ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y SISTEMAS DE TELECOMUNICACIÓN

PROYECTO FIN DE GRADO

TÍTULO: CONTEO DE OBJETOS EN IMÁGENES RGB CON REDES CONVOLUCIONALES MEDIANTE DIVISIÓN ESPACIAL

AUTOR: ALEJANDRO GONZÁLEZ FERNÁNDEZ

TITULACIÓN: GRADO EN INGENIERÍA ELECTRÓNICA DE COMUNICACIONES

TUTOR: JUANA MARIA GUTIÉRREZ ARRIOLA

DEPARTAMENTO: DEPARTAMENTO DE INGENIERÍA TELEMÁTICA Y ELECTRÓNICA

Firmado digitalmente por GUTIERREZ ARRIOLA, JUANA MARIA (FIRMA)
Fecha: 2022.06.20 14:43:28 +02'00'

VºBº

Miembros del Tribunal Calificador:

PRESIDENTE: MANUEL VÁZQUEZ LÓPEZ

TUTOR: JUANA MARIA GUTIÉRREZ ARRIOLA

SECRETARIO: VÍCTOR JOSÉ OSMA RUIZ

Fecha de lectura:

Calificación:

El Secretario,

Resumen

En acuicultura, la estimación de la biomasa es esencial para la viabilidad de las empresas, tanto para poder evaluar los procesos de cría y crecimiento como para poder determinar la cantidad de alimento a suministrar en cada tanque, con el consiguiente beneficio ecológico que conlleva evitar el desperdicio del mismo. Uno de los factores clave para la estimación de biomasa es la determinación del número de peces en los tanques de cría.

Para esta labor, se han empleado, durante la historia de la acuicultura, diferentes métodos, desde el conteo manual hasta otros más tecnológicos, como los basados en radares. Sin embargo, el desarrollo de las técnicas de aprendizaje máquina en los últimos años y los avances en la aplicación de estas técnicas a imágenes y videos, ha abierto la puerta a implementar estos algoritmos para tratar de resolver el problema de conteo de peces de una forma eficiente y no intrusiva.

Los principales avances que son de aplicación en este campo vienen de la mano de desarrollos en el ámbito de conteo de personas, cuyas últimas propuestas implementan redes neuronales convolucionales para estimar la densidad y número de individuos en imágenes y videos. En este sentido, aunque limitados en número, ya existen algunas aproximaciones al empleo de estas técnicas en el ámbito de la acuicultura.

El presente proyecto tiene como objetivo adaptar e implementar uno de los modelos que en los últimos años ha mostrado su eficacia en el conteo de personas, para evaluar su rendimiento en el conteo de crías de rodaballos en tanques de acuicultura. En concreto, se ha implementado el modelo SS-DCNet, que utiliza una aproximación basada en la estimación de densidad con la división espacial y que se encuentra entre los algoritmos que mejor resultado han demostrado, no solo en el conteo de personas, sino también de vehículos o mazorcas de maíz.

La aplicación de este algoritmo para el conteo de crías de rodaballo ha demostrado una precisión destacable, con un error medio inferior al 3.5%. Además, se ha observado la flexibilidad del modelo para el conteo en imágenes con mayor densidad y mayor número de rodaballos que los presentes en el conjunto de entrenamiento. Por último, otro de los beneficios constatados del modelo es que no necesita un conjunto de datos grande para generar resultados aceptables.

Abstract

In aquaculture, biomass estimation is essential for the viability of enterprises, both to be able to evaluate breeding and growth processes and to be able to determine the amount of food to be supplied in each tank, with the consequent ecological benefit of avoiding food waste. One of the key factors for biomass estimation is the determination of the number of fish in the tanks.

Different methods have been used throughout the history of aquaculture, ranging from manual to more technological methods such as radar-based ones. However, the development of machine learning techniques in recent years and advances in the application of these techniques to images and video, have opened the door to implementing these algorithms to try to solve the fish counting problem in an efficient and non-intrusive way.

The main advances that are applicable in this field come from developments in the field of crowd counting, whose latest proposals implement convolutional neural networks to estimate the density and number of people in images and videos. In this sense, although limited in number, there are already some approaches that the use these techniques in the field of aquaculture.

The aim of this project has been to adapt and implement one of the models that, in recent years, have shown its effectiveness in crowd counting, in order to evaluate their performance in counting turbot fry in aquaculture tanks. Specifically, the SS-DCNet model has been implemented, which uses an approach based on density estimation and spatial division and is among the algorithms that have shown the best results, not only for counting people, but also for counting vehicles or corn mazes.

The application of this algorithm for counting turbot fry has shown remarkable accuracy, with an average error of less than 3.5%. In addition, the flexibility of the model has been observed for counting in images with higher density and higher number of turbot than those present in the training set. Finally, another benefit of the model is that it does not require a large dataset to generate acceptable results.

Contenido

1. Introducción 1
2. Marco teórico 3
2.1. Aprendizaje máquina 3
2.2. Redes neuronales 4
2.3. Redes neuronales convolucionales 6
2.3.1. Arquitectura de redes convolucionales 10
2.4. Aprendizaje máquina en el conteo de objetos en imágenes 12
3. Especificaciones y restricciones de diseño 15
4. Descripción de la solución propuesta 17
4.1. Elección del modelo 17
4.1.1. SS-DCNET 18
4.2. Arquitectura 21
4.2.1. Entorno de desarrollo y entrenamiento 23
4.2.2. Generación de mapas de densidad 24
4.2.3. Entrenamiento 25
4.3. Conjunto de datos 26
4.3.1. Adaptación del archivo de etiquetas 28
4.3.2. Conjuntos de datos para entrenamiento y validación 29
4.4. Parámetros de optimización 31
4.4.1. Valor de desviación estándar para generación de mapas de densidad 31
4.4.2. Dimensión del clasificador 34
5. Resultados 37
5.1. Impacto de la desviación estándar en la generación de mapas de densidad 38
5.2. Determinación del valor de C_{Max} 39
5.3. Generalización para imágenes con más objetos 40
5.4. Evolución del error con el tamaño del conjunto de datos 42
5.5. Medición de la evolución del número de rodaballos en los vídeos 43
6. Conclusiones 45
Anexo 1. Presupuesto 51
Anexo 2. Manual de usuario 53

Índice de figuras

Figura 1: Distribución de los artículos científicos que aplicaban técnicas de IA a la acuicultura a principios de 2020. Fuente: [5]. 2
Figura 2: Tipos de algoritmos de aprendizaje máquina. Fuente: [7]. 4
Figura 3: Funcionamiento de un perceptrón. Fuente: [8]. 4
Figura 4: Funciones de activación Sigmoid y ReLU. Fuente: [9]. 5
Figura 5: Ejemplarización de diferentes valores de learning rate en el algoritmo de descenso por gradiente. Fuente: [10]. 6
Figura 6: Ejemplo de red neuronal. Fuente: [12]. 6
Figura 7: Ejemplo de proceso de convolución. Fuente: [14]. 7
Figura 8: Ejemplo de convolución con diferentes valores de stride. Fuente: [15]. 8
Figura 9: Ejemplo de convolución con padding de 1 en la entrada. Fuente: [16]. 8
Figura 10: Ejemplo de convolución sobre volumen. Fuente: [17]. 8
Figura 11: Ejemplo de operación de max. pooling. Fuente: [19]. 9
Figura 12: Ejemplo de arquitectura de red convolucional. Fuente: [21]. 9
Figura 13: Arquitectura de la red VGG16. Fuente: [23]. 10
Figura 14: Ejemplo de segmentación semántica aplicada al diagnóstico de tumores cerebrales. Fuente: [25]. 11
Figura 15: Arquitectura de la red U-Net. Fuente: [26]. 11
Figura 16: Ejemplo de modelo de CNN para estimación de densidad. Fuente: [32]. 12
Figura 17: Ejemplo de mapas de densidad generados a partir de imágenes etiquetadas. Fuente: [28]. 13
Figura 18: Ilustración de la estrategia de división espacial empleada por SS-DCNet. Fuente: [41] ... 18
Figura 19: Arquitectura del modelo SS-DCNet. Los dos primeros bloques de VGG16 se han simplificado con el bloque 'Conv'. Fuente: [41]. 19
Figura 20: Primer proceso de división espacial de SS-DCNet. Fuente: [41]. 20
Figura 21: Segundo proceso de división espacial de SS-DCNet. Fuente: [41]. 21
Figura 22: Ejemplo de funciones kernel gaussiano 2D para valores de desviación estándar diferentes. Fuente: [45]. 25
Figura 23: Ejemplo de resultado del proceso de normalización de la imagen. 26
Figura 24: Ejemplo de dos fotogramas del conjunto de datos. 27
Figura 25: Ejemplo de imagen del conjunto de datos con la información de etiquetado superpuesta. En rojo el contorno de rodaballos individuales y en azul los conjuntos de rodaballos. 27
Figura 26: Distribución del número de rodaballos etiquetados por fotograma. 28
Figura 27: Estructura de los archivos de etiquetas para el entrenamiento del modelo. 29
Figura 28: Distribución del número de rodaballos por imagen en el conjunto de imágenes de entrenamiento. 29
Figura 29:Distribución del número de rodaballos por imagen en el conjunto de imágenes de evaluación. 30
Figura 30: Estructura de archivos del conjunto de datos para entrenamiento y validación. 31
Figura 31: Imagen del conjunto de datos y visualización de sus mapas de densidad generados con valores de sigma 3, 6, 9 y 12. 33
Figura 32: Distribución de la longitud de los rodaballos etiquetados en imágenes de tamaño 1024x768. 34
Figura 33: Distribución del número de rodaballos por cuadros de 64x64 píxeles. 34
Figura 34: Evolución del MAE durante el entrenamiento con valores de sigma = 12 y C_{Max} = 5. 38

Figura 35: Evolución de los errores en el modelo (con CMax = 5) con el tamaño de la desviación estándar del kernel gaussiano en la generación de los mapas de densidad.....38
Figura 36: Evolución de los errores con el tamaño de C_{Max} para un valor de desviación estándar constante de 12. 39
Figura 37: Proyección del valor de la etiqueta frente al valor predicho para los conjuntos de entrenamiento (naranja) y de test (azul).....40
Figura 38: Ejemplo de resultado de la predicción (derecha) sobre una imagen del conjunto de test (izquierda). 40
Figura 39: Proyección del valor de la etiqueta frente al valor predicho para los conjuntos de entrenamiento (naranja) y de test (azul).....41
Figura 40: Evolución del error con el tamaño del conjunto de datos usado para el entrenamiento del modelo. 42
Figura 41: Evolución de la predicción del número de rodaballos en los vídeos analizados. 44

Índice de tablas

Tabla 1: Errores en la implementación del modelo SS-DCNet sobre el conjunto de datos ShangaiTech. 22
Tabla 2: Descripción de la estructura del archivo generado por el programa de etiquetado Turbot. 28
Tabla 3: Percentiles de la distribución de longitudes de los rodaballos etiquetados. 32
Tabla 4: Percentiles de la distribución del número de rodaballos por cuadros de 64x64 píxeles. 35
Tabla 5: Errores para mapas de densidad generados con diferentes valores de desviación estándar. 38
Tabla 6: Errores para diferentes valores de C_{Max} con un valor fijo de desviación estándar de 12. 39
Tabla 7: Errores del modelo entrenado con el conjunto de datos generado en base a la densidad de objetos en las imágenes. 41
Tabla 8: Comparación de errores entre ambos entrenamientos sobre las 9 imágenes de test comunes. 41
Tabla 9: Errores en los modelos entrenados con 10, 25, 50 y 124 imágenes. 42
Tabla 10: Detalle de las desviaciones y los errores en los vídeos analizados. 43

1. Introducción

En acuicultura, la capacidad de obtener una estimación fiable de biomasa es muy importante. Esta estimación de biomasa se obtiene como el total de peces en un área multiplicado por el peso medio de los mismos. Este dato es extremadamente relevante para la viabilidad de las industrias del sector de la acuicultura, debido a que permite optimizar recursos, por ejemplo, calculando correctamente la cantidad de alimento a proporcionar a los peces, evitando así que sobre comida y que esta afecte negativamente a la calidad del agua. Además, la biomasa permite a estas empresas optimizar el desarrollo de sus estrategias de negocio.

Durante la historia de la acuicultura, se han empleado una variedad de métodos para la estimación del número de peces en diferentes contenedores: desde métodos de conteo manual, que no solo afectan al bienestar y el crecimiento de los peces, sino que además cuentan con un alto grado de imprecisión (error entre el 15% y el 25%) [1]; hasta métodos que, haciendo uso de los diferentes avances tecnológicos, han tratado de mejorar la eficiencia del conteo y evitar, al mismo tiempo, perjudicar el desarrollo de los peces.

Entre los métodos no intrusivos desarrollados en las últimas décadas se encuentran soluciones basadas en visión artificial, tanto en el espectro visible como en el infrarrojo, aunque en muchos de estos casos se requería la circulación de los peces por estructuras donde se implementaba el conteo y presentaban grandes limitaciones para resolver los problemas de solapamiento. Otro de los campos que se ha explorado para el conteo es la acústica, tanto pasiva, midiendo la densidad por el ruido generado por los peces, como activa, estudiando el rebote de las ondas sonoras. Estos métodos presentan grandes inconvenientes como poca fiabilidad en el primer caso, y un alto coste en el segundo.

También se han explorado métodos de conteo indirecto como el análisis de muestras de ADN de los tanques, estimando el número de peces por los niveles de concentración de ADN; así como métodos de estimación de densidad a través de la resistividad, si bien, en este caso, también se requiere que los peces circulen por una estructura reducida o canal, donde se contabiliza su presencia [2].

En las últimas décadas, la Inteligencia Artificial (IA) se ha desarrollado de forma notable y ha ayudado a alcanzar soluciones para gran cantidad de retos, tanto industriales como sociales. Por IA se entiende “sistemas de software (y posiblemente también de hardware) diseñado por humanos que, ante un objetivo complejo, actúan en la dimensión física o digital: percibiendo su entorno, a través de la adquisición e interpretación de datos estructurados o no estructurados, razonando sobre el conocimiento, procesando la información derivada de estos datos y decidiendo las mejores acciones para lograr el objetivo dado. Los sistemas de IA pueden usar reglas simbólicas o aprender un modelo numérico, y también pueden adaptar su comportamiento al analizar cómo el medio ambiente se ve afectado por sus acciones previas” [3].

La Estrategia de Inteligencia Artificial española destaca esta disciplina por su “gran potencial de transformación desde el punto de vista tecnológico, económico, ambiental y social dada su penetración intersectorial, elevado impacto, rápido crecimiento y contribución a la mejora de la competitividad” [4].

El sector de la acuicultura no ha quedado ajeno a los avances de la IA y son varios los desarrollos en los que se han aplicado con éxito estas nuevas tecnologías, como en la predicción de la calidad del agua, la identificación de ejemplares y diferenciación de especies, el diagnóstico de enfermedades y la monitorización del bienestar de ejemplares, la asistencia en las necesidades de alimentación y, por

supuesto, en la estimación de biomasa. En este sentido, a principios de 2020, el 17% de los artículos científicos que aplicaban técnicas de IA a la acuicultura desarrollaban mecanismos para la estimación de biomasa [5] (**Figura 1**).

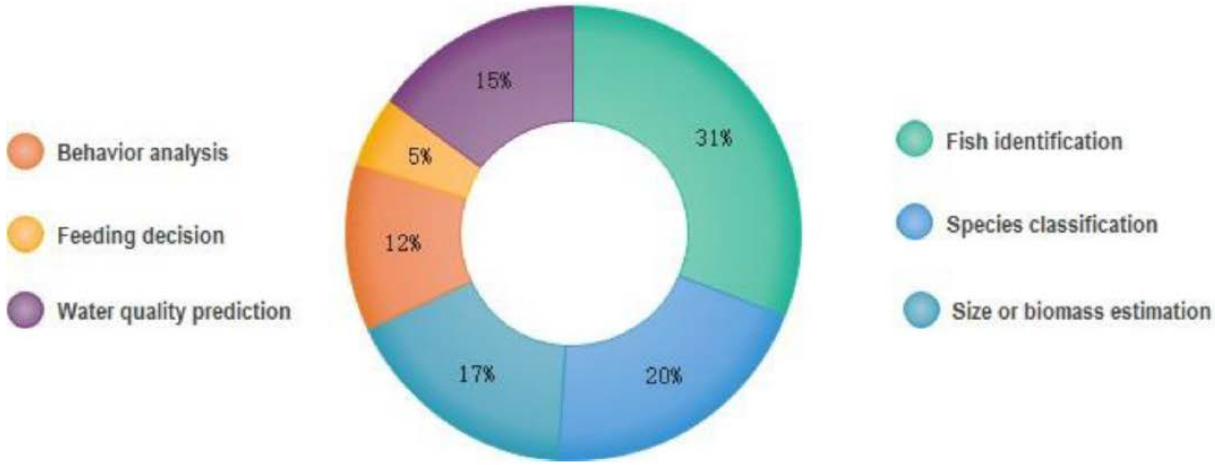

***Figura 1**: Distribución de los artículos científicos que aplicaban técnicas de IA a la acuicultura a principios de 2020. Fuente: [5].*

La aplicación de algoritmos de IA sobre imágenes de tanques de cría de peces puede permitir la implantación de sistemas de estimación de biomasa baratos y flexibles, ya que tanto la tecnología de captación de imágenes como los servicios de computación se han generalizado y abaratado. Por ello, este proyecto explorará las oportunidades que algoritmos de IA, como las redes neuronales convolucionales, ofrecen para la estimación del número de peces en tanques de cría a partir de las imágenes de los mismos.

El proyecto tendrá como objetivo el desarrollo de un sistema que permita, a partir de imágenes RGB, realizar una cuenta estimada de los objetos que aparecen en la misma. Para ello, se implementará un algoritmo de aprendizaje profundo que sea capaz de, partiendo un entrenamiento con imágenes etiquetadas de tanque de cría de peces, estimar el número de elementos presentes, teniendo en cuenta que los tamaños pueden ser variables por distintas profundidades en la imagen y que pueden existir objetos agrupados y superpuestos.

Las imágenes utilizadas en el desarrollo de este proyecto se generan gracias a la colaboración entre el Centro de Investigación en Tecnologías del Software y Sistemas Multimedia para la Sostenibilidad y la empresa Insuiña S.L. del grupo Nueva Pescanova y al proyecto Acuicultura 4.0 financiado por el Ministerio de Agricultura, Pesca y Alimentación.

La presente memoria se estructura de la siguiente manera: el capítulo 2 recoge los principales conceptos teóricos sobre los que se sustenta el proyecto, desde el funcionamiento de las redes neuronales hasta el uso de algoritmos avanzados de IA para la estimación de densidad en imágenes. El capítulo 3 desarrolla las especificaciones y restricciones para el diseño de la solución propuesta, la cual se describe en el capítulo 4. Los resultados son expuestos y valorados en el capítulo 5 y, finalmente, en el capítulo 6 se recogen las conclusiones del proyecto y se exponen las posibles líneas futuras de investigación.

2. Marco teórico

El presente capítulo desarrolla ciertos conceptos teóricos sobre el aprendizaje máquina, las redes neuronales y las estrategias que, en este ámbito, se han desarrollado para contar objetos en imágenes. El objetivo es facilitar la comprensión de los aspectos técnicos del desarrollo que se detallan en el capítulo 4.

2.1. Aprendizaje máquina

El aprendizaje máquina o, en inglés, *machine learning* es la ciencia de programar sistemas que sean capaces de aprender a partir de datos. Una definición más detalla de lo que se entiende como aprendizaje es que una "máquina es capaz de aprender a partir de una experiencia E respecto a una tarea T y una métrica de desempeño P, si su desempeño en la realización de la tarea T, medido por P, mejora con la experiencia E" [6]. A fin de ejemplarizar este proceso, si se considera una tarea T como la clasificación de correos no deseados y una métrica de desempeño P como el error entre la el número de correos no deseados clasificados como deseados y viceversa, un algoritmo de aprendizaje máquina será capaz de mejorar su desempeño en base a un conjunto de datos de correos electrónicos sobre los que ya se haya determinado si son o no correo no deseado. Este conocimiento sobre la tarea de clasificación de correos podrá ser utilizado para clasificar futuros correos.

Los algoritmos de aprendizaje máquina se pueden dividir en dos grandes grupos (**Figura 2**):

- El **aprendizaje supervisado** parte de la base de un conjunto de datos etiquetados, por ejemplo, en el caso anterior, correos electrónicos que han sido etiquetados como correo no deseado o no. El algoritmo trata entonces de encontrar un patrón o modelo en esos datos, de forma que en el futuro pueda predecir la etiqueta de nuevos datos o, en el caso descrito, correos. Existen dos tipos principales de aprendizaje supervisado, la clasificación, cuando el resultado a predecir es una categoría (por ejemplo, correo deseado o no deseado, la raza de un perro, etc.), y la regresión, cuando el resultado es predecir un valor numérico (por ejemplo, la edad de una persona, el número de objetos en una imagen, etc.).
- El **aprendizaje no supervisado** se basa en descubrir patrones en los datos, sin necesidad de que estos estén etiquetados. Entre las principales aplicaciones de estos algoritmos se encuentra el agrupamiento o *clustering*, con el objetivo de identificar características de los datos que permitan agruparlos por similitud (por ejemplo, para los sistemas de recomendación en tiendas online o redes sociales). Otra de las tareas más comunes de este tipo de algoritmos es la reducción de la dimensión de las características de los datos, esto es, identificar características que tienen correlación a fin de poder unirlas y reducir el número de características del conjunto de datos, lo cual permite reducir los tiempos de computación y mejorar el rendimiento de los modelos. Por último, también se emplean para la detección de anomalías, por ejemplo, en transacciones con tarjetas bancarias.

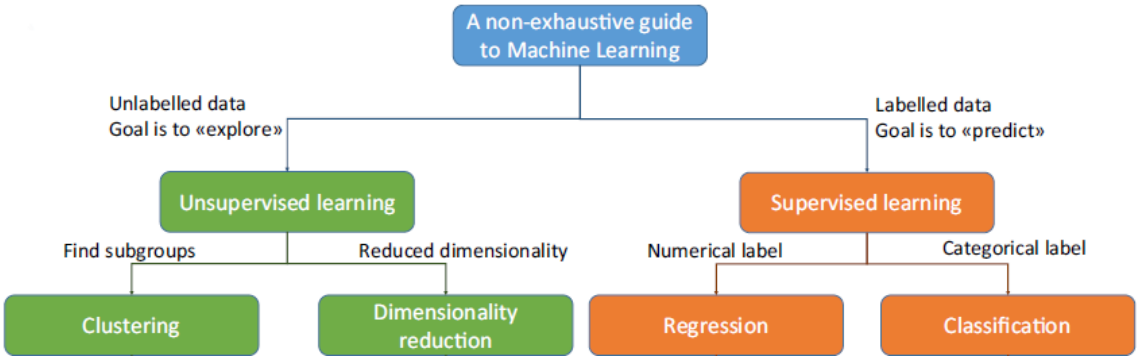

***Figura 2**: Tipos de algoritmos de aprendizaje máquina. Fuente: [7].*

Además de los dos tipos de aprendizaje máquina antes expuestos, también existen modelos de **aprendizaje semi-supervisado**, donde se trabaja con datos parcialmente etiquetados (por ejemplo, un sistema de reconocimiento de personas en imágenes de redes sociales, puede agrupar las imágenes de personas iguales y solo necesita que una de esas esté etiquetada para identificar a esa persona en el resto de fotos). Por último, se encuentran los modelos de **aprendizaje por recompensa**, donde el programa ejecuta acciones y recibe recompensas, en base a las cuales modifica su patrón de comportamiento futuro. Este tipo de aprendizaje se implementa, por ejemplo, para que los robots aprendan acciones como caminar.

2.2. Redes neuronales

Las redes neuronales en el campo de la computación se desarrollaron a mediados de del siglo pasado basándose en el funcionamiento de las neuronas biológicas. La unidad básica de estas redes es el perceptrón (**Figura 3**). El cual, partiendo del sumatorio de una serie de entradas x_1, x_2, …, x_n, ponderadas por una serie de pesos w_1, w_2, …, w_n y al que se le añade un valor de *offset* o *bias* b, es capaz de generar una salida y, mediante una función de activación σ.

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

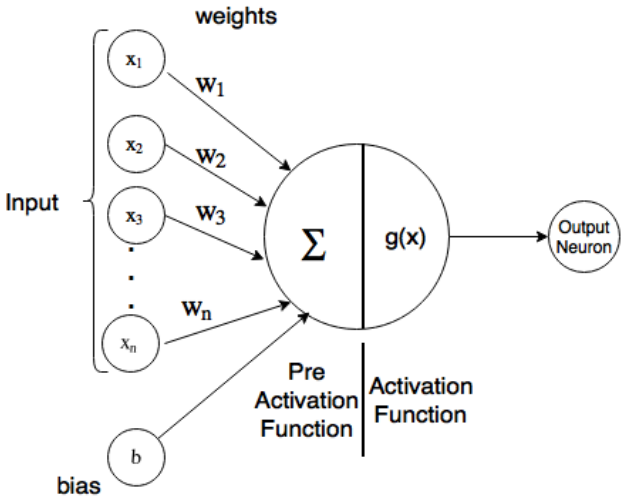

***Figura 3**: Funcionamiento de un perceptrón. Fuente: [8].*

Entre las funciones de activación más comunes se encuentran:

- Función sigmoide: es una función no lineal (2) cuya salida tiene forma de S, tal como se observa en la **Figura 4**.

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2}$$

- Función ReLU: es una función no lineal (3) que asigna el valor de 0 para todos los valores negativos. Esta función es generalmente usada en las capas ocultas de las redes neuronales.

$$\sigma(z) = \begin{cases} 0 \; si \; z < 0 \\ z \; si \; z \geq 0 \end{cases} \tag{3}$$

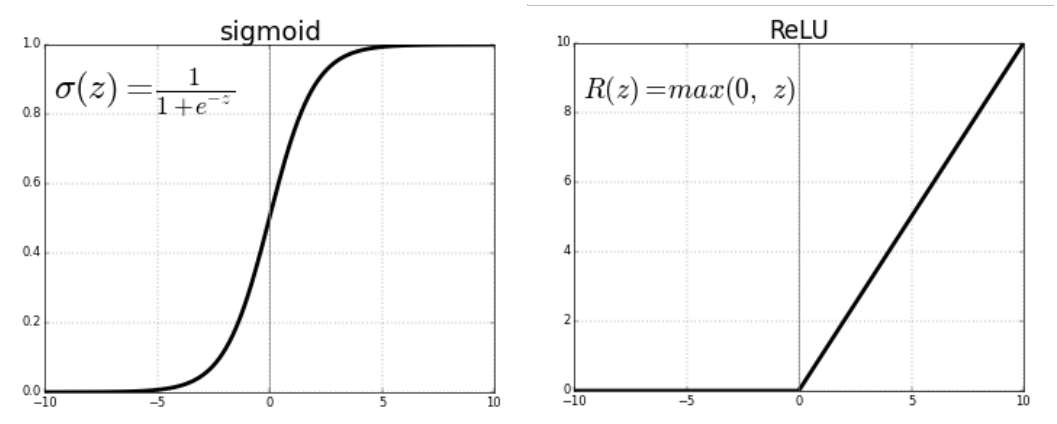

***Figura 4**: Funciones de activación Sigmoid y ReLU. Fuente: [9].*

El aprendizaje en estas redes se produce mediante la adaptación de los pesos de cada una de las neuronas. Para el proceso de entrenamiento, en primer lugar, es necesario definir una función de perdida (*loss function*), que medirá el error entre el resultado esperado y el calculado por la neurona para cada iteración del entrenamiento. Ejemplo de función de perdida es la Raíz del Error Cuadrático Medio (RMSE). Una vez elegida la función de perdida, los pesos de la neurona se suelen inicializar, por ejemplo, de forma aleatoria, y se van actualizando en cada iteración de entrenamiento.

A fin de ir actualizando los pesos de la neurona se usa un algoritmo que, en base al resultado de la función de perdida para el valor entrenado, adapte los valores de los pesos. El algoritmo básico para lograr este objetivo es el de descenso por gradiente, un algoritmo de optimización iterativo usado para encontrar el mínimo o máximo local de una función, en este caso, de la función de perdida. Para que se pueda aplicar descenso por gradiente la función de perdida debe ser diferenciable (que tenga derivada para cada punto) y convexa.

El algoritmo de descenso por gradiente calcula de forma iterativa el próximo valor de los pesos (w_{n+1}) en base al gradiente de la función de perdida (∇f_L) y a un parámetro denominado tasa de aprendizaje o *learning rate (α)*, que determina el tamaño de los saltos para encontrar el mínimo local (ver ecuación (4)). Si este valor es muy pequeño se tardará mucho en llegar al mínimo o incluso puede no alcanzarse si no existen suficientes datos de entrenamiento. Por el contrario, si es muy grande puede no llegar a converger en el mínimo (ver **Figura 5**).

$$w_{n+1} = w_n - \alpha \nabla f_L(w_n) \tag{4}$$

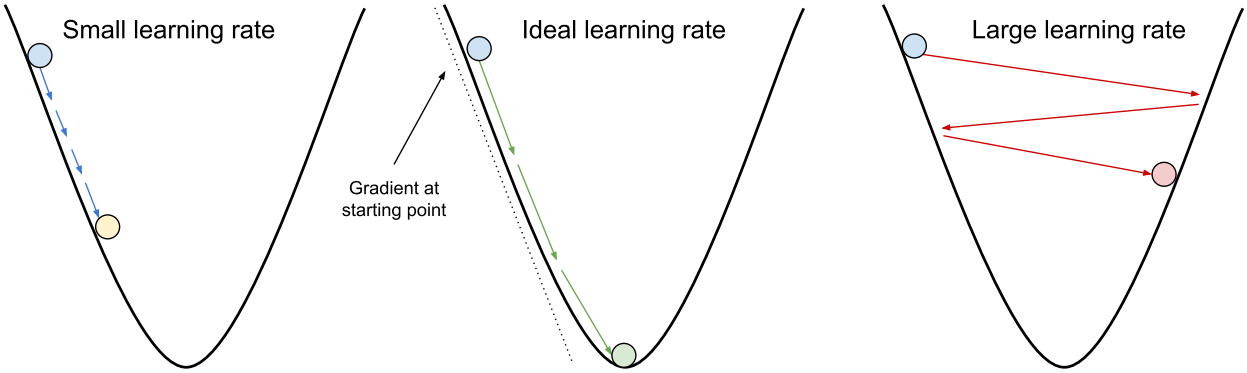

***Figura 5**: Ejemplificación de diferentes valores de learning rate en el algoritmo de descenso por gradiente. Fuente: [10].*

Una variación del algoritmo de descenso por gradiente es el descenso estocástico por gradiente (Stocastich Gradient Descent o SGD) especialmente diseñado para reducir las necesidades computacionales del original cuando se abordan problemas de optimizacion multidimensionales [11].

Mediante la unión de varios perceptrones se consigue la generación de redes neuronales. Estas se basan en la unión de perceptrones en capas o etapas, como se observa en la **Figura 6**, donde cada perceptrón tiene un conjunto de pesos propio y cada capa puede emplear una función de activación diferente. Según la complejidad del modelo, el mismo contará con más o menos capas.

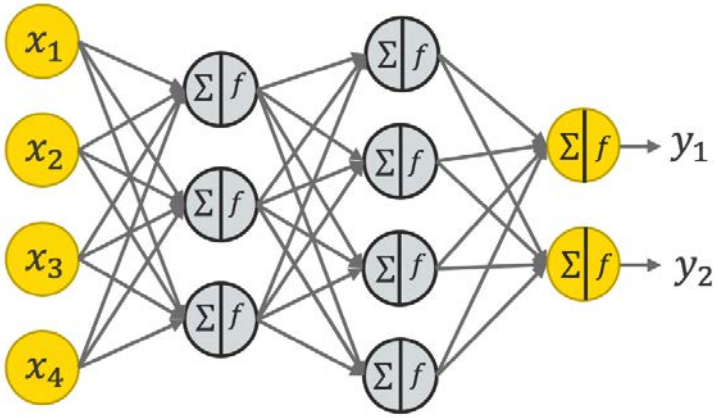

***Figura 6**: Ejemplo de red neuronal. Fuente: [12].*

Cuando se unen varias capas de neuronas, se encuentra el problema de que solo es posible medir el error en la capa de salida, por lo que solo se podrían actualizar los pesos de las neuronas de la última capa siguiendo el proceso descrito anteriormente. A fin de solventar este problema, se utilizan técnica de propagación hacia atrás de errores (*backpropagation*), de forma que las neuronas de las capas intermedias reciben una fracción del error total en base a la contribución de cada neurona a la generación de la salida [13].

2.3. Redes neuronales convolucionales

Las redes neuronales convolucionales (CNN, de las siglas en inglés) o simplemente redes convolucionales, son un tipo de red neuronal empleada para tratar datos con forma de cuadrícula, como, por ejemplo, las imágenes.

Estas redes, como su propio nombre indica, utilizan la operación de convolución para aplicar un filtro o *kernel* sobre una entrada y obtener como resultado una salida, generalmente denominada mapa de características. Si consideramos una entrada bidimensional I y un filtro K, la operación de convolución que se emplea se observa en la siguiente fórmula [13].

$$S(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{5}$$

Como ventaja frente a las redes neuronales convencionales (ver apartado 2.2) las redes convolucionales permiten reducir notablemente el número de pesos a entrenar en un modelo de *machine learning*. Si observamos el ejemplo de la **Figura 7**, se observa que para una cuadrícula de 5x5, lo cual supone 25 entradas, se usa solo un filtro de 3x3, es decir 9 valores. En el proceso de entrenamiento el modelo solo tendría que aprender estos nueve valores de pesos y no los 25 pesos que tendría que aprender si introdujéramos estas entradas en una red neuronal convencional. Por este motivo, suele ser el modelo elegido cuando se trabaja con imágenes, dado el elevado número de entradas, por ejemplo, en una imagen de 1024x768 píxeles.

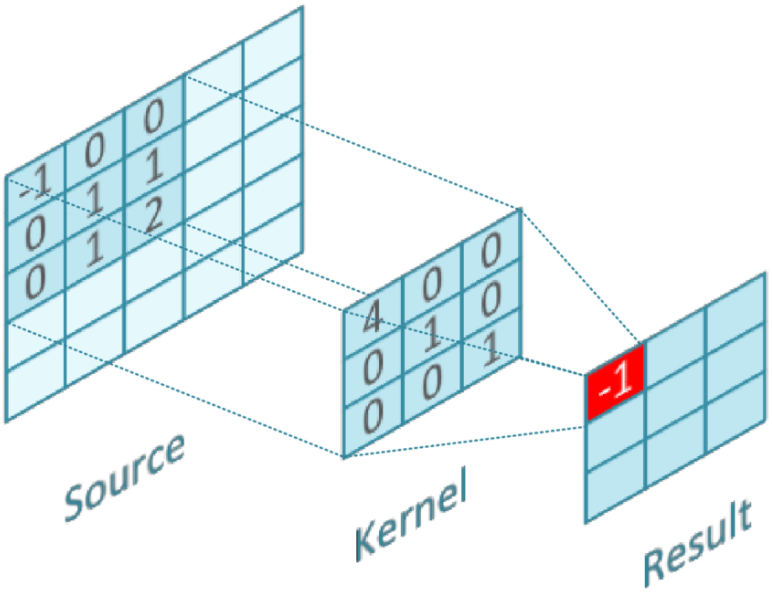

***Figura 7**: Ejemplo de proceso de convolución. Fuente: [14].*

El proceso de convolución se aplica sobre toda la imagen por lo que el tamaño de la salida se puede calcular como I - F + 1, donde I es el tamaño de la entrada y F el del filtro. Este valor se debe calcular en cada una de las dimensiones. En el ejemplo anterior, tenemos una entrada de dimensión 5x5 y un filtro de 3x3, por lo que el resultado tendrá una dimensión de 3x3 (5-3+1)x(5-3+1).

Con el objetivo de reducir el tamaño del resultado se puede introducir un valor de *stride*, es decir que la aplicación de la convolución no se aplique de forma consecutiva por toda la imagen, sino que, entre aplicación y aplicación, exista un salto o *stride*, tal como se observa en la **Figura 8**.

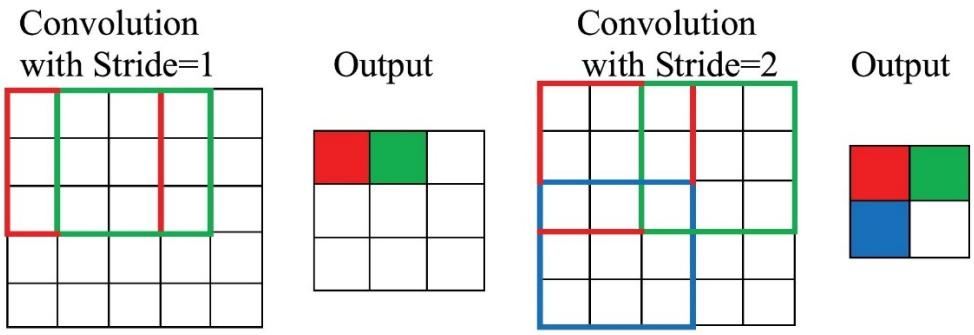

***Figura 8**: Ejemplo de convolución con diferentes valores de stride. Fuente: [15].*

El *stride* no solo permite reducir el tamaño de la salida, sino que con él también se consigue destacar las características más generales de la entrada. Otros de los problemas de este proceso es que los valores de los bordes de las imágenes son tenidos menos en cuenta que los centrales. Para corregir esto, y a fin de garantizar que las características de los bordes de la entrada pueden ser detectados, se suele incluir un reborde o *padding* a la entrada, tal como se observa en la **Figura 9**. El tamaño de la salida cuando se aplican valores de *stride* (S) y *padding* (P) es 1+(I-F+2P)/S.

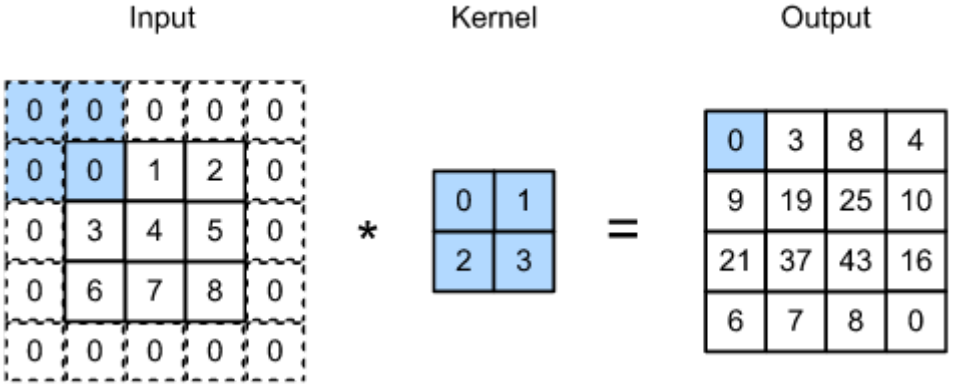

***Figura 9**: Ejemplo de convolución con padding de 1 en la entrada. Fuente: [16].*

En el caso de entrada de tres dimensiones o canales, por ejemplo, imágenes RGB, la convolución se realiza con filtros de la misma dimensión, sumándose igualmente los resultados de cada posición, con lo cual se consigue reducir la dimensión en la salida a dos dimensiones. A fin de aumentar la dimensión de la salida, se pueden usar convoluciones en paralelo como se observa en la **Figura 10**.

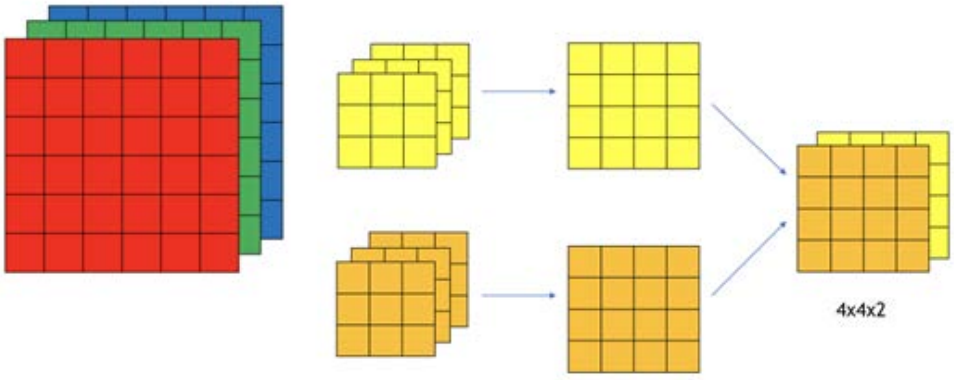

***Figura 10**: Ejemplo de convolución sobre volumen. Fuente: [17].*

Las redes convolucionales se forman mediante la unión de diferentes capas, principalmente:

- Capas **convolucionales**: en estas capas se realiza la operación de convolución tal como se ha visto. Dado que la convolución es una operación lineal, a fin de que cobre sentido la unión de varias capas, se suele añadir a las capas convolucionales una función de activación no lineal, generalmente la función ReLU [18], que se aplica a cada valor de la matriz del resultado de la convolución.
- Capas de ***pooling***: estas capas se utilizan para reducir la dimensión de la salida de la capa anterior. Generalmente se reduce cada cuadrante (por ejemplo, de 2x2) a su valor medio (*average pooling*) o máximo (*max pooling*) (**Figura 11**).

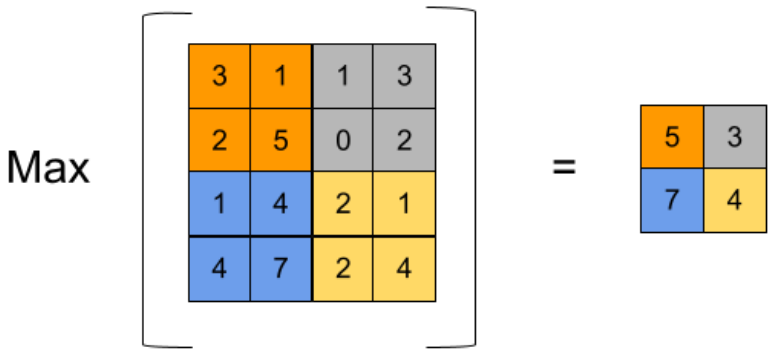

***Figura 11**: Ejemplo de operación de max. pooling. Fuente: [19].*

- Capas ***fully-connected***: En estas capas, que suelen estar al final de las redes convolucionales, se reduce la dimensión de la salida de la capa anterior a una dimensión, un vector, y se conectan cada uno de los valores a cada una de las neuronas de una capa de redes neuronales convencionales.
- **Softmax**: suele ser la última capa de estos modelos y permite asociar finalmente una etiqueta de predicción al valor de entrada [20], tal como se observa en la **Figura 12**.

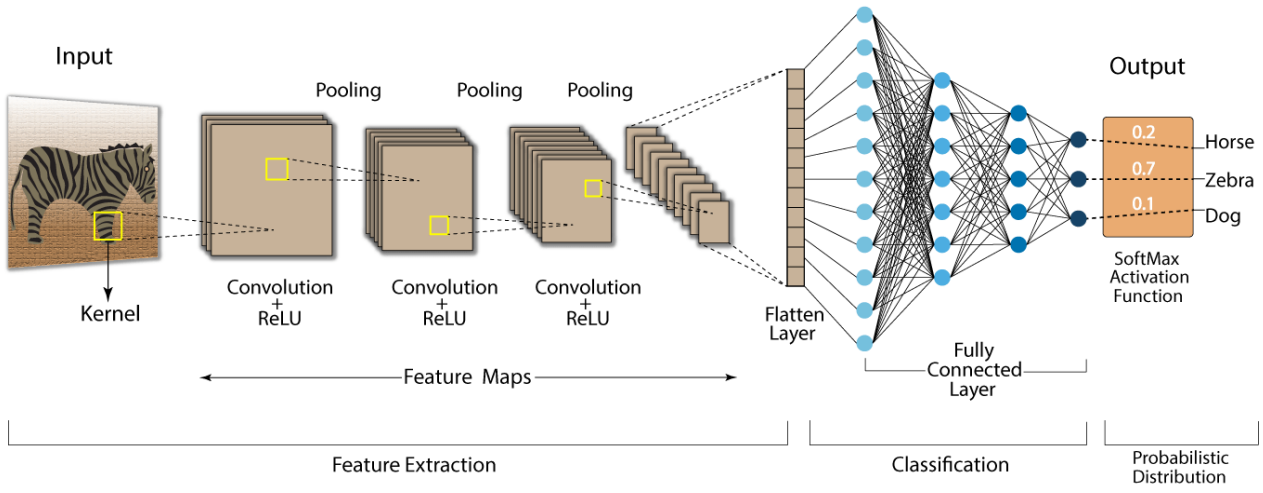

Figura 12: Ejemplo de arquitectura de red convolucional. Fuente: [21].

2.3.1. Arquitectura de redes convolucionales

En este apartado se explicarán dos modelos de redes convolucionales ampliamente usados en diferentes desarrollos en los últimos años y que también se implementará en el desarrollo actual, por lo que el conocimiento de su funcionamiento permitirá entender mejor el del modelo que se describirá más adelante.

VGG16

El modelo VGG16, descrito por Simonyan y Zisserman en 2015 [22], surge como forma de simplificar las redes convolucionales desarrolladas hasta ese momento. Para ello, adopta un modelo más sencillo, que está compuesto por 13 capas convolucionales en 5 bloques, seguidos cada uno por una capa de *max pooling*; y 3 capas *fully connected*, terminando con una capa *softmax*. Cada capa convolucional usa filtros de 3x3 con *stride* de 1 y *padding* de 1 píxel. Las capas de *max pooling* utilizan un filtro de 2x2 y un *stride* de 2. Todas las capas convolucionales y *fully conected* emplean la función de activación ReLU.

El número de canales se duplica en cada uno de los cuatro primeros bloques convolucionales, pasando de 64 a 512, tal como se observa en la **Figura 13**. En total, la red tiene alrededor de 138 millones de parámetros.

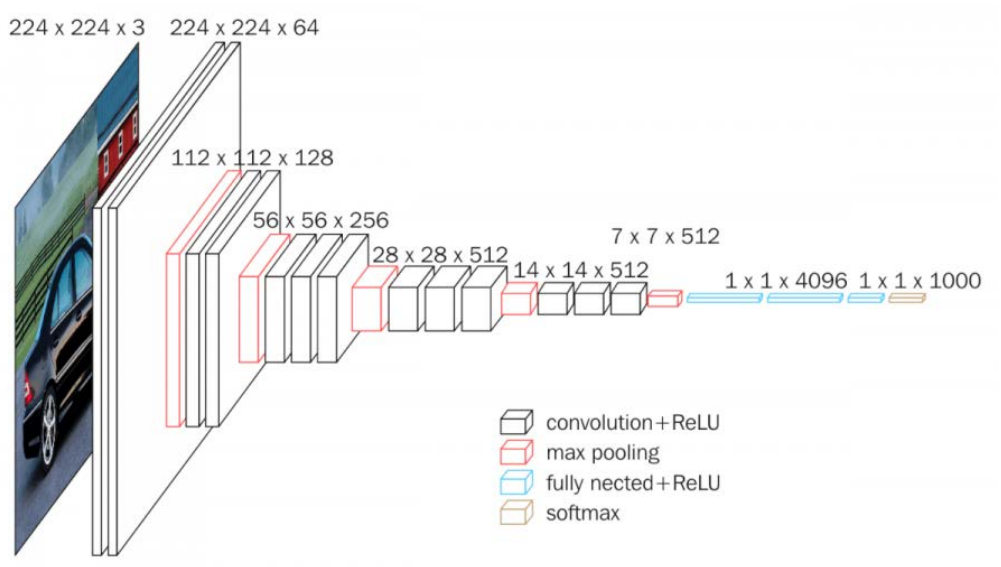

***Figura 13**: Arquitectura de la red VGG16. Fuente: [23].*

Una de las ventajas del uso de este modelo, es que existe la posibilidad de acceder a una versión del mismo pre-entrenada con el conjunto de datos de ImageNet, que contiene más de 14 millones de imágenes agrupadas en más de 20.000 categorías [24].

U-Net

Los modelos de arquitectura vistos anteriormente son de gran utilidad para tareas de clasificación, por ejemplo, identificar si una fotografía se corresponde con una imagen de un perro o un coche. Sin embargo, existen aplicaciones en las que la mera clasificación no es suficiente y se necesita también asociar una etiqueta a cada píxel y no solo a la imagen en su conjunto. Esta tarea se conoce como segmentación semántica y es de gran relevancia, por ejemplo, en el campo de la medicina, para el diagnóstico mediante imágenes, donde es necesario conocer la ubicación de la lesión, como en el ejemplo de la **Figura 14**.

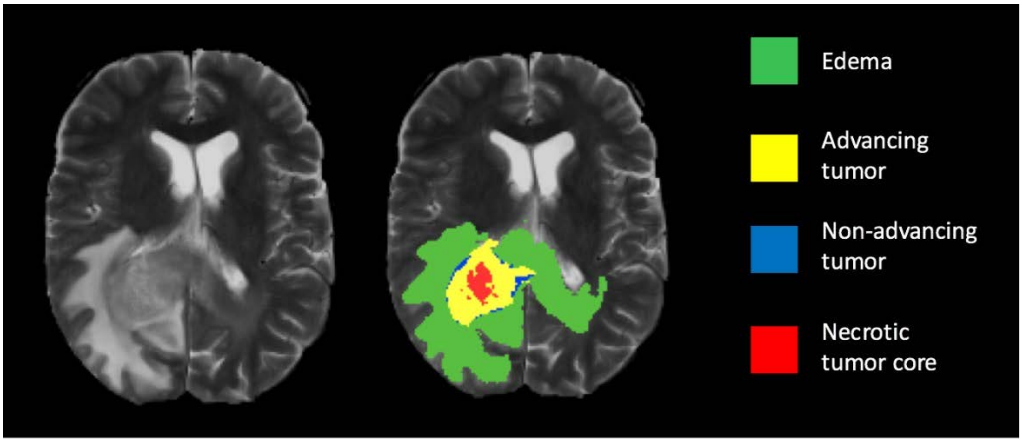

***Figura 14**: Ejemplo de segmentación semántica aplicada al diagnóstico de tumores cerebrales. Fuente: [25].*

A fin de lograr este objetivo de asignar una clasificación a cada píxel, U-Net, descrito por Ronneberger, Fischer y Brox en 2015 [26], implementa un modelo en el que, primero se reduce la resolución de la imagen mediante convoluciones y *pooling* para extraer las características de la misma (proceso de codificación), y luego se vuelve a aumentar la resolución a fin de obtener un mapa de características de resolución similar a la de la imagen de entrada, donde se puedan asignar etiquetas a cada píxel (proceso de decodificación). La arquitectura de esta red prescinde de las capas *fully connected* y *softmax*, siendo el resultado de la red, similar a una imagen y no una etiqueta.

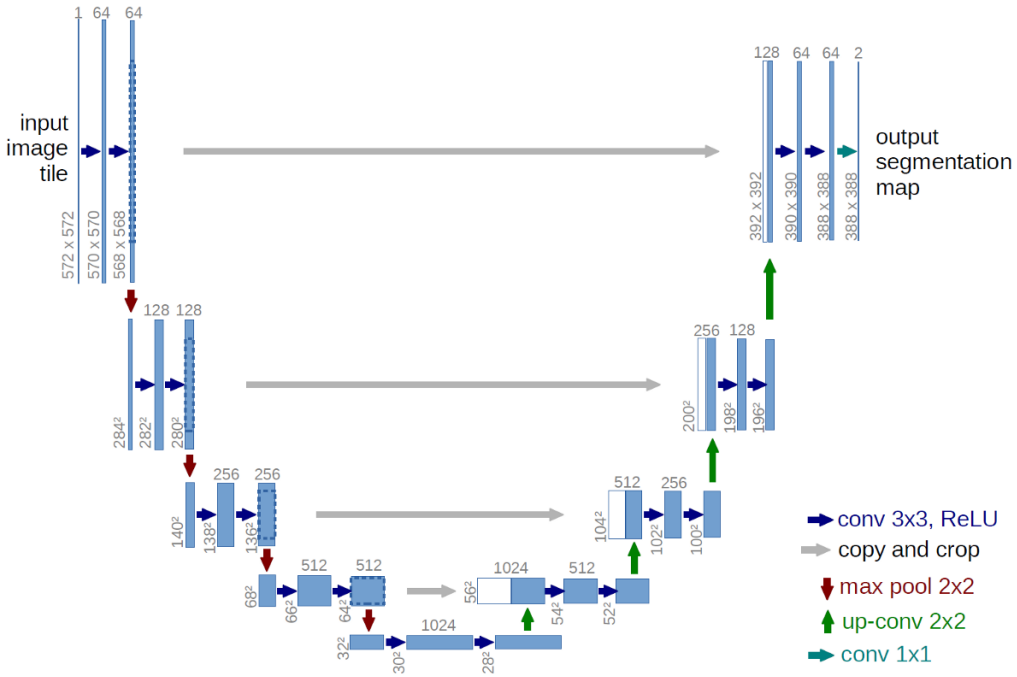

***Figura 15**: Arquitectura de la red U-Net. Fuente: [26].*

La arquitectura de la parte de codificación es muy similar a la expuesta en el modelo VGG16, con cinco bloques de capas convolucionales con filtros de 3x3, sin *padding*, y seguidos de una ReLU. A cada bloque le sucede un *max pooling* de 2x2 con *stride* de 2. En cada bloque se duplica el número de canales, pasando de 64, en el primero, a 1024 en el último.

La parte de decodificación consisten en cuatro bloques, cada uno se inicia con la aplicación de una operación de convolución transpuesta (proceso inverso al de la convolución que permite incrementar

el tamaño de la entrada, su explicación matemática puede verse en [27]), con un filtro de 2x2, sobre la salida del bloque anterior, operación en la que, además, se reduce a la mitad el número de canales. Al resultado de esta operación se le añade el mapa de características generado al final del bloque equivalente de la parte de codificación, recortado para igualar las dimensiones (la asociación de bloques de codificación-decodificación está indicado en la **Figura 15** con flechas de color gris). Sobre esta unión se aplican dos convoluciones de 3x3, seguidas cada una por una ReLU.

Por último, se realiza una convolución con un filtro de 1x1 y un número de canales igual al número de clases que se quieren reconocer o etiquetar. La composición final se genera mediante la asignación a cada píxel de la etiqueta correspondiente al canal con mayor valor para ese píxel.

2.4. Aprendizaje máquina en el conteo de objetos en imágenes

Para el conteo de objetos en imágenes se han utilizado generalmente tres aproximaciones diferentes: conteo por detección, conteo por regresión y conteo por estimación de densidad [28].

El conteo por detección se basa en la localización de cada objeto de la imagen en base a patrones de características de los objetos. Estos métodos han mostrado buenos resultados en conjuntos de datos donde los objetos están separados unos de otros, sin embargo, en escenas donde los objetos están muy juntos y existe solapamiento entre ellos, los resultados no han sido tan buenos. Algunas de las últimas propuestas en este campo, que utilizan características locales en vez las globales de la imagen, han mejorado los resultados en el conteo en imágenes con alta densidad de objetos [29].

Por su parte, los modelos basados en regresión intentan identificar una relación entre las características de la imagen y el número de objetos usando técnicas de aprendizaje máquina supervisado. Estos modelos no utilizan conjuntos de datos basados en la ubicación de cada objeto, sino que simplemente necesitan el número total de objetos en la imagen. Por ello, si bien los resultados de estos modelos suelen ser mejores que los basados en la detección, suelen necesitar grandes conjuntos de datos para el entrenamiento [28].

Finalmente, frente a los modelos de basados en detección de características o en regresión, que ignoran la información espacial de las imágenes sobre las que se realizaba el conteo de objetos, en 2010 se comenzó a incluir esta información en las soluciones propuestas [30] adoptando un mapeo de las características en las imágenes y sus correspondientes mapas de densidad, lo cual mejoró los resultados en la precisión de las mediciones comparado con aproximaciones anteriores [31].

Los modelos de CNN basados en mapas de densidad (**Figura 16**) reciben como entrada una imagen y en vez de generar directamente una estimación del número de objetos en la misma, generan un mapa de densidad del que luego, mediante integración, se obtienen la estimación del número de objetos.

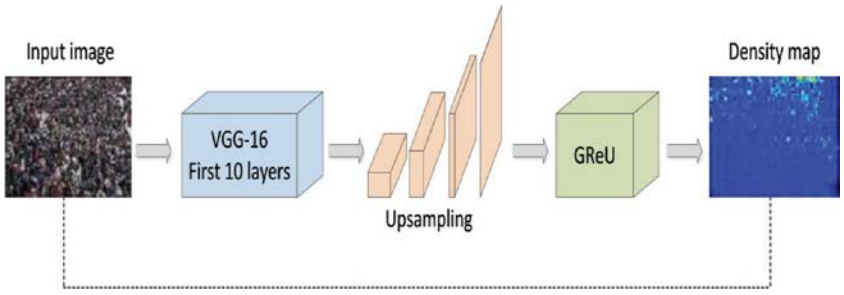

***Figura 16**: Ejemplo de modelo de CNN para estimación de densidad. Fuente: [32].*

Algunos de los beneficios de esta aproximación son, por un lado, que los mapas de densidad mantienen más información sobre la distribución de los objetos en la imagen, y por otro, los filtros aprendidos mediante mapas de densidad tienen mayor capacidad de adaptación para objetos de diferentes tamaños, por lo tanto, toleran mejor las imágenes diferentes, por ejemplo, con cambios de perspectiva, lo cual mejora la precisión de los modelos [33].

Además, la generación de mapas de densidad también ayuda a compensar las diferencias existentes en el etiquetado de las imágenes, generalmente realizado como el punto medio de cada objeto, el cual puede variar, por ejemplo, varias personas etiquetando cabezas pueden elegir definir el centro en lugares diferentes (frente, nariz, etc.).

La generación de los mapas de densidad (**Figura 17**) suele ser llevada a cabo mediante el difuminado de los puntos etiquetados, generalmente empleando para ello un *kernel* gaussiano.

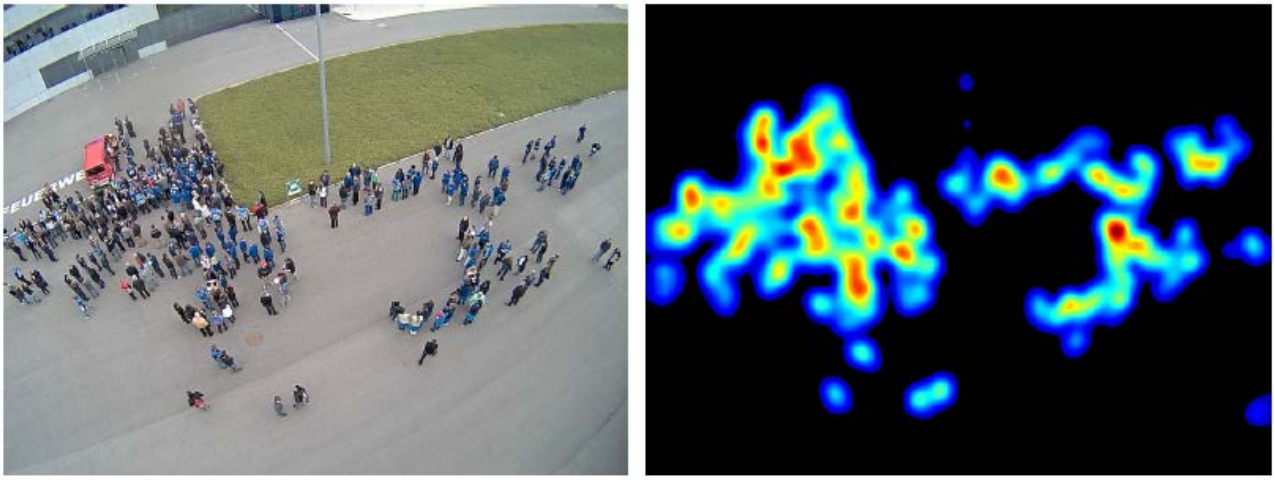

***Figura 17**: Ejemplo de mapas de densidad generados a partir de imágenes etiquetadas. Fuente: [28].*

3. Especificaciones y restricciones de diseño

El objetivo del proyecto es la implementación un sistema de conteo de crías de rodaballos en tanques a partir de imágenes de los mismos. Para el desarrollo del mismo se definen las siguientes restricciones de diseño:

- La entrada del sistema serán imágenes RGB, si bien no se establecen limitaciones del tamaño de las imágenes, esta podrá limitarse mediante escalado de las originales a fin de mejorar el rendimiento del sistema.
- Se empleará un modelo de redes convolucionales que apliquen técnicas de estimación de densidad.
- Para el entrenamiento del modelo se usará el conjunto de datos elaborado por el Grupo de Aplicaciones Multimedia y Acústica.
- El sistema debe tener en cuenta y ser capaz de contar objetos agrupados y superpuestos, siempre que sean diferenciables.
- El sistema se desarrollará en Python 3 y se emplearán únicamente librerías de código abierto. Sin embargo, se podrán utilizar otros programas para el tratamiento del conjunto de datos o de las métricas generadas por el sistema.
- La interfaz de la aplicación será en modo consola y, a través de la misma, se debe poder entrenar el modelo y obtener datos sobre la eficacia del mismo.

4. Descripción de la solución propuesta

El presente apartado recoge todo el proceso de desarrollo de la solución, desde la elección del algoritmo y su implementación, hasta la creación y análisis del conjunto de datos, pasando por el análisis de los parámetros para la optimización del modelo.

4.1. Elección del modelo

Para la elección del modelo de redes convolucionales a implementar en el actual desarrollo, se analizaron los modelos de conteo de objetos en imágenes que mejores resultados habían conseguido sobre algunos de los conjuntos de datos más extendidos en el ámbito de conteo de personas.

Si bien, como se expone en la introducción, ya existen a día de hoy algunas propuestas que aplican el aprendizaje máquina sobre imágenes para estimar el número de peces presente, entre otros [34] [35] [36] [37] [38], todos los que se basan en redes neuronales siguen aproximaciones parecidas a las del ámbito de conteo de personas por lo que se ha decidido trabajar sobre estos últimos ya que cuentan con mayor desarrollo y existen conjuntos de datos comunes y públicos que permiten comparar la eficacia de los modelos, lo cual no ocurre en el ámbito de la acuicultura, al no existir conjuntos de datos públicos.

Entre los conjuntos de datos de imágenes de personas más utilizados como referencia para comparar los distintos modelos de conteo, se encuentran:

- **ShangaiTech**: Consta de 1198 imágenes anotadas de muchedumbres. El conjunto de datos se divide en dos partes: la Parte A, que contiene 482 imágenes, y la Parte B, que contiene 716 imágenes. La parte A se divide en subconjuntos de entrenamiento y de prueba, que constan de 300 y 182 imágenes, respectivamente. La parte B se divide en subconjuntos de entrenamiento y de prueba, que constan de 400 y 316 imágenes. Cada persona en una imagen de la muchedumbre se anota con un punto cerca del centro de la cabeza. En total, el conjunto de datos consta de 330.165 personas anotadas. Las imágenes de la parte A se recogieron de Internet, mientras que las de la parte B se recogieron en las calles más concurridas de Shanghai [33].
- **UCF CC 50**: consiste en imágenes de multitudes con mucha densidad. Tiene 50 imágenes con 63.974 anotaciones de centros de cabeza en total. El número de cabezas oscila entre 94 y 4.543 por imagen [39].
- **UCF QNRF**: contiene una gran variedad tanto de escenas como de tipos de fondo. Consta de 1.535 imágenes de alta resolución procedentes de Internet, incluyendo imágenes del Hajj (peregrinación a la Meca). El número de personas etiquetadas varía de 50 a 12.000 entre las imágenes [31].

En base a los diferentes análisis del estado del arte de modelos de CNN para la estimación por densidad de número de personas en imágenes [28] [29] [40] y atendiendo a las especificaciones de diseño y la eficacia de los modelos, se ha seleccionado el modelo SS-DCNet (*Supervised Spatial Divide-and-Conquer for Object Counting*) [41], ya que presenta las siguientes características:

- Sus resultados para los conjuntos de datos mencionados se encuentran entre los que menores errores arrojan [41] [40].
- Los autores han demostrado la eficacia del modelo no solo para el conteo de personas, sino también, para el conteo de vehículos y de mazorcas de maíz. Esta generalización es muy relevante de cara a su adaptabilidad al conjunto de datos del presente proyecto.

- Uno de los objetivos del modelo es que el mismo sea capaz de contar objetos incluso cuando la imagen tenga un número mayor de objetos que las que se usaron para entrenarlo. Este aspecto también es de interés para el actual desarrollo, ya que, no solo los tanques de cría de rodaballos pueden variar en tamaño y densidad, sino que, además, el conjunto de datos de entrenamiento a emplear se corresponde con imágenes parciales de los tanques. Por ello, este modelo permitiría, partiendo de este conjunto de datos, contar rodaballos en imágenes del tanque completo.

4.1.1. SS-DCNET

El modelo SS-DCNet (*Supervised Spatial Divide-and-Conquer for Object Counting*) [41] implementado en el desarrollo actual consta de un sistema de codificador-decodificador, como el visto en el apartado 2.3.1. Para el proceso de codificación se usa una red VGG16 sin las capas *fully connected* ni *softmax* y pre-entrenada con el conjunto de datos Imagenet. Para la decodificación, se utiliza un modelo de decodificación basado en la red U-Net (ver apartado 2.3.1).

Sin embargo, en vez de utilizar regresión para obtener el número de objetos a partir del mapa de densidad, utiliza un clasificador cuyas clases son valores discretos en un rango bajo $[0 \sim 20]$. A fin de poder implementar este sistema, el modelo sigue una estrategia de división espacial (**Figura 18**), dividiendo la imagen en subimágenes más pequeñas hasta conseguir que el número de objetos en cada una esté comprendido en el rango del clasificador.

***Figura 18**: Ilustración de la estrategia de división espacial empleada por SS-DCNet. Fuente: [41] .*

El objetivo de la estrategia seguida por SS-DCNet es conseguir un modelo que pueda generalizar el conteo de imágenes con mayor número de objetos que los que figuraban en el conjunto de datos de entrenamiento, lo cual supone un reto para los modelos de conteo por regresión.

El funcionamiento de SS-DCNet se muestra en la **Figura 19**. Las imágenes se dividen en parches de 64x64 píxeles y se codifican con los cinco bloques de VGG16, resultando en un mapa de características de 2x2, denominado F_0. Este bloque se decodifica siguiendo el modelo U-Net generando los mapas F_1 y F_2. En teoría, se podrían decodificar otros dos mapas de características, ya que el proceso de codificación tiene cuatro bloques, sin embargo, los desarrolladores del modelo han demostrado que, con una selección adecuada de la dimensión del clasificador, dos decodificaciones son suficientes.

Las clases del clasificador se definen con un paso o *step* de 0.5 resultando en las clases: {0}, (0, 0.5], (0.5, 1], ..., (C_{Max} − 0.5, C_{Max}] y (C_{Max}, +∞), con C_{Max} siendo el valor máximo del clasificador por lo que a cualquier parche de la imagen que supere este número de objetos se le asignará la última etiqueta y dicho parche será divido espacialmente para evaluar sus subregiones. La elección del valor de C_{Max} se aborda en el apartado 4.4.2. Además, para el intervalo (0, 0.5] se adopta una segunda partición (estrategia denominada en [41] *two linear partition*) con un step de 0.05, a fin de afinar el conteo de objetos parciales.

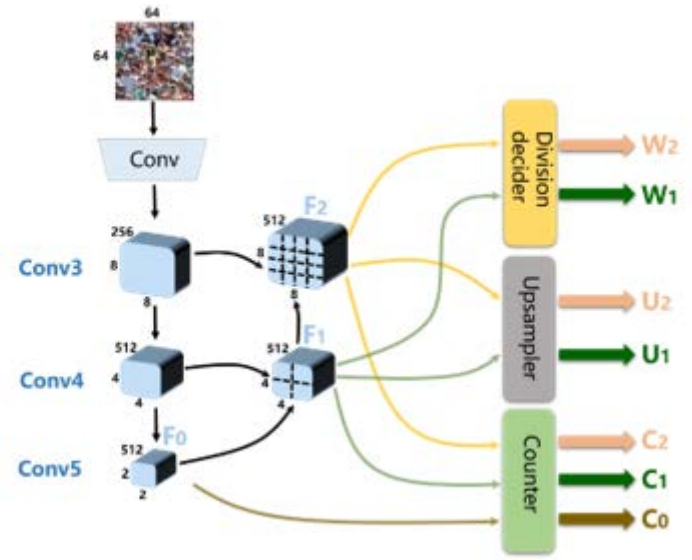

***Figura 19**: Arquitectura del modelo SS-DCNet. Los dos primeros bloques de VGG16 se han simplificado con el bloque 'Conv'. Fuente: [41].*

El modelo implementa, además, tres bloques convolucionales para la decisión sobre la división espacial (Division decider), el sobremuestreo del contador (Upsampler) y la función de contador (Counter). La composición de estos bloques es la siguiente:

- **Counter**: una capa *avgpooling* (*pooling* con el valor medio) de 2x2 con un *stride* de 2, seguida por una convolución con filtro de 1x1, *stride* de 1 y 512 canales de salida y finalmente una capa de convolución de 1x1, con *stride* de 1, y número de canales de salida igual al número de clases del clasificador.
- **Division decider**: una capa *avgpooling* de 2x2 con un *stride* de 2, seguida por una convolución con filtro de 1x1, *stride* de 1 y 512 canales de salida y finalmente una capa de convolución de 1x1, con *stride* de 1, y un solo canal a la que se le aplica una función de rectificación sigmoide.
- **Upsampler**: una capa *avgpooling* de 2x2 con un *stride* de 2, seguida por una convolución con filtro de 1x1, *stride* de 1 y 512 canales de salida y finalmente una capa de convolución de 1x1, con *stride* de 1, y un solo canal a la que se le aplica una función de rectificación Spatial Softmax (ver descripción de la función en [42]).

El resultado de pasar F_0 a través del módulo contador, C_0 es el valor del número de objetos totales del parche de 64x64 (dentro de las clases del clasificador), lo cual sería el resultado final de modelos desarrollados con anterioridad.

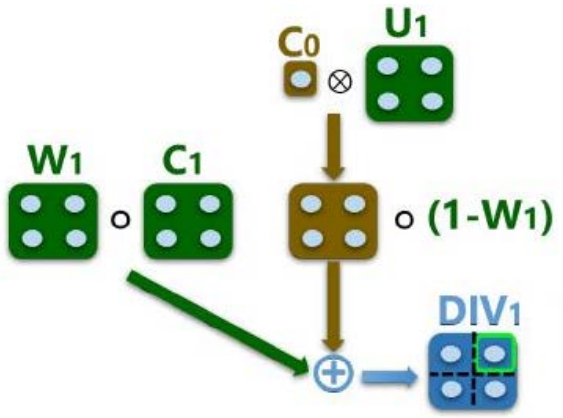

***Figura 20**: Primer proceso de división espacial de SS-DCNet. Fuente: [41].*

Sin embargo, SS-DCNet realiza un primer proceso de división (**Figura 20**) a fin de comprobar si el valor de C_0 es óptimo o es mejor realizar la cuenta sobre parches más pequeños de 32x32, lo cual sería el resultado de pasar el mapa de característica F_1 por el módulo contador, generando una matriz de 2x2 con el número de objetos para cada parche de 32x32. El proceso de división se realiza a partir de la matriz de 2x2, W_1, generada por el módulo decisor de división a partir de F_1. Esta matriz indica, para cada cuadrante, si es 1, que se debe usar el valor de C_1, y si es 0, que se puede usar el valor proporcional para ese cuadrante de C_0. Tras este proceso, se genera una matriz DIV_1, de 2x2, con el número de objetos por cuadro de 32x32 (ver ecuación (7)). Por último, a fin de dividir C_0 proporcionalmente por cada cuadro de 32x32, el módulo Upsampler genera una matriz, U_1, de 2x2 cuya suma total da 1 y que indica la distribución proporcional de C_0, generándose una nueva matriz $\hat{C}_0$ (ver ecuación (6)).

$$\hat{C}_0 = (C_0 \otimes 1_{2x2}) \circ U_1 \tag{6}$$

$$DIV_1 = (1 - W_1) \circ \hat{C}_0 + W_1 \circ C_1 \tag{7}$$

Este proceso de división espacial se realiza una segunda vez, tal como se indica en la **Figura 21** y pudiéndose generalizar la fórmula como se muestra en (8).

$$DIV_i = (1 - W_i) \circ \widehat{DIV}_{i-1} + W_i \circ C_i \tag{8}$$

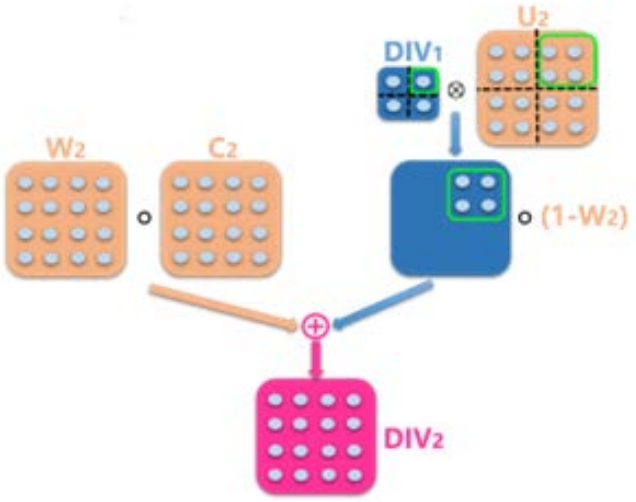

***Figura 21**: Segundo proceso de división espacial de SS-DCNet. Fuente: [41].*

Por último, el modelo implementa cuatro funciones de pérdida (*loss functions*) diferentes:

- Pérdida del contador (***counter loss***): mediante una función de entropía cruzada (*cross entropy loss*) [43], mide el error del módulo contador como el sumatorio del error parcial de cada una de las salidas C_0, …, C_2.
- Pérdida pro fusión (***merging loss***): mediante una función de error medio absoluto (MAE), mide el error en las salidas DIV_N. Esta señal permite supervisar de forma implícita el error sobre las salidas de decisor de división W_N.
- Pérdida por división (***division loss***): esta función de perdida permite supervisar directamente el valor de salida del decisor de división. La medición se basa en que, cuando el total de objetos etiquetados, C_0^{gt}, en el parche de 64x64 es mayor que el de C_{Max}, al menos un valor de la matriz W_1 debe ser 1, ya que hay al menos un cuadrante donde el número de objetos es mayor que el que la redistribución de C_0 recoge. El error se calcula para W_1 según (11) y de forma análoga para W_2. Donde $1\{C_0^{gt} > C_{Max}\}$ es igual a 1 si el valor C_0^{gt} es mayor que C_{Max} e igual a 0 en caso contrario.

$$L_{DIV}^1 = -1\{C_0^{gt} > C_{Max}\} \times log\big(max(W_1)\big) \tag{9}$$

- Pérdida de sobremuestreo (***upsampling loss***): se utiliza para supervisar el módulo Upsampler y se calcula como el error medio absoluto (MAE) entre los valores de la matriz generada por el módulo de $U_i y$ la matriz U_i^{gt} calculada como se muestra en (10).

$$U_i^{gt} = C_i^{gt} / (C_{i-1}^{gt} \otimes 1_{2x2}) \tag{10}$$

4.2. Arquitectura

Si bien los autores del artículo del modelo SS-DCNet [41] han publicado un código fuente para evaluar la precisión de su modelo, esta implementación solo cuenta con capacidad para evaluar un modelo ya entrenado y no cuenta con rutinas para entrenar desde cero un modelo con un conjunto de datos específico. Por este motivo, el código fuente de base que se empleará en este proyecto es el publicado en la plataforma Github, como código libre, por Dmitry Burdeiny [44].

A fin de evaluar la precisión de la implementación del modelo SS-DCNET en el código de Dmitry Burdeiny, en la **Tabla 1** se puede observar los resultados del error (MAE y MSE) para el modelo SS-DCNet original descrito en el artículo y los obtenidos usando el código de Dmitry Burdeiny para las dos partes del conjunto de datos ShangaiTech [33].

***Tabla 1**: Errores en la implementación del modelo SS-DCNet sobre el conjunto de datos ShangaiTech.*

		Parte A		Parte B	
		MAE	MSE	MAE	MSE
SS-DCNet	Artículo original [41]	56.1	88.9	6.6	10.8
	Código de Dmitry Burdeiny [44]	61.16	105.23	8.22	13.65

El desarrollo, escrito en Python 3, cuenta con cuatro módulos ejecutables de forma independiente para realizar las funciones de generación de mapas de densidad a partir de las imágenes etiquetadas, entrenamiento y evaluación del modelo e inferencia de nuevas imágenes. Además, el código cuenta con archivos de configuración y cuatro librerías comunes.

Módulos:

- **Generación de mapas de densidad**: permite generar mapas de densidad partiendo de un conjunto de datos en el formato especificado en el apartado 4.3.
- **Entrenamiento**: entrena al modelo SS-DCNet con el conjunto de datos de entrenamiento y los mapas de densidad y genera un archivo con el modelo, así como métricas sobre el proceso de entrenamiento.
- **Evaluación**: evalúa el error del modelo tanto para el conjunto de datos de test como para el de entrenamiento.
- **Inferencia**: permite, a partir del modelo generado en el módulo de entrenamiento, inferir el número de objetos en nuevas imágenes.

Librerías communes:

- **SDCNet**: recoge la implementación del modelo de red convolucional y clasificador SS-DCNet.
- **Loss**: funciones de error para el entrenamiento del modelo.
- **Dataset**: clase para la creación de la estructura del conjunto de datos para su uso en los módulos de entrenamiento y validación.
- **Label_count_utils**: para la generación de las etiquetas del módulo clasificador.

El código ha sido adaptado para cumplir con las especificaciones de diseño y para hacerlo compatible con el actual conjunto de datos. El manual de usuario del código con las modificaciones se encuentra en el **Anexo 2**.

4.2.1. Entorno de desarrollo y entrenamiento

Es este apartado se describen las principales herramientas utilizadas para el desarrollo del proyecto, desde el lenguaje de programación y principales librerías, hasta los servicios de computación en la nube, pasando por el entorno de desarrollo. Muchas de estas herramientas y servicios han sido elegidos por existir conocimiento previo de su uso o por contar con un convenio con la Universidad Politécnica de Madrid que eliminaba los costes de su adquisición o uso. Sin embargo, muchas de las herramientas y servicios cuentan con equivalentes en el mercado que permiten obtener funcionalidades similares.

Python 3

El lenguaje de programación principal del desarrollo es Python en su versión 3.8. Python es un lenguaje de alto nivel interpretado, lo que significa que se pueden ejecutar directamente las instrucciones, sin necesidad de compilar previamente el programa y traducirlo a instrucciones de lenguaje máquina. El intérprete de Python se distribuye bajo licencia de código abierto por la Python Software Fundación y puede ser descargado en su página web para plataformas Windows, macOS o Linux/UNIX, entre otras.

PyTorch

PyTorch es la librería de aprendizaje máquina que se utilizará para implementar el modelo de redes convolucionales. Es una biblioteca de código abierto desarrollada por el Laboratorio de Investigación de Inteligencia Artificial de Facebook (FAIR). Esta librería es ampliamente utilizada para aplicaciones de visión artificial y procesamiento de lenguajes naturales. Además, permite desplegar fácilmente modelos, incluso pre-entrenados, para tareas de clasificación de imágenes, segmentación semántica por píxeles, detección de objetos o clasificación de vídeos, entre otras. En este sentido, se incluye el modelo VGG16 expuesto en el apartado 2.3.1.

CUDA

Con el objetivo de mejorar la velocidad de entrenamiento del modelo de *machine learning*, se utilizará las ventajas del modelo de programación y plataforma de cálculo en paralelo CUDA, desarrollado por NVIDIA para el cálculo en unidades de procesamiento gráfico (GPU). CUDA permite aprovechar las capacidades de algunas GPU del fabricante NVIDIA para acelerar las aplicaciones de cálculo aprovechando la potencia de las GPU. CUDA está disponible de forma gratuita para plataformas Windows y Linux.

Pycharm

Como entorno de desarrollo se utilizará el software multiplataforma Pycharm, de la empresa JetBrains. El mismo cuenta con funciones de depuración y asistencia y análisis en la codificación. Además, permite gestionar las dependencias y librerías del intérprete de Python. Existe una versión profesional de pago y una edición comunitaria gratuita que se publica bajo la licencia apache. La versión profesional se ofrece sin coste para estudiantes.

MATLAB

Para el tratamiento y análisis del conjunto de datos se utilizará la plataforma de programación y cálculo numérico MATLAB. Además, se usará el complemento de Estadística y *Machine Learning* para el análisis estadístico del conjunto de datos. Este software es comercializado por la empresa MathWorks y ofrece un amplio rango de licencias, incluida la de estudiantes. Además, la Universidad Politécnica de Madrid tiene una licencia que permite el uso sin coste para sus alumnos.

Azure Cloud

El entrenamiento de modelos de aprendizaje máquina exige gran demanda de recursos de computación, incluyendo unidades de procesamiento gráfico (GPU), que además cuenten con gran cantidad de memoria.

Con el objetivo de agilizar las fases de prueba y entrenamiento del modelo implementado, se utilizarán los servicios de Azure Cloud, en concreto su plataforma Azure Machine Learning, que permite crear, implementar y administrar modelos de *machine learning*.

Sin embargo, dado que ya se cuenta con un modelo propio en el presente desarrollo, se utilizará la plataforma con el código de esta implementación.

La citada plataforma permite la creación de máquinas virtuales y el acceso a las mismas mediante consola. Estas máquinas tienen recursos adaptados para el despliegue de modelos de *machine learning* y ya tienen preinstalado CUDA y Python, así como las principales librerías necesarias.

El uso de esta plataforma se realiza en el marco del convenio que existe entre la empresa Microsoft y la Universidad Politécnica de Madrid, por lo que no supone ningún coste. En concreto se utilizará una máquina virtual con 6 núcleos, 56 GB de memoria RAM, 380 GB de disco duro y una GPU NVIDIA Tesla K80 (con 24 GB de memoria GDDR5 y 4992 núcleos de NVIDIA CUDA con diseño de dos GPU).

4.2.2. Generación de mapas de densidad

Para la generación de mapas de densidad se utilizará el procedimiento descrito en [30]. Para ello se asume que se parte de un conjunto de N imágenes $I_1, I_2, \ldots, I_N$. La imagen está compuesta por píxeles que se denominan p y corresponden a una función bidimensional, $p=(x,y)$, siendo x la localización del píxel en la imagen en la dimensión de abscisas e y la localización del pixel en la imagen en la dimensión de ordenadas. Cada imagen I_i está etiquetada con un conjunto 2D de puntos que corresponden a la posición de los objetos de interés $P_i = \{P_1, P_2, \ldots, P_{C(i)}\}$, siendo $C(i)$ el número total de objetos etiquetados en la imagen I_i.

Para cada pixel p de la imagen se define una función de densidad $F(p)$ tal que:

$$\forall p \in I_i, \qquad F_i^0(p) = \sum_{P \in P_i} \mathcal{N}(p, P;\ \sigma) \tag{11}$$

Donde $\mathcal{N}(p; P, \sigma^2 1_{2x2})$ denota un *kernel* gaussiano 2D normalizado, cuya función se define como:

$$\mathcal{N}(p, P;\ \sigma) = \frac{1}{2\pi\sigma^2}\, e^{-\frac{(p_x - P_x)^2 + (p_y - P_y)^2}{2\sigma^2}} \tag{12}$$

Teniendo en cuenta que tanto p como P contienen valores para las coordenadas x e y. La variable σ es la desviación estándar de la distribución y marca el grado de difuminado o suavizado (ver ejemplo en **Figura 22**). En principio, la función del *kernel* gaussiano afecta, para cada punto P, a todos los demás píxeles de la imagen, si bien cuando más alejado, la afectación es menor. Para limitar el alcance de la función, se suele limitar la aplicación de la misma a un conjunto de píxeles en torno al punto P, esta limitación espacial se denomina comúnmente tamaño del *kernel.*

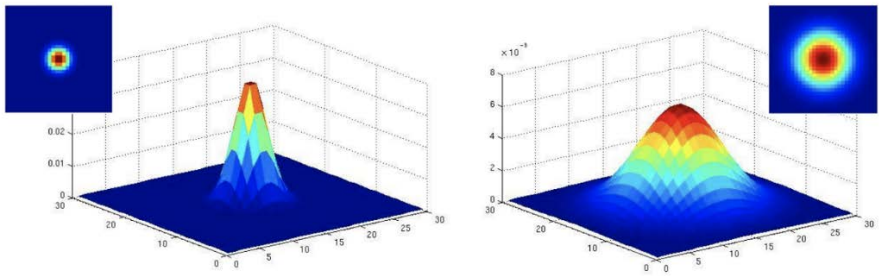

***Figura 22**: Ejemplo de funciones kernel gaussiano 2D para valores de desviación estándar diferentes. Fuente: [45].*

Esta implementación no tiene en cuenta la diferencia de tamaño entre cada uno de los objetos generada por la reducción de un espacio tridimensional a dos dimensiones. Para corregir esta deficiencia, algunos modelos aplicados al conteo de personas emplean un sistema de generación de mapas de densidad por geometría adaptativa. Estos sistemas se basan en la correlación entre los tamaños de las cabezas y la distancia entre las mismas [33], por lo que la desviación estándar σ se calcula de forma dinámica, para cada punto etiquetado, como el producto de un coeficiente reductor y la media de la distancia entre el punto etiquetado y sus vecinos más cercanos (generalmente dos y tres).

Si bien el desarrollo original implementaba esta última estrategia, el mismo se ha modificado para permitir la definición de valores de desviación estándar σ fijos. Esto se debe a que, como se explica en el apartado 4.4.1, el conjunto de datos que se utiliza en el presente proyecto no cumple las condiciones para las que se han diseñado las estrategias de geometría adaptativa descritas.

4.2.3. Entrenamiento

Para llevar a cabo el entrenamiento se realiza una división aleatoria del conjunto de datos denominado como de entrenamiento, ver apartado 4.3.2, por la cual se reserva un 10% de las imágenes para el proceso de validación. Esta validación es distinta a la evaluación final del error sobre el conjunto de test. El objetivo de esta evaluación es poder medir, durante el entrenamiento, la evolución de la precisión tras ciertas iteraciones de entrenamiento.

Además, a fin de obtener un mayor número de instancias de entrenamiento, y reducir así la necesidad de tener conjuntos de datos muy grandes para obtener buenos resultados, se utiliza una técnica de *data augmentation* o generación artificial de datos. La estrategia seguida para este fin [41] [46], consiste en generar nueve subimágenes de un cuarto de la resolución total de la imagen. Cuatro subimágenes se extraen de las cuatro esquinas, sin solapamiento, y las otros cinco se extraen de forma aleatoria de la imagen. De estas subimágenes se obtiene también sus correspondientes imágenes reflejadas horizontalmente.

Asimismo, a las imágenes se las somete a un proceso de normalización (**Figura 23**), mediante el cual, de cada píxel de canal RGB se sustrae el valor medio de los píxeles de ese canal, calculado sobre todas las imágenes del conjunto, y el resultado se divide por 255, valor máximo de cada píxel.

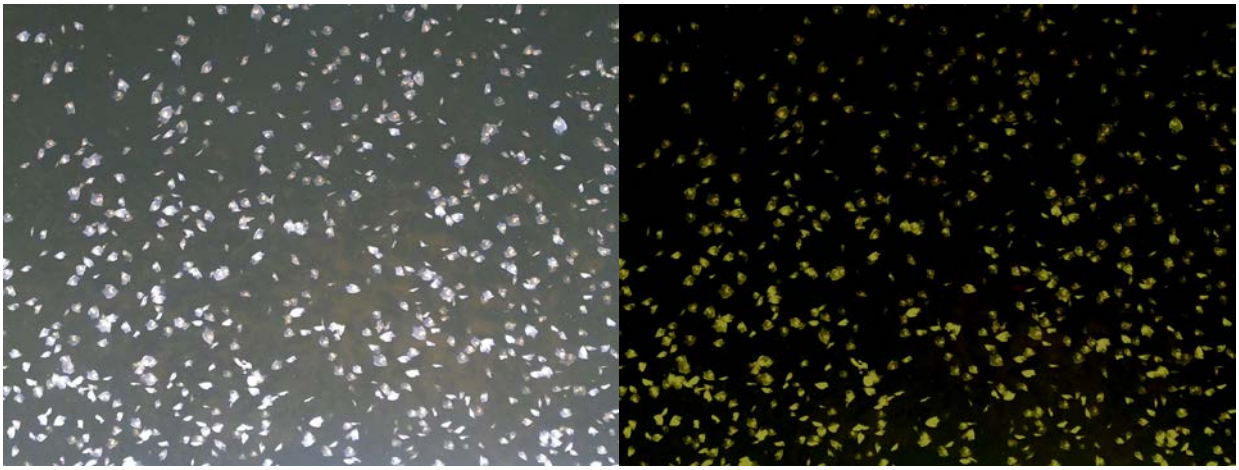

***Figura 23**: Ejemplo de resultado del proceso de normalización de la imagen.*

En relación a los parámetros del aprendizaje del modelo, se usa el algoritmo de optimización de descenso estocástico por gradiente (SGD) con una tasa de aprendizaje inicial igual a 0.0001 que se divide por un factor de 10 cada reiteración del proceso completo de entrenamiento. Para la inicialización de los pesos se utiliza una inicialización gaussiana aleatoria con una desviación estándar de 0.01. La red VGG16 se inicializa pre-entrenada con el conjunto de datos ImageNet.

El tamaño de *batch* (*batch size*), es decir, el número de imágenes con las que se entrena el modelo antes de calcular los errores y actualizar los valores de los pesos, es de 1.

Además, para la mejora del algoritmo de optimización SGD se utilizan las siguientes técnicas:

- ***Momentum***: se utiliza para disminuir las fluctuaciones excesivas en los cambios de peso a lo largo de las iteraciones consecutivas y mejorar con ello la velocidad de aprendizaje [47]. El valor que se utiliza para este parámetro es de 0.9.

- **Weight decay**: es una técnica de regularización cuyo objetivo principal es evitar un exceso de adaptación (*overfitting*) del modelo al conjunto de entrenamiento, lo cual afectaría a su generalización para nuevos datos. Este parámetro introduce una penalización en la función de coste con el fin de reducir los pesos durante la propagación hacia atrás del error [48].

4.3. Conjunto de datos

El conjunto de datos del que se dispone consiste en 156 imágenes RGB, de resolución 2560 x 1920 (ver ejemplo en **Figura 24**), anotadas en el Grupo de Aplicaciones Multimedia y Acústica (GAMMA) con una aplicación desarrollada a tal efecto y que, internamente se ha llamado Turbot. Como parte del proyecto se realizó el testeo de la aplicación y el etiquetado de algunas de las imágenes que componen el actual conjunto de datos.

Los fotogramas han sido extraídos de videos de cinco minutos de duración grabados en tanques de cría de rodaballos de forma paralela a la superficie del mismo. Estas grabaciones no recogen la totalidad de cada tanque sino una parte de los mismos, por lo que el número de rodaballos no es constante durante los vídeos.

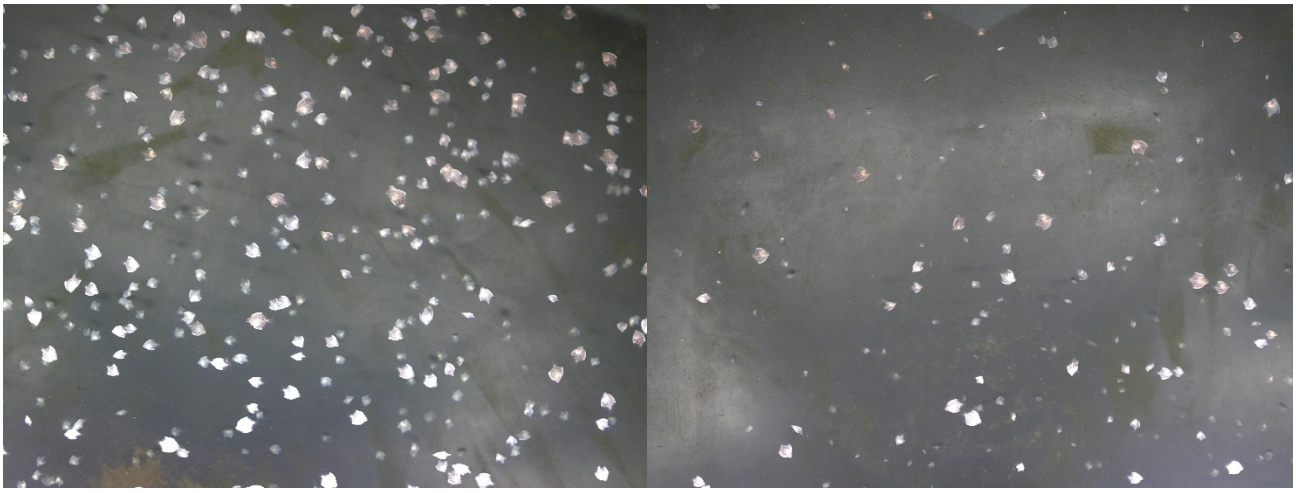

***Figura 24**: Ejemplo de dos fotogramas del conjunto de datos.*

Cada imagen esta acompaña de un archivo contenedor de datos binarios que utiliza el programa MATLAB con extensión "mat". El archivo, está generado con el programa de etiquetado Turbot, desarrollado por el Grupo de Aplicaciones Multimedia y Acústica. Dicha aplicación elige el fotograma central de cada vídeo y realiza un proceso de segmentación para identificar los objetos presentes en la imagen (ver ejemplo en **Figura 25**). Tras esto, el usuario debe evaluar cada objeto para determinar si se trata de un rodaballo aislado, un conjunto de rodaballos o si no es un rodaballo. En el caso de que se identifique un grupo de rodaballos el usuario debe señalar el centro de cada uno de los que conforman el grupo. Además, para cada anotación, el usuario puede señalar el grado de confianza entre "seguro" y "con dudas".

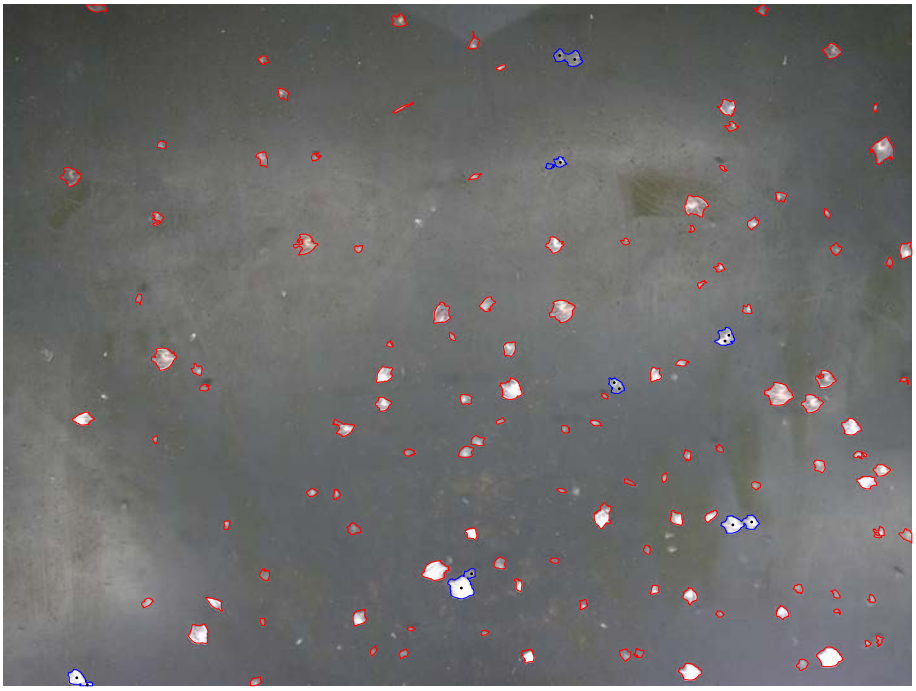

***Figura 25**: Ejemplo de imagen del conjunto de datos con la información de etiquetado superpuesta. En rojo el contorno de rodaballos individuales y en azul los conjuntos de rodaballos.*

La distribución de rodaballos anotados en las imágenes se observa en la **Figura 26**, con una media de 251 rodaballos por imagen.

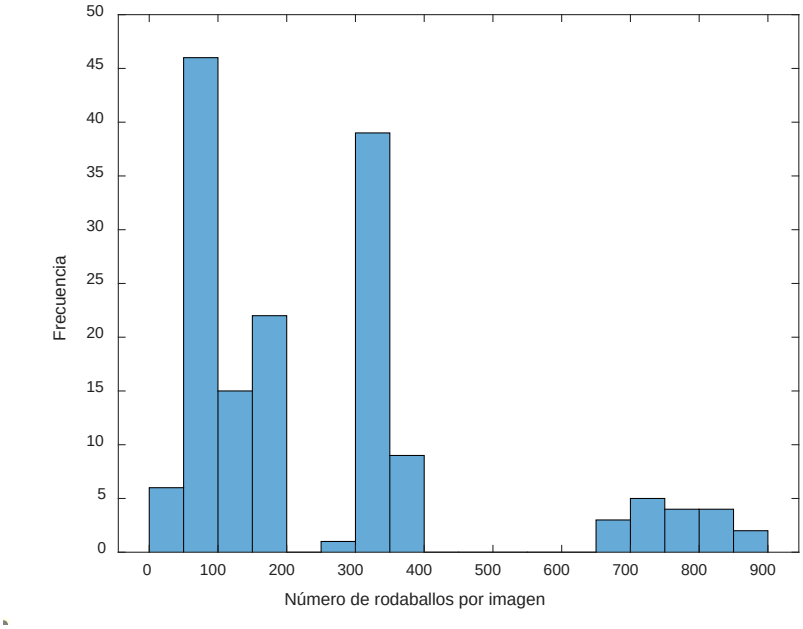

***Figura 26**: Distribución del número de rodaballos etiquetados por fotograma.*

4.3.1. Adaptación del archivo de etiquetas

El archivo de etiquetado generado por la aplicación Turbot en formato Matlab contiene las siguientes variables (**Tabla 2**).

***Tabla 2**: Descripción de la estructura del archivo generado por el programa de etiquetado Turbot.*

Nombre	Descripción
BS	Estructura con la información de los bordes de cada uno de los objetos segmentados.
countObjectS	Último objeto etiquetado
iFrameS	El número de fotograma al que corresponde la imagen etiquetada
LS	Matriz con las dimensiones de la imagen e información con la ubicación de los objetos segmentados píxel por píxel.
mrPropsS	Matriz con información de todos los objetos segmentados incluyendo características morfológicas y el valor numeral de la etiqueta que se le ha asignado a ese objeto.
nS	El número total de objetos obtenidos de la segmentación
turbotCentroidS	Matriz con la información de la ubicación del centroide de cada uno de los objetos etiquetados como rodaballos. En el caso de conjuntos de rodaballos, la ubicación se corresponde a la indicada como centro por el usuario que ha etiquetado la imagen.

A fin de adaptar el archivo proporcionado en el conjunto de datos a la estructura del archivo necesario para entrenar el modelo elegido, se ha desarrollado una rutina en Matlab para extraer la información de interés de este archivo y crear uno nuevo. El archivo usado para entrenar el modelo

es también de formato Matlab (extensión “mat”). La estructura de variables dentro del archivo se muestra en la **Figura 27**.

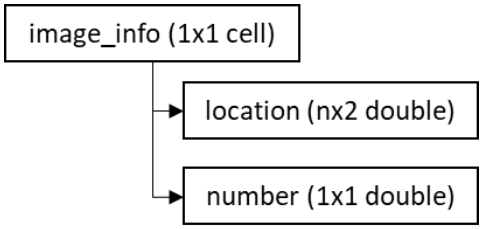

***Figura 27**: Estructura de los archivos de etiquetas para el entrenamiento del modelo.*

El archivo contiene una variable tipo “arreglo de celdas” denominada *image_info* que contiene dos variables. La primera denominada *location*, con n pares de instancias de tipo *double,* correspondientes a las coordenadas (x, y) de los n objetos etiquetados. Por último, una variable de tipo *double*, denominada *number* que contiene el número n de objetos etiquetados en la imagen.

4.3.2. Conjuntos de datos para entrenamiento y validación

Para la generación del conjunto de datos se han dividido las imágenes de forma aleatoria en 124 (80%) para entrenamiento y 32 (20%) para validación. La distribución de rodaballos en ambos conjuntos es de 246 y 273 rodaballos de media por imagen y su distribución se puede observar en las **Figura 28** y **Figura 29**, respectivamente.

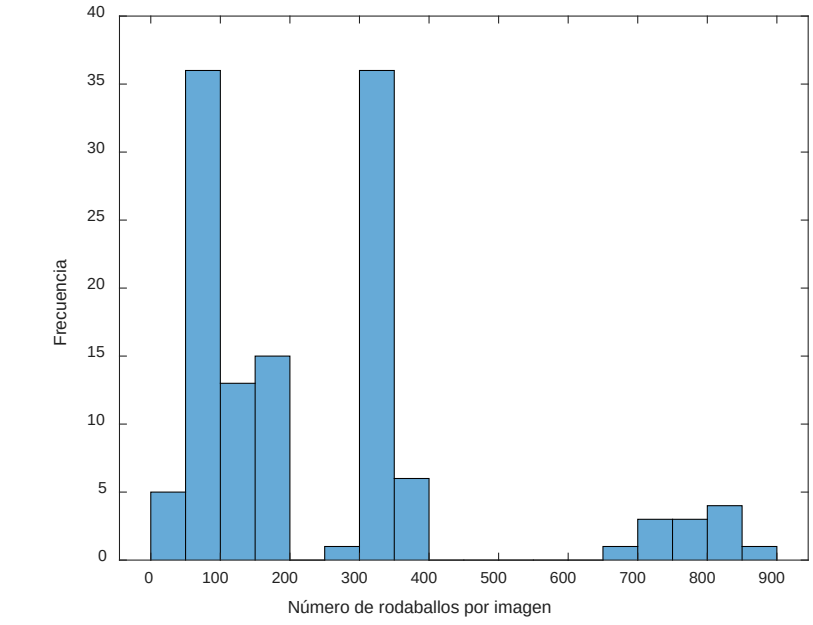

***Figura 28**: Distribución del número de rodaballos por imagen en el conjunto de imágenes de entrenamiento.*

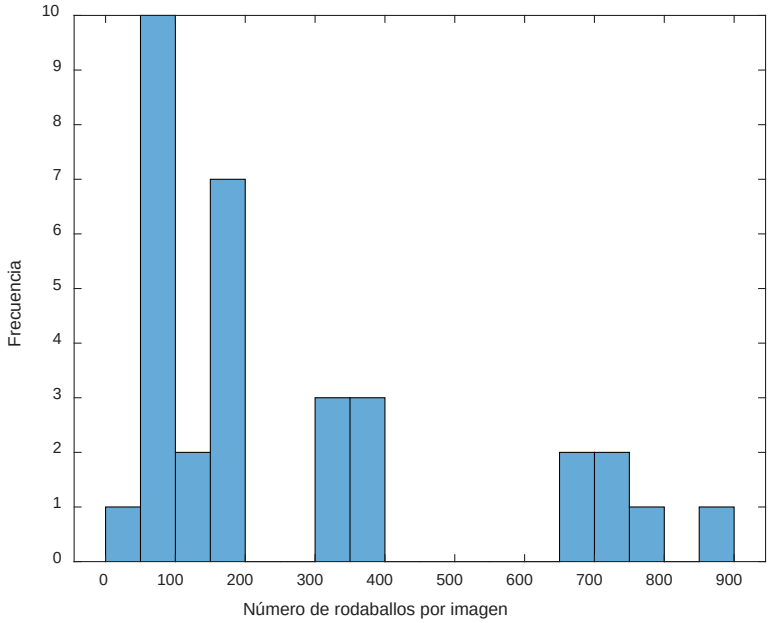

***Figura 29**:Distribución del número de rodaballos por imagen en el conjunto de imágenes de evaluación.*

Dado que el desarrollo elegido divide la imagen en cuadrículas de 64x64 píxeles, y que la imagen original tiene un tamaño de 2560 x 1920, se generarían 40x30 cuadrículas, es decir, 1.200. Teniendo en cuenta que el número medio de rodaballos por imagen es de 251, esto arroja una media de 0.2 rodaballos por cuadro. A fin de evitar una segmentación excesiva de los mismos, se ha decidido reducir las imágenes con un factor de 2.5, resultando en unas dimensiones de 1024x768. De esta forma se obtienen 16x12 cuadros, y una media de 1.3 rodaballos por cuadro.

Asimismo, se han adaptado los valores de los archivos de etiquetas para que se correspondan las posiciones de los rodaballos anotados con los píxeles de las nuevas imágenes.

La estructura de archivos y carpetas que debe tener el conjunto de datos para que sea compatible con el programa es el que se muestra en la **Figura 30**. Existirá una carpeta para cada conjunto: entrenamiento, con el nombre *train_data*; y test, con el nombre *test_data*. Cada una de estas carpetas contendrá una carpeta llamada *images* y otra llamada *ground_truth*. La primera contendrá las imágenes en formato JPG, con el nombre “IMG_” seguido de un número entero consecutivo, empezando por uno. La segunda carpeta contendrá los archivos de etiquetas con extensión “mat” y mismo nombre que la imagen con la que se corresponden, precedido por “GT_”.

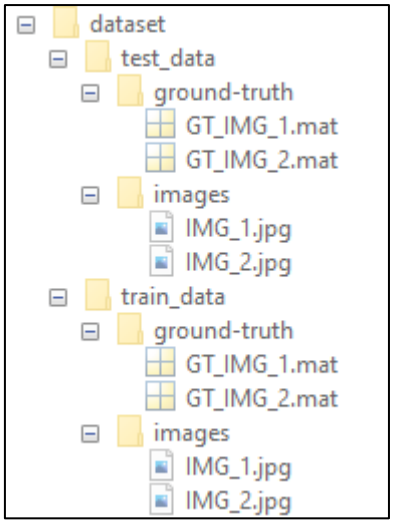

***Figura 30**: Estructura de archivos del conjunto de datos para entrenamiento y validación.*

4.4. Parámetros de optimización

Si bien el desarrollo implementado sigue el modelo planteado en la sección 4.1.1, y se utilizarán los parámetros propuestos para el entrenamiento, existen dos variables que son dependientes del conjunto de datos a usar y por lo tanto deben ser adaptadas al caso actual.

Por un lado, es necesario determinar un valor de desviación estándar para la generación de los mapas de densidad. Como se expone en el apartado 4.2.2, se ha optado por seguir una estrategia de valor de desviación estándar fijo, ya que el conjunto no reúne las condiciones para utilizar técnicas de geometría adaptativa. Por otro lado, es necesario elegir un valor máximo para el clasificador del modelo que sea adecuado a la distribución de valores por cuadro presente en el conjunto de datos.

4.4.1. Valor de desviación estándar para generación de mapas de densidad

Tal como se explica en el apartado 4.2.2, para la generación de mapas de densidad es necesario obtener el valor de desviación estándar del *kernel* gaussiano. En las estrategias de generación de mapas de densidad por geometría adaptativa se suele calcular este valor calculando el producto de la distancia media a los vecinos más cercanos y multiplicándolo por un coeficiente reductor, generalmente 0.3 [33]. Sin embargo, el cálculo de la desviación estándar mediante geometría adaptativa está diseñado para imágenes en las que el tamaño del objeto se distribuye de forma uniforme en las distintas regiones de la imagen, por ejemplo, una imagen de una calle donde las cabezas de las personas que están en el frente de la imagen tienen tamaños similares entre sí, ocurriendo lo mismo con los que están al final y, sin embargo, los tamaños medios de unos (frente) y otros (fondo) son muy diferentes. Igual caso se da en imágenes de vehículos en carreteras.

Sin embargo, en las imágenes de los tanques de cría de rodaballo el tamaño no se distribuye de forma homogénea por regiones de la imagen ya que el tamaño varía principalmente con la distancia a la superficie. Por ello, se pueden observar crías de rodaballos de gran tamaño, por estar cerca de la superficie, junto a otras pequeñas, por estar estas últimas en un plano distinto, por ejemplo, el fondo del tanque. Dado que en este caso los vecinos no se encuentran en el mismo plano, se ha elegido emplear una desviación estándar fija para la generación de los mapas de densidad.

A fin de poder determinar el valor de la desviación estándar, se han analizado las imágenes del conjunto de datos, aprovechando los datos de la segmentación que se realiza mediante el proceso

de etiquetado (ver **Figura 32**). El área media de los rodaballos etiquetados como aislados en las imágenes originales es de 922 pixeles, por lo que, asumiendo la figura de los rodaballos como cuadrada, obtendríamos una longitud media de 30 píxeles. Lo cual, para el caso de las imágenes que se emplearán en el entrenamiento, cuyo tamaño se ha reducido con un factor 2.5, equivaldría a una longitud media de 12.13 píxeles. Asimismo, el tamaño máximo del conjunto es de 28 píxeles, se observa la distribución en la **Figura 32** y en la **Tabla 3** algunos de los percentiles superiores.

***Tabla 3**: Percentiles de la distribución de longitudes de los rodaballos etiquetados.*

Percentil	Longitud (px)
65	13.4
75	14.8
85	16.8
95	20.3

Como ya se ha visto, en las estrategias de geometría adaptativa, se adopta un coeficiente de 0.3 sobre el valor medio de la distancia de un objeto a sus vecinos más cercanos. Si se consideran algunos de los conjuntos de datos de conteo de personas sobre los que se han aplicado estas estrategias de generación de mapas de densidad, se observa que, en las imágenes con alta densidad, la distancia entre los centros de las cabezas es, aproximadamente, la anchura de las mismas, al estar pegadas unas con otras. Por ello, se puede considerar como valor mínimo de sigma el resultado de aplicar el coeficiente 0.3 a la media de la longitud, 12, lo cual arroja un valor mínimo de sigma de 3.6.

Es necesario tener en cuenta que valores de desviación estándar muy pequeños o muy grandes en relación a los objetos aumentan el nivel de ruido de los píxeles afectados, lo cual tiene repercusiones en el modelo.

A fin medir cómo afecta el valor de sigma a la precisión del modelo, se generarán mapas de densidad generados con distintos valores de sigma entre 3 y 15, en intervalos de tres, todos ellos con un tamaño de *kernel* de 30 píxeles (ver ejemplo en **Figura 31**).

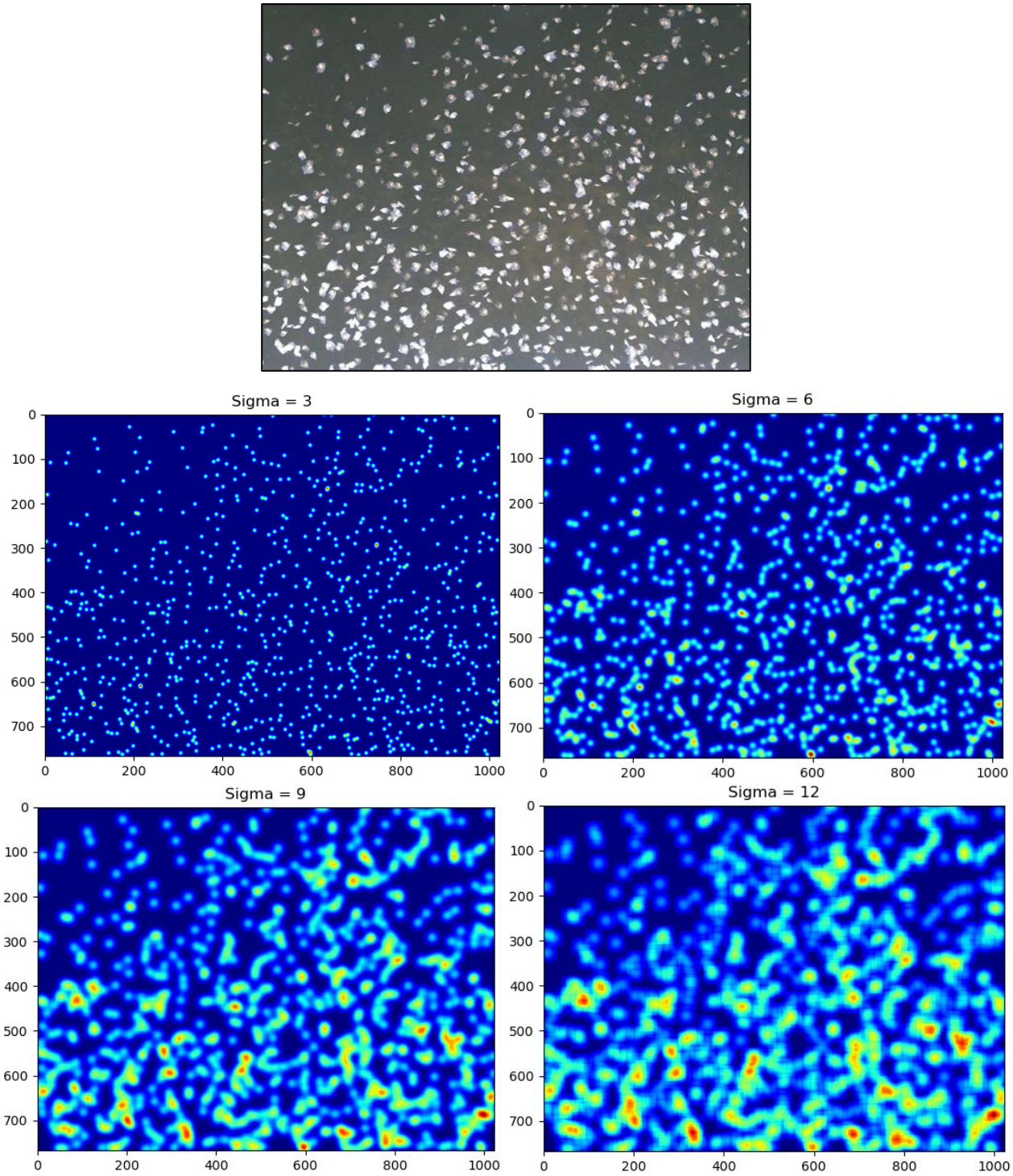

***Figura 31**: Imagen del conjunto de datos y visualización de sus mapas de densidad generados con valores de sigma 3, 6, 9 y 12.*

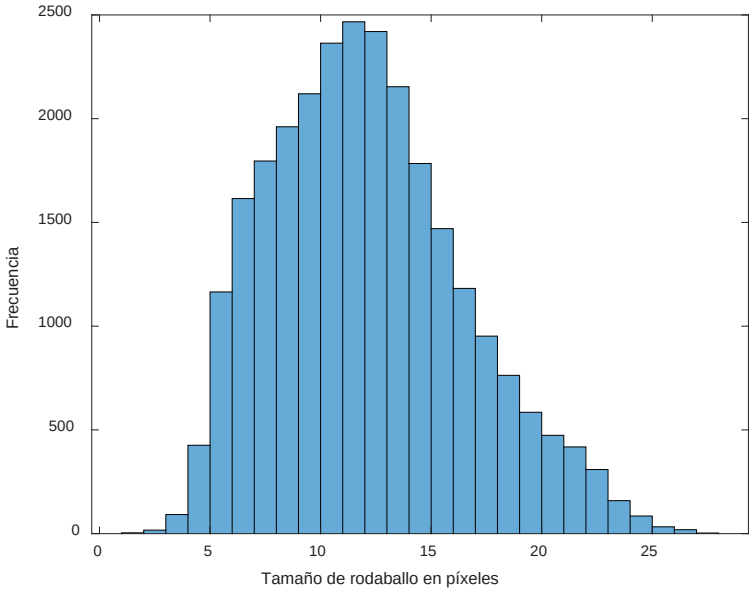

***Figura 32**: Distribución de la longitud de los rodaballos etiquetados en imágenes de tamaño 1024x768.*

4.4.2. Dimensión del clasificador

Analizando la distribución de objetos por cuadros de 64x64 (**Figura 33**), en las imágenes ya reducidas a una resolución de 1024x768, se observa que el percentil 95 (**Tabla 4**) se corresponde con el valor de 5 rodaballos por cuadro de 64x64, por lo que, siguiendo las recomendaciones de los desarrolladores del modelo, será el valor que se elegirá como punto de partida para el entrenamiento del modelo. Sin embargo, se realizarán pruebas con los valores inmediatamente inferiores y superiores para analizar su variación.

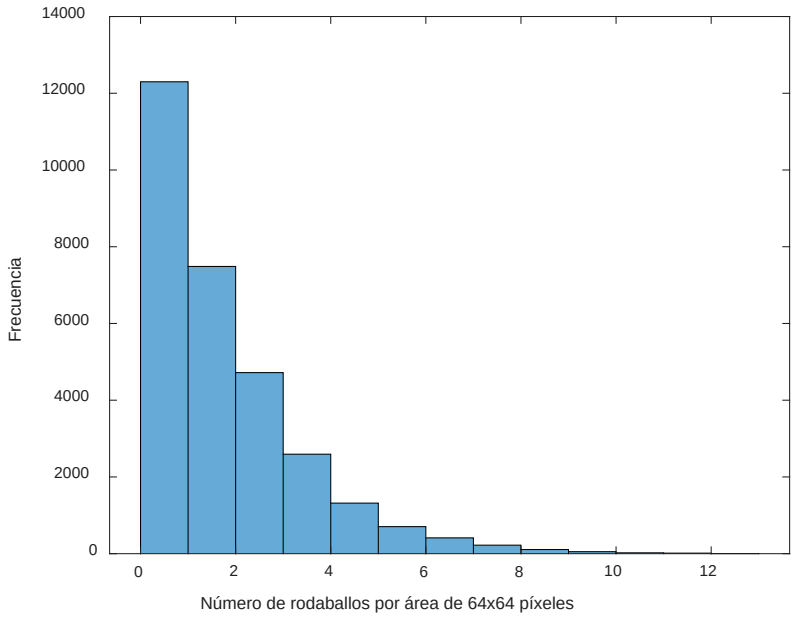

***Figura 33**: Distribución del número de rodaballos por cuadros de 64x64 píxeles.*

***Tabla 4**: Percentiles de la distribución del número de rodaballos por cuadros de 64x64 píxeles.*

Percentil	Valor
65	1
75	2
85	3
95	5

5. Resultados

A fin de obtener los parámetros óptimos de desviación estándar para la generación de los mapas de densidad (ver apartado 4.4.1) y de C_{Max} del clasificador (ver apartado 4.4.2), se ha partido de los valores calculados en los apartados anteriores, valor de desviación estándar igual a 12 y valor de C_{Max} igual a 5, y se han analizado los resultados entrenando con los valores superiores e inferiores para analizar la evolución. Teniendo en cuenta el factor de inicialización aleatoria de los pesos de la red y la generación aleatoria de los conjuntos de entrenamiento y validación durante la fase de entrenamiento, se han realizado dos procesos de entrenamiento para cada par de valores y se ha hallado la media de ambas iteraciones.

Para medir el error se han usado tres parámetros distintitos:

- Error Absoluto Medio (MAE): calculado como promedio de diferencias absolutas entre los valores objetivo $C(i)$ y las predicciones $\hat{C}(i)$.

$$MAE = \frac{1}{N}\sum_{i=1}^{N} | C(i) - \hat{C}(i)| \tag{13}$$

- Raíz del Error Cuadrático Medio (RMSE): calculado como la raíz cuadrada del promedio de la diferencia al cuadrado entre valores objetivo $C(i)$ y las predicciones $\hat{C}(i)$.. Esta métrica es útil para detectar desviaciones grandes en algunas de las muestras del conjunto de datos que en el MAE es más fácil que se diluyan en la media.

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(C(i) - \hat{C}(i))^2} \tag{14}$$

- Error Porcentual Medio Absoluto (MAPE): calculado como el promedio de la relación entre los errores absolutos ($|C(i) - \hat{C}(i)|$) y los valores objetivo $C(i)$, midiendo, de esta forma, la desviación en términos porcentuales. Esta métrica resulta interesante cuando la distribución de valores objetivo es grande, como en el caso actual del conjunto de datos utilizados.

$$MAPE(\%) = \frac{100}{N}\sum_{i=1}^{N}\frac{|C(i) - \hat{C}(i)|}{C(i)} \tag{15}$$

A fin de identificar el número de iteraciones (*epochs*) del algoritmo de entrenamiento necesario para que el error se estabilice, se realizó un entrenamiento con 1000 iteraciones y valores desviación estándar igual a 12 y C_{Max} igual a 5, observándose que tanto el MAE (**Figura 34**) como el MSE se estabilizan en torno a la iteración 400. Por este motivo, y a fin de optimizar los recursos durante el entrenamiento, el resto de pruebas se realizaron con este número de iteraciones.

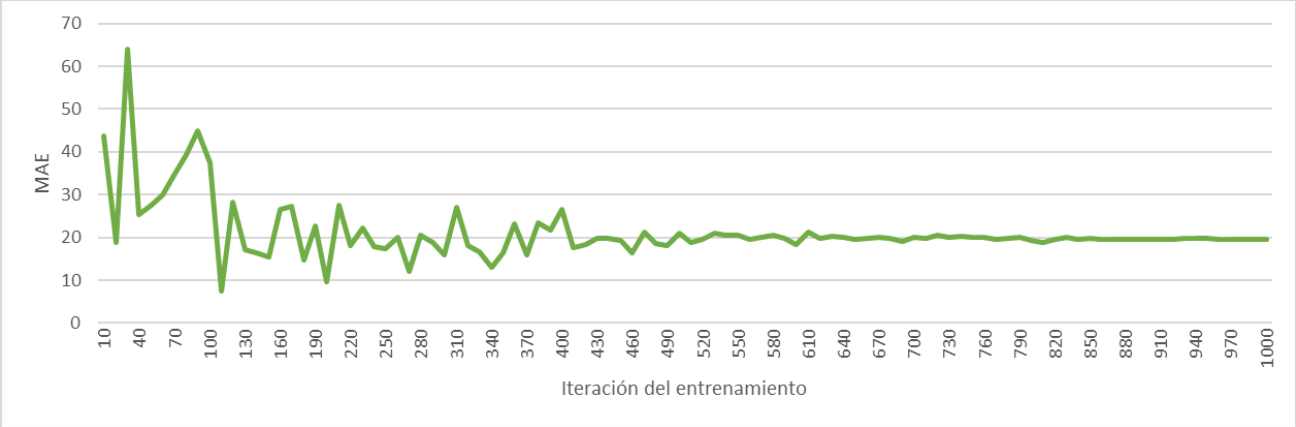

***Figura 34**: Evolución del MAE durante el entrenamiento con valores de sigma = 12 y C_{Max} = 5.*

5.1. Impacto de la desviación estándar en la generación de mapas de densidad

A fin de evaluar cómo afecta a la precisión del modelo la elección del valor de desviación estándar para el *kernel* gaussiano durante la generación de los mapas de densidad, se entrenó el modelo con un valor de C_{Max} igual a 5 y los mapas de densidad generados para distintos valores de desviación estándar, entre 3 y 15, en intervalos de tres. Tal como se observa en la **Tabla 5** y la **Figura 35**, no existe un impacto significativo en los errores del modelo para valores de desviación estándar pequeño.

***Tabla 5**: Errores para mapas de densidad generados con diferentes valores de desviación estándar.*

Valor de σ	MAE	RMSE	MAPE
3	9.00	18.82	3.52%
6	11.66	19.46	4.04%
9	11.05	19.22	3.56%
12	9.66	18.20	3.48%
15	10.62	18.09	3.69%

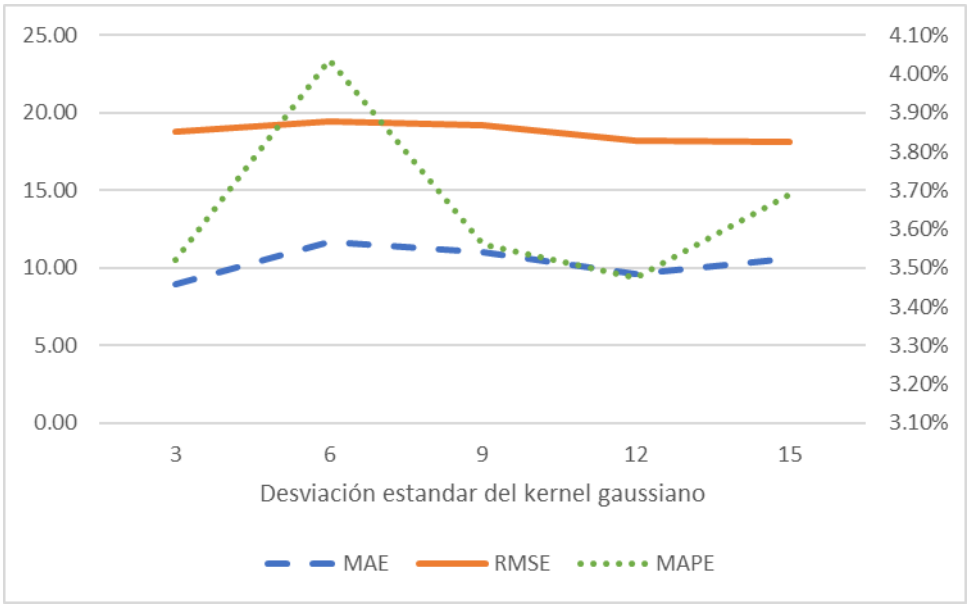

***Figura 35**: Evolución de los errores en el modelo (con CMax = 5) con el tamaño de la desviación estándar del kernel gaussiano en la generación de los mapas de densidad.*

Para el resto de pruebas se utilizarán los mapas de densidad generados con un valor de desviación estándar de 12 ya que, si bien presenta ligeramente peor valor de MAE que los generados con desviación estándar de 3, los valores de MSE y MAPE son mejores, por lo que las desviaciones serán más homogéneas.

5.2. Determinación del valor de C_{Max}

Los creadores del modelo SS-DCNet obtuvieron los mejores resultados de precisión del modelo para un valor máximo del clasificador (C_{Max}) igual al valor del percentil 95 de la distribución de objetos en cuadros de 64x64 píxeles. Este valor para el conjunto de datos actual es de 5. A fin de comprobar si esta correlación es de aplicación al conjunto actual, se ha analizado el error al entrenar el modelo con valores de C_{Max} inferiores y superiores a 5.

Tal como se observa en la **Tabla 6** y la **Figura 36**, para un valor de C_{Max} igual a 5 se obtienen los menores errores tanto en MAE como MAPE. Sin embargo, para un valor de C_{Max} igual a 6 el RMSE presenta un valor ligeramente menor, lo que significa que existen menos desviaciones grandes. Pese a esto, se considera que la diferencia tanto en MAE como en MAPE es más significativa que la de RMSE y se utilizarán un valor de C_{Max} igual a 5 durante el resto de las pruebas.

***Tabla 6**: Errores para diferentes valores de C_{Max} con un valor fijo de desviación estándar de 12.*

C_{max}	MAE	RMSE	MAPE
2	14.45	25.98	3.90%
3	15.02	27.26	4.13%
4	14.67	26.49	4.09%
5	9.66	18.20	3.48%
6	10.39	18.05	3.64%

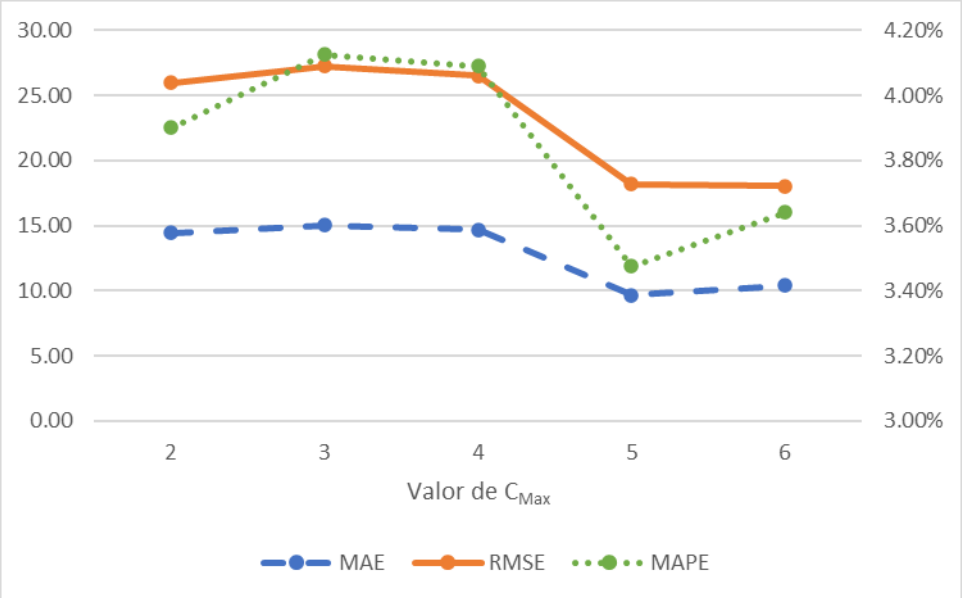

***Figura 36**: Evolución de los errores con el tamaño de C_{Max} para un valor de desviación estándar constante de 12.*

A continuación (**Figura 37**), se muestra la relación entre los valores de las etiquetas y los valores predichos por el modelo para un valor de C_{Max} de 5 y una desviación estándar de 12. Como se puede observar, existe mayor dispersión para valores superiores, aunque hay que tener en cuenta que se trata de valores absolutos.

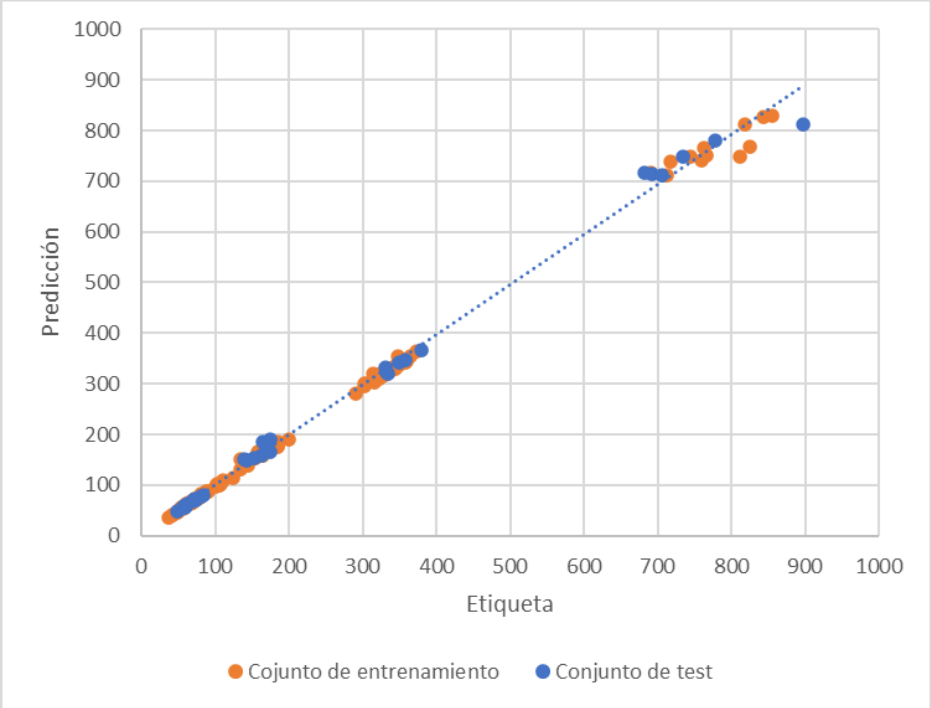

***Figura 37**: Proyección del valor de la etiqueta frente al valor predicho para los conjuntos de entrenamiento (naranja) y de test (azul).*

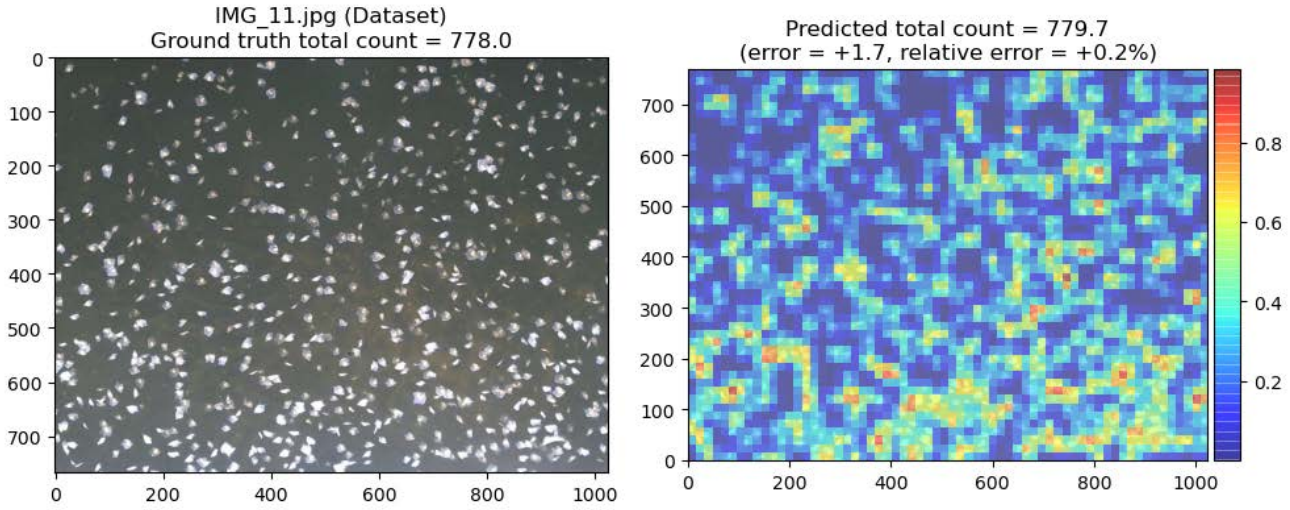

***Figura 38**: Ejemplo de resultado de la predicción (derecha) sobre una imagen del conjunto de test (izquierda).*

5.3. Generalización para imágenes con más objetos

Con el objetivo de evaluar una de las características del modelo, según la cual el mismo puede ser aplicado a imágenes que contengan más rodaballos de los que existían en las imágenes de entrenamiento (ver apartado 4.1), se ha creado un conjunto de datos donde la división entre las imágenes de entrenamiento y test se ha realizado por el número de rodaballos en cada imagen,

resultando en 129 imágenes de entrenamiento con menos de 350 rodaballos en cada una, y 27 imágenes de test con entre 350 y 898 rodaballos etiquetados.

La **Tabla 7** muestra el error asociado al entrenamiento con este el conjunto de test. Asimismo, a fin de comparar los resultados del modelo entrenado con el conjunto de datos actual y aquel entrenado con el conjunto de datos original, detallado en el apartado 4.3.2, se ha evaluado el error de estos modelos sobre las únicas 9 imágenes comunes en los conjuntos de test de ambos, mostrándose estos resultados en la **Tabla 8**. Como se aprecia, si bien la precisión es mayor para el modelo entrenado con el conjunto original, las métricas de los errores para el modelo entrenado con el conjunto de datos actual mantienen unos errores de orden similar.

***Tabla 7**: Errores del modelo entrenado con el conjunto de datos generado en base a la densidad de objetos en las imágenes.*

MAE	RMSE	MAPE
19.33	27.45	3.00%

***Tabla 8**: Comparación de errores entre ambos entrenamientos sobre las 9 imágenes de test comunes.*

	MAE	RMSE	MAPE
Conjunto de datos original	21.59	32.39	3.23%
Conjunto de datos por densidad	27.91	37.26	3.88%

Además, en la **Figura 39** se observa como la desviación de las predicciones, si bien un poco más dispersa para valores altos respecto a la **Figura 37**, sigue manteniendo una desviación aceptable, por lo que efectivamente, el modelo mantiene una precisión aceptable para imágenes con mayor densidad y número de objetos que los del conjunto de entrenamiento.

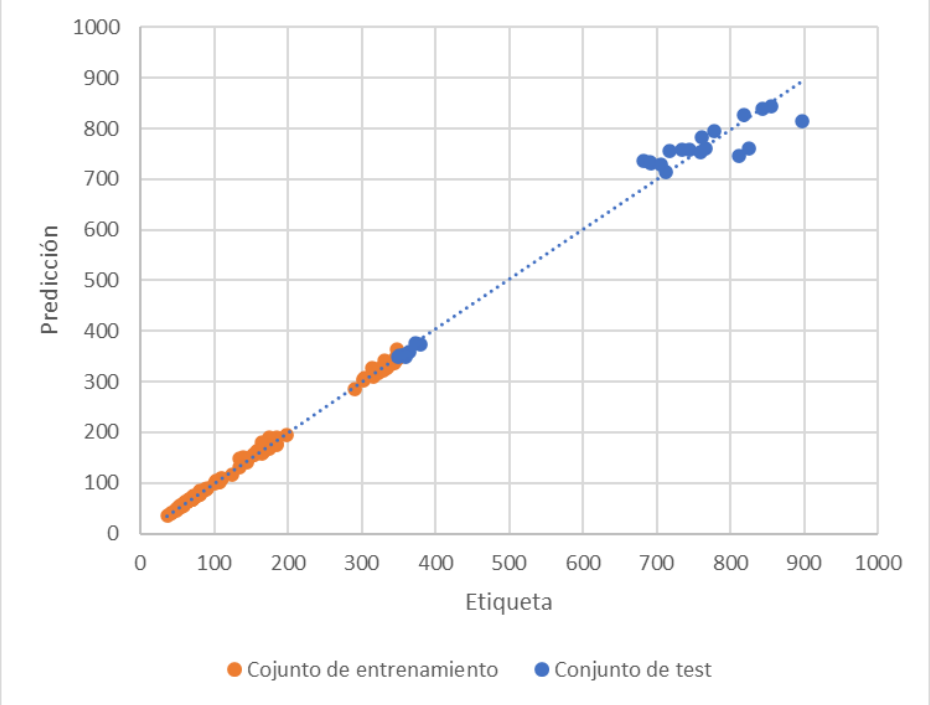

***Figura 39**: Proyección del valor de la etiqueta frente al valor predicho para los conjuntos de entrenamiento (naranja) y de test (azul).*

5.4. Evolución del error con el tamaño del conjunto de datos

Uno de los factores clave de los modelos de aprendizaje máquina es el contar con un buen conjunto de datos para el entrenamiento. Sin embargo, esto suele ser un factor limitante en muchas aplicaciones ya que, o no se cuenta con suficientes datos, o el proceso de etiquetado es tedioso. A fin de medir como responde el actual modelo a conjuntos de datos de distintos tamaños y poder dimensionar correctamente este en futuros desarrollos o proyectos donde se emplee el presente modelo, se han creado de forma aleatoria tres subconjuntos de entrenamiento con 10, 25 y 50 imágenes elegidas únicamente de las 124 imágenes del conjunto de entrenamiento original, manteniendo así el mismo conjunto de test original para realizar la validación de los entrenamientos. Cada subconjunto se ha generado por duplicado, con igual tamaño, pero distintas imágenes, y se ha hallado la media de ambos errores. A continuación (**Tabla 9** y **Figura 40**), se muestra la evolución de los errores con el tamaño del conjunto de entrenamiento.

***Tabla 9**: Errores en los modelos entrenados con 10, 25, 50 y 124 imágenes.*

Tamaño	MAE	RMSE	MAPE
10	26.99	56.57	6.19%
25	11.61	20.64	3.73%
50	10.05	19.82	3.39%
124	9.66	18.20	3.48%

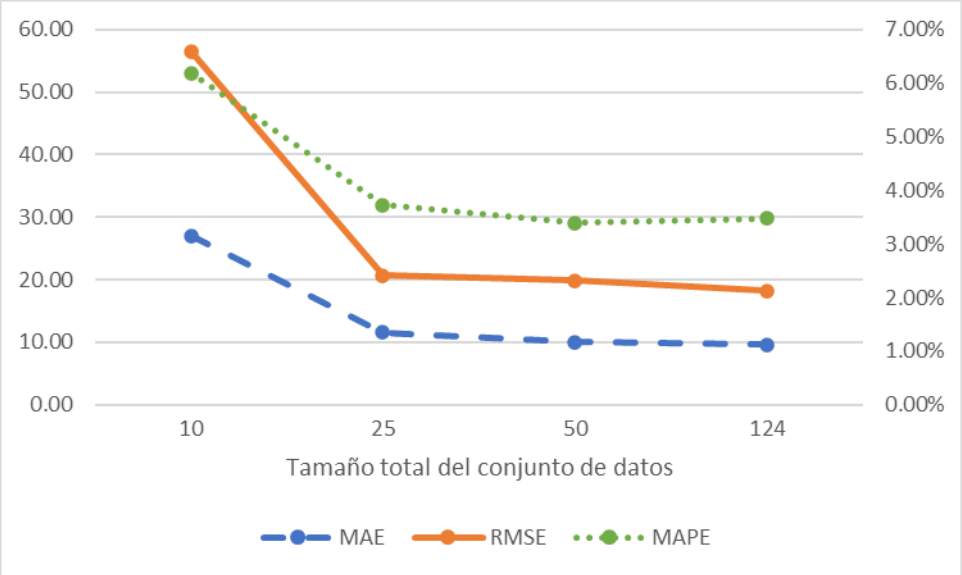

***Figura 40**: Evolución del error con el tamaño del conjunto de datos usado para el entrenamiento del modelo.*

Tal como se aprecia, a partir de 25 imágenes los valores de los errores son del mismo orden que los del modelo entrenado con el conjunto original de 124 imágenes.

A fin de contextualizar estos resultados, hay que tener en cuenta, primero, que se han seguido estrategias de aumento de datos (ver apartado 4.2.3) y que el modelo recibe como entrada parches de 64x64 píxeles, por lo que cada imagen en realidad son 216 instancias de entrenamiento del modelo; y segundo, que los conjuntos de datos, tanto de entrenamiento como de test, presentan homogeneidad tanto en el fondo como en los objetos a contar (rodaballos) sin variaciones de perspectiva, lo cual facilita y reduce la labor de aprendizaje. Además, se debe recordar que la red VGG16 de codificación se encuentra ya entrenada con el conjunto de datos ImageNet.

5.5. Medición de la evolución del número de rodaballos en los vídeos

Si bien el objetivo del desarrollo es la implementación de un modelo para el conteo de rodaballos en imágenes, la herramienta implementada permite evaluar la evolución del número de rodaballos en un vídeo mediante la inferencia de sus fotogramas. De esta forma, se puede evaluar también el modelo con nuevas imágenes no anotadas. En este sentido, si bien como se explica en el apartado 4.3, lo videos son grabaciones de solo una parte del tanque, por lo que el número de rodaballos puede variar según estos entren o salgan del cuadro de grabación, no se debe esperar cambios bruscos en la evolución de la predicción durante la duración del vídeo.

A efectos de conocer cómo es esta variación se han analizado 1 de cada 25 fotogramas de cinco de los vídeos utilizados para el etiquetado. En la **Tabla 10** se observan los valores medios de cada vídeo y su desviación media tanto absoluta como porcentual, así como la etiqueta asociada al fotograma central de cada vídeo.

Los nombres de los vídeos siguen la estructura de nombre de la cámara (L4T8 o L4T2), fecha y hora de inicio de la grabación y de finalización, ambas en formato (AAAAMMDDHHMMSS).

***Tabla 10**: Detalle de las desviaciones y los errores en los vídeos analizados.*

Vídeo	Etiqueta	Media	Desviación media	
L4T2-20220109074548-20220109075048	174	163	8.51	5.21%
L4T8-20220109074044-20220109074545	360	326	12.55	3.84%
L4T8-20220224140126-20220224140627	48	57	8.37	14.61%
L4T2-20220223222709-20220223223210	855	787	20.05	2.55%
L4T8-20220109080547-20220109081047	355	322	12.39	3.85%

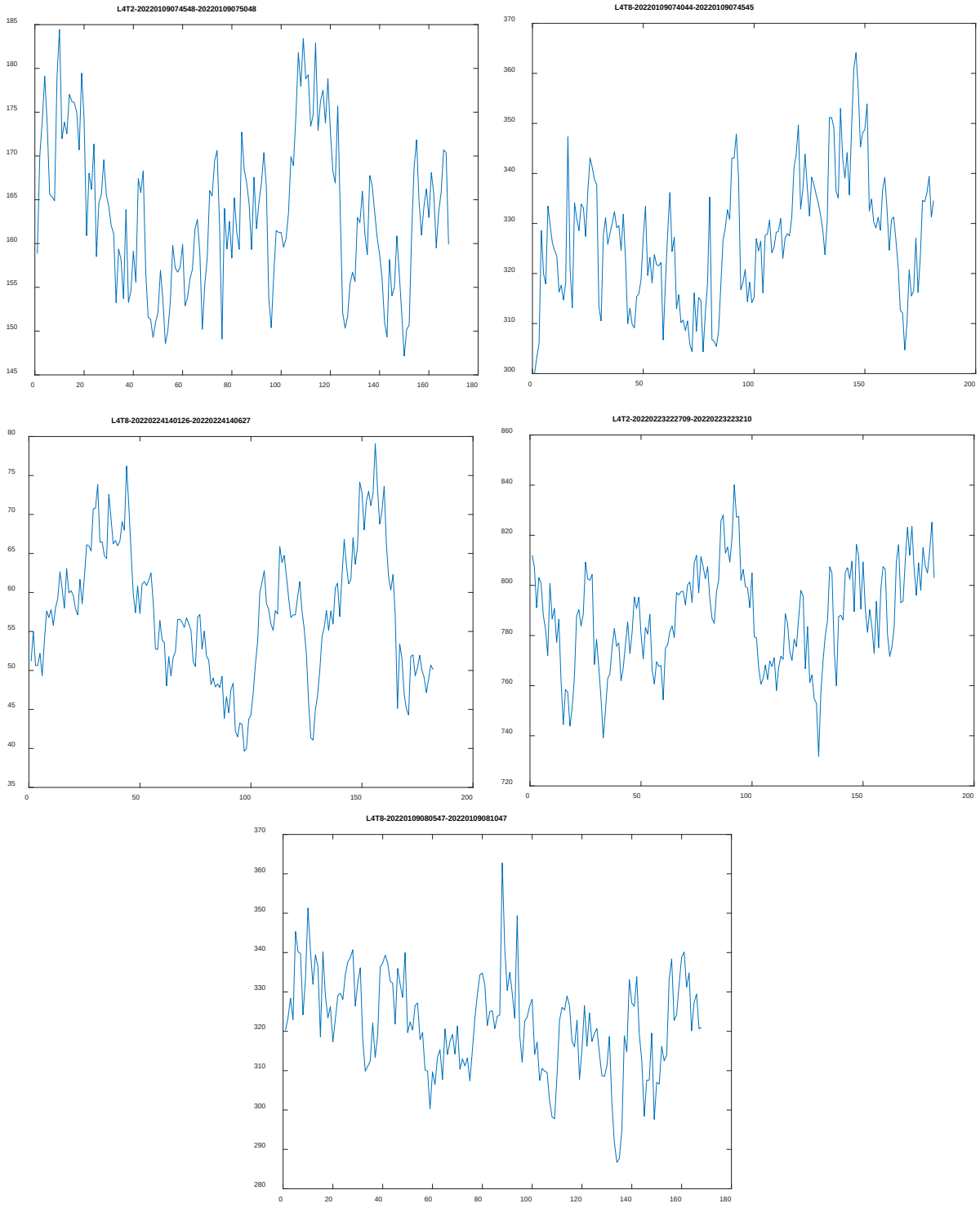

***Figura 41**: Evolución de la predicción del número de rodaballos en los vídeos analizados.*

6. Conclusiones

La aplicación de técnicas de IA para incrementar la eficiencia y competitividad del sector industrial queda de manifiesto, en este trabajo, por la capacidad de desplegar algoritmos de forma ágil que mejoren la eficiencia y reduzcan los costes de procesos como la estimación de biomasa en el ámbito de la acuicultura. En este caso, la aplicación de un modelo de redes neuronales convolucionales para el conteo de crías de rodaballos a partir de imágenes permite obtener una precisión más que aceptable en esta tarea, con un error medio menor al 3.5%, con una inversión relativamente pequeña tanto en hardware, para obtención de imágenes, como en desarrollo de software.

Garantizar al sector de la acuicultura herramientas precisas y de bajo coste para la estimación de biomasa no solo supone un impacto ecológico positivo, por la reducción de desperdicios de comida para los peces y la mejora en la gestión logística y comercial del proceso de cría, sino, además, tiene un impacto social y económico positivo para las pequeñas y medianas empresas de este sector, las cuales pueden mejorar su competitividad sin necesidad de grandes inversiones.

En concreto el modelo de redes convolucionales analizado, el SS-DCNet, no solo ha mostrado una alta precisión en la tarea de conteo de crías de rodaballos, sino que, además, con relativamente pocas imágenes etiquetadas, se pueden obtener resultados aceptables. Esto facilita enormemente la adaptación del modelo para contar otro tipo de peces o rodaballos en otras etapas de crecimiento, al no ser necesario tener que construir grandes conjuntos de datos, con la consiguiente inversión de tiempo que eso supone. Además, otras de las ventajas del modelo estudiado es la capacidad para contar imágenes donde la densidad y el número total de objetos sea muy superior al presente en las imágenes de entrenamiento, con los consiguientes beneficios de generalización que supone.

Si bien los resultados del modelo son más que aceptables, existen varias líneas de trabajo futuras, que pueden suponer vías para mejorar los resultados obtenidos.

En primer lugar, en los modelos de estimación de densidad, el proceso de generación de mapas de densidad es uno de los aspectos clave para la precisión de los resultados. Tal como se explica en el apartado 4.2.2, las características del conjunto de datos empleado no permiten la adopción de las estrategias de geometría adaptativa usadas en el conteo de personas. Sin embargo, podrían explorarse otras estrategias para la generación de los mapas de densidad, por ejemplo, adaptando la desviación estándar para cada punto etiquetado en base a las características morfológicas extraídas durante el proceso de segmentación para el etiquetado.

En segundo lugar, si bien el uso de una red de codificación VGG16 pre-entrenada ayuda a reducir la necesidad de un conjunto de datos de entrenamiento grande, es posible que entrenando el codificador desde cero con imágenes propias se pueda mejorar la precisión, ya que, en el conjunto de datos de ImageNet, con el que se ha pre-entrenado el codificador, pese a su gran extensión de imágenes y categorías, es posible que la existencia de imágenes sobre crías de rodaballos sea limitada o nula.

Bibliografía

[1] D. Li, Y. Hao and Y. Duan, "Nonintrusive methods for biomass estimation in aquaculture with emphasis on fish: a review," *Rev. Aquac.,* vol. 12, p. 1390–1411, 2020.

[2] D. Li, Z. Miao, F. Peng, L. Wang, Y. Hao, Z. Wang, T. Chen, H. Li and Y. Zheng, "Automatic counting methods in aquaculture: A review," *J. World Aquac. Soc.,* vol. 52, p. 269–283, 2021.

[3] S. Samoili, M. Lopez Cobo, . E. Gomez Gutierrez, G. De Prato, F. Martinez-Plumed y B. Delipetrev, «AI WATCH. Defining Artificial Intelligence,» Publications Office of the European Union, Luxembourg, 2020.

[4] Ministerio de Asuntos Económicos y Transformación , «Estrategia Nacional de Inteligencia Artificial,» Ministerio de Asuntos Económicos y Transformación Digital, 2020.

[5] X. Yang, S. Zhang, J. Liu, Q. Gao, S. Dong and C. Zhou, "Deep learning for smart fish farming: applications, opportunities and challenges," *Rev. Aquac.,* vol. 13, p. 66–90, 2021.

[6] T. M. Mitchell, Machine Learning, New York: McGraw-Hill, 1997.

[7] S. Badillo, B. Banfai, F. Birzele, I. I. Davydov, L. Hutchinson, T. Kam-Thong, J. Siebourg-Polster, B. Steiert and J. D. Zhang, "An introduction to machine learning," *Clin. Pharmacol. Ther.,* vol. 107, p. 871–885, 2020.

[8] A. Malhotra, «Tutorial on Feedforward Neural Network,» Medium, [En línea]. Available: https://medium.com/@akankshamalhotra24/tutorial-on-feedforward-neural-network-part-1-659eeff574c3. [Último acceso: 11 06 2022].

[9] M. Reddy, «Fundamentals of Deep Learning: First Principles Part II,» Medium, [En línea]. Available: https://manasreddy11.medium.com/fundamentals-of-deep-learning-first-principles-part-ii-cbcb0dbe0104. [Último acceso: 11 06 2022].

[10] N. Cingillioglu, «Machine Learning - Neural Networks,» Imperial College London, [En línea]. Available: https://www.doc.ic.ac.uk/~nuric/teaching/imperial-college-machine-learning-neural-networks.html. [Último acceso: 10 06 2022].

[11] Suvrit, S. Nowozin y S. J. Wright, Edits., Optimization for Machine Learning, London, England: MIT Press, 2011.

[12] K. Melcher, «A Friendly Introduction to [Deep] Neural Networks,» KNIME, [En línea]. Available: https://www.knime.com/blog/a-friendly-introduction-to-deep-neural-networks. [Último acceso: 11 06 2022].

[13] I. Goodfellow, Y. Bengio y A. Courville, Deep Learning, London, England: MIT Press, 2016.

[14] B. Wicht, Deep Learning feature Extraction for Image Processing, Université de Fribourg, 2018.

[15] D. Kalita, «Basics of CNN in Deep Learning,» Analytics Vidhya, [En línea]. Available: https://www.analyticsvidhya.com/blog/2022/03/basics-of-cnn-in-deep-learning/. [Último acceso: 11 06 2022].

[16] A. Zhang, Z. C. Lipton, M. Li, A. J. Smola y B. Werness, Dive into Deep Learning, 2021.

[17] P. Sharma , «A Comprehensive Tutorial to learn Convolutional Neural Networks from Scratch,» Analytics Vidhya, [En línea]. Available: https://www.analyticsvidhya.com/blog/2018/12/guide-convolutional-neural-network-cnn/. [Último acceso: 11 06 2022].

[18] C.-C. J. Kuo, "Understanding convolutional neural networks with a mathematical model," *J. Vis. Commun. Image Represent.,* vol. 41, p. 406–413, 2016.

[19] «Deep Learning Bible,» Wikidocs, [En línea]. Available: https://wikidocs.net/book/7972. [Último acceso: 11 06 2022].

[20] T. Woods, «Softmax Function,» Deep IA, [En línea]. Available: https://deepai.org/machine-learning-glossary-and-terms/softmax-layer. [Último acceso: 11 06 2022].

[21] C. Goyal, «20 Questions to Test your Skills on CNN,» Analytics Vidhya, [En línea]. Available: https://www.analyticsvidhya.com/blog/2021/05/20-questions-to-test-your-skills-on-cnn-convolutional-neural-networks/. [Último acceso: 11 06 2022].

[22] K. Simonyan y A. Zisserman, «Very deep convolutional networks for large-scale image recognition,» 2014.

[23] R. Thakur, «Step by step VGG16 implementation in Keras for beginners,» Towardsdatascience, [En línea]. Available: https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c. [Último acceso: 11 06 2022].

[24] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li y L. Fei-Fei, «ImageNet: A large-scale hierarchical image database,» de *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[25] M. Malathi y P. Sinthia, «Brain Tumour Segmentation Using Convolutional Neural Network with Tensor Flow,» *Asian Pacific Journal of Cancer Prevention : APJCP,* 2019.

[26] O. Ronneberger, P. Fischer y T. Brox, «U-Net: Convolutional Networks for Biomedical Image Segmentation,» 2015.

[27] V. Dumoulin y F. Visin, «A guide to convolution arithmetic for deep learning,» 2016.

[28] R. Perko, M. Klopschitz, A. Almer and P. M. Roth, "Critical aspects of person counting and density estimation," *J. Imaging,* vol. 7, p. 21, 2021.

[29] B. Li, H. Huang, A. Zhang, P. Liu and C. Liu, "Approaches on crowd counting and density estimation: a review," *Pattern Anal. Appl.,* vol. 24, p. 853–874, 2021.

[30] V. Lempitsky y A. Zisserman, «Learning To Count Objects in Images.,» 2010.

[31] R. Gouiaa, M. A. Akhloufi and M. Shahbazi, "Advances in convolution neural networks based crowd counting and density estimation," *Big Data Cogn. Comput.,* vol. 5, p. 50, 2021.

[32] L. Wang, F. Zhou and H. Zhao, "Crowd Density Estimation based on Global Reasoning," *J. Robot. Netw. Artif. Life,* vol. 7, p. 279, 2020.

[33] Y. Zhang, D. Zhou, S. Chen, S. Gao y Y. Ma, «Single-image crowd counting via multi-column convolutional neural network,» de *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[34] S. Zhang, X. Yang, Y. Wang, Z. Zhao, J. Liu, Y. Liu, C. Sun and C. Zhou, "Automatic fish population counting by machine vision and a hybrid deep neural network model," *Animals (Basel),* vol. 10, p. 364, 2020.

[35] L. Zhang, W. Li, C. Liu, X. Zhou and Q. Duan, "Automatic fish counting method using image density grading and local regression," *Comput. Electron. Agric.,* vol. 179, p. 105844, 2020.

[36] J. Zhang, G. Yang, L. Sun, C. Zhou, X. Zhou, Q. Li, M. Bi and J. Guo, "Shrimp egg counting with fully convolutional regression network and generative adversarial network," *Aquacult. Eng.,* vol. 94, p. 102175, 2021.

[37] J. Zhang, H. Pang, W. Cai and Z. Yan, "Using image processing technology to create a novel fry counting algorithm," *Aquac. Fish.,* vol. 7, p. 441–449, 2022.

[38] X. Yu, Y. Wang, D. An and Y. Wei, "Counting method for cultured fishes based on multi-modules and attention mechanism," *Aquacult. Eng.,* vol. 96, p. 102215, 2022.

[39] H. Idrees, I. Saleemi, C. Seibert y M. Shah, «Multi-source multi-scale counting in extremely dense crowd images,» de *2013 IEEE Conference on Computer Vision and Pattern Recognition*, 2013.

[40] *Papers with code - Crowd Counting.*

[41] H. Xiong, H. Lu, C. Liu, L. Liu, C. Shen y Z. Cao, «From open set to closed set: Supervised Spatial Divide-and-conquer for object counting,» 2020.

[42] C. Finn, X. Y. Tan, Y. Duan, T. Darrell, S. Levine y P. Abbeel, «Deep spatial autoencoders for visuomotor learning,» de *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016.

[43] B. Fortuner, «ML Cheatsheet - Loss Functions,» Read the Docs, [En línea]. Available: https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html. [Último acceso: 11 06 2022].

[44] D. Burdeiny, «Unofficial implementation (Pytorch) of S-DCNet and SS-DCNet,» Github, [En línea]. Available: https://github.com/dmburd/S-DCNet.

[45] D. Vazquez-Guevara, «Intro to Computer Vision and Computational Photography,» The University of California Berkeley, [En línea]. Available: https://inst.eecs.berkeley.edu/~cs194-26/fa20/upload/files/proj2/cs194-26-ado/. [Último acceso: 11 06 2022].

[46] Y. Li, X. Zhang y D. Chen, «CSRNet: Dilated convolutional neural networks for understanding the highly congested scenes,» de *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018.

[47] S. Ruder, «An overview of gradient descent optimization algorithms,» [En línea]. Available: https://ruder.io/optimizing-gradient-descent/index.html#fn5. [Último acceso: 11 06 2022].

[48] S. Kirsch, «Weight Decay in Neural Networks,» Programmathically: A Blog on Building Machine Learning Solutions, [En línea]. Available: https://programmathically.com/weight-decay-in-neural-networks/. [Último acceso: 11 06 2022].

Anexo 1. Presupuesto

A continuación, se muestran los costes en los que incurriría el proyecto realizado en caso de que su desarrollo se llevara a cabo con un propósito comercial y no educativo. Toda vez que, para el desarrollo actual no se ha incurrido en coste alguno, pues se ha contado con licencias de software educativas (Matlab y PyCharm) y suscripciones a servicios de computación en la nube (Azure) que contaban con convenios de colaboración con la universidad.

En la siguiente tabla se desglosa el presupuesto del desarrollo con las siguientes consideraciones:

- La duración del proyecto se ha calculado en base al número de horas empleadas (320) y considerando una dedicación de 40 horas semanales, resultando en 8 semanas.
- Para los cálculos de material informático se ha considerado un periodo de amortización de tres años.
- Para las licencias de software se ha calculado el valor porcentual por el tiempo de duración del proyecto.
- Se ha incluido un 10% de costes indirectos.
- No se ha incluido en el presupuesto posible concepto de beneficio.

Concepto	**Coste (EUR)**	**Unidades**	**Total (EUR)**
Portatil ASUS ZenBook UX535LH-BN033	1500	0.06	90
Licencia anual Matlab	800	0.17	136
Licencia anual Pycharm Pro	199	0.17	34
Uso de Azure Machine Learning	0.9	300	270
Mano de obra	25	320	8000
		Subtotal	8530
		Costes indirectos (10%)	*853*
		Total sin impuestos	*9383*
		IVA (21%)	*1970*
		Presupuesto total	**11.533**

Anexo 2. Manual de usuario

Generación del conjunto de datos

El script de MATLAB "createDataset.m" permite crear automáticamente los conjuntos de datos de entrenamiento (80%) y test (20%) y con un factor de reducción de las imágenes de 2.5. Solo es necesario ejecutarlo sobre el directorio donde se encuentra el conjunto de datos anotados con la aplicación Turbot del Grupo de Aplicaciones Multimedia y Acústica.

Generación de mapas de densidad

El *script* para generar los mapas de densidad es "gen_density_maps.py". Genera dos archivos, uno para el conjunto de entrenamiento y otro para el de test, en formato de archivo comprimido de Numpy (extensión npz). Los argumentos que recibe como entrada mediante línea de comandos son los siguientes:

- **dataset_rootdir**: ruta donde se encuentra el conjunto de datos sobre el que generar los mapas de densidad.
- **sigma_fixed**: Determina si se utiliza una desviación estándar fija para el *kernel* gaussiano o una estrategia de geometría adaptativa (ver apartado 4.2.2). Toma el valor "True" para usar sigma fija y "False" en caso contrario.
- **sigma**: si se ha optado por usar una desviación estándar fija para el *kernel* gaussiano se indica aquí el valor a usar.
- **knn**: en caso de usar geometría adaptative, número de vecinos más cercanos para calcular la distancia media.
- **max_knn_avg_dist**: en caso de usar geometría adaptative, valor máximo de la distancia media. En caso de que el valor calculado sea superior se sustituirá por el indicado.
- **sigma_coef**: en caso de usar geometría adaptative, valor del coeficiente por el que se multiplica la distancia media para obtener el valor de desviación estándar para el *kernel* gaussiano.
- **sqr_side**: Limitación en pixels de la amplitude del *kernel* gaussiano, de tal forma que solo afectará la función a [-sqr_side/2, +sqr_side/2] pixels en torno a cada punto etiquetado.
- **num_proc**: número de proceso en paralelo para ejecutar el algoritmo de generación.

Ejemplo de uso

```
gen_density_maps.py --dataset_rootdir=./Turbots/ --sigma_fixed True --sigma 4
```

Entrenamiento del modelo

El *script* para entrenar el modelo es "train.py". El script genera los archivos con el modelo en la carpeta "./checkpoints". Además, genera archivos compatibles con Tensorboard para analizar la evolución del entrenamiento. Los argumentos que recibe como entrada mediante línea de comandos son los siguientes:

- **dataset_rootdir**: ruta donde se encuentra el conjunto de datos sobre el que generar los mapas de densidad.

- **densmaps_gt_npz**: ruta a los archivos de mapas de densidad en la forma "densitymap_*.npz" donde el asterisco sustituye a las palabras *train* y *test*. Ver apartado 4.2.2 para información sobre el formato de los mapas de densidad.
- **num_intervals**: valor de C_{Max} para el contador.

Ejemplo de uso

```
train.py --dataset_rootdir=./Turbots/ --dataset_rootdir=./Turbots/densitymap_*.npz
--num_intervals 4
```

Además, el archivo "config_train_val_test.yaml" contiene otros parámetros sobre la configuración del modelo ver apartado 4.1.1 para más información sobre los mismos.

Evaluación del modelo

El *script* para entrenar el modelo es "evaluate.py". A partir de un modelo ya entrenado calculas las métricas de error MAE, RMSE y MAPE tanto para los conjuntos de entrenamiento como de test.

Los argumentos que recibe como entrada mediante línea de comandos son los siguientes:

- **dataset_rootdir**: ruta donde se encuentra el conjunto de datos sobre el que generar los mapas de densidad.
- **densmaps_gt_npz**: ruta a los archivos de mapas de densidad en la forma "densitymap_*.npz" donde el asterisco sustituye a las palabras *train* y *test*. Ver apartado 4.2.2 para información sobre el formato de los mapas de densidad.
- **num_intervals**: valor de C_{Max} para el contador.
- **trained_ckpt_for_inference**: ubicación del archivo con el modelo a evaluar.
- **visualize**: "True" para que genere imágenes con la visualización de la predicción para el conjunto de test. "False" en caso contrario y por defecto. Las imágenes se almacenan en el directorio "visualized_test_predictions".

Ejemplo de uso

```
evaluate.py --dataset_rootdir=./Turbots/ --dataset_rootdir=./Turbots/densitymap_*.npz
--num_intervals 4 --trained_ckpt_for_inference=./checkpoints/epoch_0200.pth --visualized True
```

Además, el archivo "config_train_val_test.yaml" contiene otros parámetros sobre la configuración del modelo ver apartado 4.1.1 para más información sobre los mismos.

Inferencia de nuevas imágenes

Existen dos *scripts* para la inferencia de nuevas imágenes con un modelo ya entrenado "inferenceCPU.py" y "inference.py". La única diferencia es que el primero utiliza solo la capacidad de computación de la CPU, y el segundo aprovecha la de la GPU mediante CUDA.

Los argumentos que recibe como entrada mediante línea de comandos son los siguientes:

- **num_intervals**: valor de C_{Max} para el contador.
- **trained_ckpt_for_inference**: ubicación del archivo con el modelo a evaluar.
- **imgs_for_inference_dir**: ruta al directorio que contiene las imágenes para la inferencia.

Ejemplo de uso

```
inference.py --num_intervals 4 --trained_ckpt_for_inference=./checkpoints/epoch_0200.pth --imgs_for_inference_dir=./images/
```

Además, el archivo "config_train_val_test.yaml" contiene otros parámetros sobre la configuración del modelo ver apartado 4.1.1 para más información sobre los mismos.