https://towardsdatascience.com/unlocking-the-true-power-of-support-vector-regression-847fd123a4a0. [Último acceso: 15 Junio 2022].

[11] R. Gandhi, «Support Vector Machine - Introduction to Machine Learning Algorithms,» Towards Data Science, 7 Junio 2018. [En línea]. Available: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47. [Último acceso: 15 Junio 2022].

[12] L. Gebel, «Why We Need Bias in Neural Networks,» Towards Data Science, 21 Agosto 2021. [En línea]. Available: https://towardsdatascience.com/why-we-need-bias-in-neural-networks-db8f7e07cb98. [Último acceso: 8 Junio 2022].

[13] E. Kavakoglu, «AI vs. Machine Learning vs. Deep Learning vs. Neural Networks: What's the difference?,» IBM, 27 Mayo 2020. [En línea]. Available: https://www.ibm.com/cloud/blog/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks. [Último acceso: 13 Junio 2022].

[14] C. Thomas, «An Introduction to Convolutional Neural Networks,» Towards Data Science, 27 Mayo 2019. [En línea]. Available: https://towardsdatascience.com/an-introduction-to-convolutional-neural-networks-eb0b60b58fd7. [Último acceso: 13 Junio 2022].

[15] A. Borna, «ResNeXt Explained, Part 1,» Medium, 4 Octubre 2021. [En línea]. Available: https://medium.datadriveninvestor.com/resnext-explained-part-1-17a6b510fb0a. [Último acceso: 14 Junio 2022].

[16] N. Adaloglou, «Intuitive Explanation of Skip Connections in Deep Learning,» AI Summer, 23 Marzo 2020. [En línea]. Available: https://theaisummer.com/skip-connections/. [Último acceso: 14 Junio 2022].

[17] R. Gandhi, «R-CNN, Fast R-CNN, Faster R-CNN, YOLO - Object Detection Algorithms,» Towards Data Science, 9 Julio 2018. [En línea]. Available: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e. [Último acceso: 15 Junio 2022].

[18] Common Object in Context (COCO), «COCO Dataset,» [En línea]. Available: https://cocodataset.org/#home.

[19] Common Objects in Context (COCO), «Data format,» [En línea]. Available: https://cocodataset.org/#format-data.

[20] Internet Engineering Task Force (IETF), «The JavaScript Object Notation (JSON) Data Interchange Format,» [En línea]. Available: https://datatracker.ietf.org/doc/html/rfc7159.

[21] Meta, «Detectron2,» [En línea]. Available: https://colab.research.google.com/drive/16jcaJoc6bCFAQ96jDe2HwtXj7BMD_-m5.

[22] Stack Overflow, «Build confusion matrix for instance segmantation (mask r-cnn from detectron2),» [En línea]. Available: https://stackoverflow.com/questions/71475164/build-confusion-matrix-for-instance-segmantation-mask-r-cnn-from-detectron2.

[23] IBM Cloud Education, «Redes Neuronales,» IBM, 17 Agosto 2020. [En línea]. Available: https://www.ibm.com/pe-es/cloud/learn/neural-networks. [Último acceso: 8 Junio 2022].

[24] P. Baheti, «12 Types of Nueral Network Activation Funtions: How to Choose?,» V7 Labs, 26 Mayo 2022. [En línea]. Available: https://www.v7labs.com/blog/neural-networks-activation-functions. [Último acceso: 8 Junio 2022].

[25] M. J. B. R. X. C. G. P. F. Divish Rengasamy, «Deep Learning with Dynamically Weighted Loss Function for Sensor-Based Prognostics and Health Management,» 2020.

[26] Developers Google, «Machine Learning Glossary,» Developes Google, [En línea]. Available: https://developers.google.com/machine-learning/glossary. [Último acceso: 10 Junio 2022].

[27] D. Mechea, «What is Panoptic Segentation and why you should care,» Medium, 29 Enero 2019. [En línea]. Available: https://medium.com/@danielmechea/what-is-panoptic-segmentation-and-why-you-should-care-7f6c953d2a6a. [Último acceso: 20 Mayo 2022].

[28] S. Saha, «A Comprenhensive Guide to Convoutional Networks,» Towards Data Science, 15 Diciembre 2018. [En línea]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. [Último acceso: 13 Junio 2022].

[29] R. G. P. D. Z. T. K. H. Saining Xie, «Aggregated Residual Transformactions for Deep Neural Networks,» 2017.

[30] Microsoft, «Evaluate automated machine learning experiment results,» Microsoft, 24 Mayo 2022. [En línea]. Available: https://docs.microsoft.com/en-us/azure/machine-learning/how-to-understand-automated-ml. [Último acceso: 2022 Junio 25].

[31] K. Ahirway, «Everything you need to know about neural networks,» HackerNoon, 1 Noviembre 2017. [En línea]. Available: https://medium.com/ravenprotocol/everything-you-need-to-know-about-neural-networks-6fcc7a15cb4. [Último acceso: 8 Junio 2022].

[32] A. Shaffi, «Building a Confusion Matrix from Scratch,» Medium, 16 Marzo 2021. [En línea]. Available: https://medium.datadriveninvestor.com/building-a-confusion-matrix-from-scratch-85a8bfb97626. [Último acceso: 25 Junio 2022].

[33] J. Hui, «mAP (mean Aveage Precision) for Object Detection,» Medium, 7 Marzo 2018. [En línea]. Available: https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173. [Último acceso: 25 Junio 2022].