



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática



**TFG del Grado en Ingeniería
Informática**

**Detección de insectos
mediante Inteligencia
Artificial**



Presentado por Arturo Carretero Mateo
en Universidad de Burgos — 9 de junio de 2024
Tutor: Carlos Cambra Baseca



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática



D. Carlos Cambra Baseca, profesor del departamento de Digitalización, área de Ciencia de la Computación e Inteligencia Artificial.

Expone:

Que el alumno D.Arturo Carretero Mateo, con DNI 47313259M, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado Detección de Insectos.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 9 de junio de 2024

Vº. Bº. del Tutor:

D. Carlos Cambra Baseca

Resumen

Este Trabajo de Fin de Grado se enfoca en el desarrollo y aplicación de tecnologías avanzadas para la detección de insectos, con el objetivo de mejorar las estrategias de control de plagas en el ámbito agrícola.

El estudio se centra en la identificación automática de tres especies clave de insectos plaga: *Macrolophus pygmaeus*, la mosca blanca (*Bemisia tabaci*) y *Nesidiocoris tenuis*.

Dada la importancia de mantener las poblaciones de plagas bajo control así protegiendo los cultivos y asegurando la producción alimentaria, este estudio propone una solución innovadora que combina técnicas de *visión por computadora* e *inteligencia artificial (IA)* para la identificación y cuantificación automática de insectos plaga.

La metodología empleada se basa en el diseño y entrenamiento de modelos de aprendizaje profundo, específicamente redes neuronales convolucionales de máscaras (Mask R-CNN), mediante la segmentación de imágenes, obteniendo así características que nos permitirán llegar a una conclusión final. Se recopiló un conjunto de datos de imágenes de alta resolución a través de un sistema de cámaras, que mediante la colocación estratégica de trampas adhesivas por todo el cultivo, permitió agrupar a estos insectos.

Los resultados obtenidos demuestran la eficacia del sistema propuesto, logrando una precisión y exactitud significativas en la detección de plagas en comparación con los métodos convencionales. La implementación de esta tecnología no solo permite una detección temprana y precisa de las plagas, sino que también facilita la aplicación de tratamientos fitosanitarios de manera más dirigida y racional, reduciendo el uso de pesticidas y minimizando su impacto ambiental.

En conclusión, este TFG contribuye al campo del control de plagas agrícolas ofreciendo una herramienta potente para la detección automática de insectos, promoviendo así prácticas de agricultura más sostenibles y eficientes. Futuras investigaciones podrían explorar la integración de esta tecnología con sistemas de monitoreo en tiempo real y la expansión de su capacidad para cubrir una gama más amplia de especies plaga.

Descriptores

Modelo de aprendizaje profundo, vision por computadora, inteligencia artificial (IA), redes neuronales convolucionales (CNN) , segmentacion de imagenes.

Abstract

This Bachelor's Thesis focuses on the development and application of advanced technologies for insect detection, aiming to improve pest control strategies in the agricultural field.

The study centers on the automatic identification of three key insect pest species: *Macrolophus pygmaeus*, the whitefly (*Bemisia tabaci*), and *Nesidiocoris tenuis*.

Given the importance of keeping pest populations under control to protect crops and ensure food production, this study proposes an innovative solution that combines *computer vision techniques* and *artificial intelligence (AI)* for the automatic identification and quantification of pest insects.

The methodology employed is based on the design and training of deep learning models, specifically Mask R-CNN (Mask Region-Based Convolutional Neural Networks), through image segmentation, thereby obtaining characteristics that will allow us to reach a final conclusion. A set of high-resolution image data was collected through a camera system, which, by strategically placing adhesive traps throughout the crop, allowed for the grouping of these insects.

The results obtained demonstrate the effectiveness of the proposed system, achieving significant precision and accuracy in pest detection compared to conventional methods. The implementation of this technology not only allows for early and precise detection of pests but also facilitates the application of phyto-sanitary treatments in a more targeted and rational manner, reducing the use of pesticides and minimizing their environmental impact.

In conclusion, this Bachelor's Thesis contributes to the field of agricultural pest control by offering a powerful tool for the automatic detection of insects, thereby promoting more sustainable and efficient agricultural practices. Future research could explore the integration of this technology with real-time monitoring systems and the expansion of its capacity to cover a wider range of pest species.

Keywords

Deep learning model, computer vision, artificial intelligence (AI), convolutional neural networks (CNN), image segmentation.

Índice general

Índice general	v
Índice de figuras	vii
Índice de tablas	ix
1. Introducción	1
2. Objetivos del proyecto	5
2.1. Objetivos marcados por los requisitos del software	5
2.2. Objetivos de carácter técnico	6
3. Conceptos teóricos	7
3.1. Redes neuronales	8
3.2. Redes Neuronales Convolucionales - CNNs	16
3.3. R-CNN – Región basada en una Red Convolutacional	20
3.4. Fast R-CNN – Red Convolutacional basada en Regiones Rápidas	22
3.5. Mask R-CNN	24
3.6. Cuatro fases fundamentales	28
3.7. Herramientas y tecnologías	29
4. Técnicas y herramientas	31
5. Aspectos relevantes del desarrollo del proyecto	35
5.1. Datos	36
5.2. COCO - Common Objects in Context	39
5.3. Objetos	40
5.4. Arquitectura de la red neuronal	42

5.5. Optimizadores	45
5.6. Resultados	47
5.7. Mejoras	52
6. Trabajos relacionados	55
7. Conclusiones y Líneas de trabajo futuras	57
7.1. Líneas de trabajo futuras	59
Bibliografía	61

Índice de figuras

3.1.	red neuronal <i>feedforward (FNN)</i> simple de tres capas, compuesta por una capa de entrada, una capa oculta y una capa de salida.	8
3.2.	Comparativa del error según el número de épocas y neuronas [11]	10
3.3.	<i>Función de activación sigmoidea</i>	12
3.4.	<i>Función de activación Tanh</i>	13
3.5.	<i>Función de activación ReLU</i>	13
3.6.	Gráfica comparativa precisión y perdida	15
3.7.	Secuencia de una CNN paso a paso [36]	16
3.8.	Secuencia de entrada RGB en CNN [33]	17
3.9.	Aplicación de convolución [15]	17
3.10.	Demostración gráfica de Max Pooling y Average Pooling	18
3.11.	Capa flatten en la estructura de red CNN [35]	19
3.12.	Descripción general de la estructura de red R-CNN [19]	20
3.13.	Representación gráfica del algoritmo Greedy en búsqueda selectiva [16]	21
3.14.	A la izquierda, la Red de Propuestas de Región (RPN). A la derecha, se presentan ejemplos de detecciones utilizando propuestas RPN [30]	21
3.15.	Modelo Fast R-CNN [17]	22
3.16.	Comparativas entre <i>Image recognition, Semantic segmentation, Object detection y Instance segmentation</i> [38]	24
3.17.	Primera máscara sobre un objeto a detectar	25
3.18.	<i>Bounding boxes</i> y máscaras generadas por un modelo Mask R-CNN [27]	25
3.19.	Pérdida al hacer <i>Max Pooling</i>	26
3.20.	Ejemplo de stride utilizando interpolación bilineal [20]	26
3.21.	Arquitectura Mask R-CNN	27

4.1. Proceso de detección de objetos utilizado por la arquitectura de red neuronal YOLO [29]	33
5.1. Ejemplo de la imagen 001.jpg y su etiquetado 001.xml	37
5.2. Ejemplo de una imagen de entrada a Mask R-CNN	41
5.3. Ejemplo de una imagen tras haber realizado predicciones	41
5.4. Figura correspondiente a los atributos de 2 insectos detectados .	42
5.5. Figura correspondiente a un bloque residual [41]	43
5.6. Pérdida de entrenamiento por paso (izquierda) y pérdida de validación por época (derecha) después de 600 pasos y 12 épocas usando SGD.	48
5.7. Pérdida de entrenamiento por paso (izquierda) y pérdida de validación por época (derecha) después de 7000 pasos y 50 épocas usando SGD.	48
5.8. Primer resultado de la aplicación usando SGD.	49
5.9. Segundo resultado de la aplicación usando SGD.	49
5.10. Segundo resultado de la aplicación usando RMSprop.	50
5.11. Segundo resultado de la aplicación usando Adam.	51
7.1. Posible sistema SIG	60

Índice de tablas

3.1. Herramientas y tecnologías utilizadas en cada parte del proyecto 29

1. Introducción

En los últimos años han aparecido las primeras aplicaciones de la informática en la agricultura. Estas aplicaciones resuelven tareas repetitivas, mecanicistas o de manejo de grandes volúmenes de información. En el sector industrial, que tiene menos factores incontrolados y aleatorios que la agricultura, se vienen aplicando cada vez más los métodos CAD (Computer-Aided Design), CAM (Computer-Aided Manufacture) y la robótica.[8]

En la agricultura no pueden aplicarse con tanta rapidez estos métodos tan deterministas por trabajar con seres vivos y no ser posible el control de todas las variables climáticas, ecológicas ni económicas. Es necesario recurrir a la rama más moderna de la informática que es la Inteligencia Artificial. Distintos trabajos han analizado con cierto detalle las aplicaciones de estas técnicas a la agricultura. La Inteligencia Artificial (IA) tiene dos campos de aplicación en la agricultura: la robótica y la construcción de sistemas expertos.[8]

Un sistema experto se define en el documento como un tipo de sistema informático que emula la capacidad de toma de decisiones de un experto humano. Estos sistemas son capaces de resolver problemas complejos mediante el uso de bases de conocimientos especializados y una forma de razonamiento que imita la lógica humana. [3]

En nuestro caso utilizaremos un sistema experto el cual utiliza la segmentación de imágenes para la obtención de características e información y poder tomar decisiones a partir de esta información obtenida.

En nuestro caso utilizaremos estas técnicas para la detección y control de los insectos: *Trialeurodes vaporariorum* y *Bemisia tabaci* los cuales están listados entre las diez plagas más problemáticas en cultivos de vegetales en invernadero.[26]

Podremos interactuar con un modelo ya preentrenado o bien entrenar uno nuevo mediante la aplicación web, mostrándose en esta los resultados producidos durante la detección de dichos insectos.

2. Objetivos del proyecto

Este apéndice detalla de manera clara y breve los propósitos buscados a través de la ejecución del proyecto. Se diferencian dos tipos de metas: objetivos marcados por los requisitos del software y requisitos funcionales y no funcionales.

2.1. Objetivos marcados por los requisitos del software

Requisitos funcionales y no funcionales

El sistema está diseñado para cumplir con requisitos funcionales específicos que permiten a los usuarios cargar imágenes para su análisis y procesamiento. Esto incluye la carga de imágenes tanto para entrenamiento como para predicción, el preprocesamiento de imágenes y la segmentación utilizando la arquitectura Mask R-CNN. Además, proporciona una interfaz de usuario intuitiva y una visualización clara de los resultados de la detección.

Educación y usabilidad

Un enfoque clave del sistema es educar a los usuarios sobre el proceso de segmentación de imágenes y redes neuronales. Por ende, el sistema incluye una sección educativa y está diseñado para ser intuitivo, asegurando que todos los usuarios puedan aprender a usar la aplicación web en menos de 10 minutos, permitiendo también entrenar sus propios modelos sin un conocimiento amplio de programación o redes neuronales.

2.2. Objetivos de carácter técnico

Programación en Python

Proporcionar conocimientos al lector sobre el lenguaje de programación Python, además del manejo del IDE Spyder, permitiendo gestionar y crear proyectos.

Desarrollo de modelos y uso de datos

El proyecto tiene como objetivo desarrollar un modelo de detección de insectos basado en el uso de redes neuronales y la arquitectura Mask R-CNN, utilizando un conjunto de datos del repositorio 4TU.ResearchData. Los datos utilizados son imágenes etiquetadas de insectos recolectados en invernaderos comerciales, lo cual es esencial para entrenar algoritmos de aprendizaje profundo en tareas de reconocimiento y clasificación de insectos.

Antes y después del entrenamiento

Se utilizan pesos preentrenados del conjunto de datos MS COCO, que sirven como punto de partida para el modelo Mask R-CNN. Después del entrenamiento, el modelo puede ser ajustado y mejorado en función de los resultados obtenidos para garantizar su eficacia en la detección de insectos.

Metodología y planificación

Para la estructuración del trabajo, se ha elegido la metodología Kanban por su eficiencia en la representación mediante tableros y la mejora continua del flujo de trabajo. Se han definido un total de ocho epics para la planificación y ejecución del proyecto, que incluyen desde la configuración del entorno de desarrollo hasta la creación de la interfaz web y la documentación final del proyecto.

Viabilidad y expectativas

La viabilidad económica del proyecto se ha evaluado y categorizado en costes directos, indirectos, fijos y variables, asegurando así que el proyecto es económicamente factible y sostenible.

3. Conceptos teóricos

En esta sección, abordaremos de manera teórica el proceso de detección y clasificación de insectos, profundizando en la *instance segmentation*, la cual utiliza Mask R-CNN, además de los métodos previos que han ido evolucionando hasta llegar a este destacado. Estos métodos son:

1. RNA: Redes Neuronales Artificiales
2. CNN: Red Neuronal Convolutacional
3. R-CNN: Región Basada en una Red Convolutacional
4. Faster R-CNN: Faster Region-Based Convolutional Network
5. Mask R-CNN: Mask (Faster) Region-Based Convolutional Network (RoIAlign)

Además posteriormente de los diferentes procesos desde la adquisición inicial de las imágenes hasta la identificación de los insectos, detallando cada fase crucial que compone este sistema automatizado. Siguiendo con el siguiente apéndice, hablando así sobre las herramientas que han hecho posible que este proceso se realice de manera efectiva y eficiente.

3.1. Redes neuronales

Una red neuronal artificial es una simulación por computadora que intenta modelar los procesos del cerebro humano para imitar la forma en que aprende. [4].

Las redes neuronales artificiales se estructuran a partir de múltiples nodos de procesamiento, conocidos como neuronas, interrelacionados entre sí. Estos nodos colaboran de manera distribuida para procesar la información de entrada, aprendiendo de manera conjunta para mejorar la precisión de los resultados obtenidos.

La entrada generalmente se carga en forma de un vector multidimensional en la capa de entrada, que luego se distribuye a las capas ocultas. Las capas ocultas toman decisiones a partir de la información recibida de la capa anterior y sopesan cómo un cambio estocástico dentro de sí mismas perjudica o mejora el resultado final, proceso conocido como aprendizaje.[5]

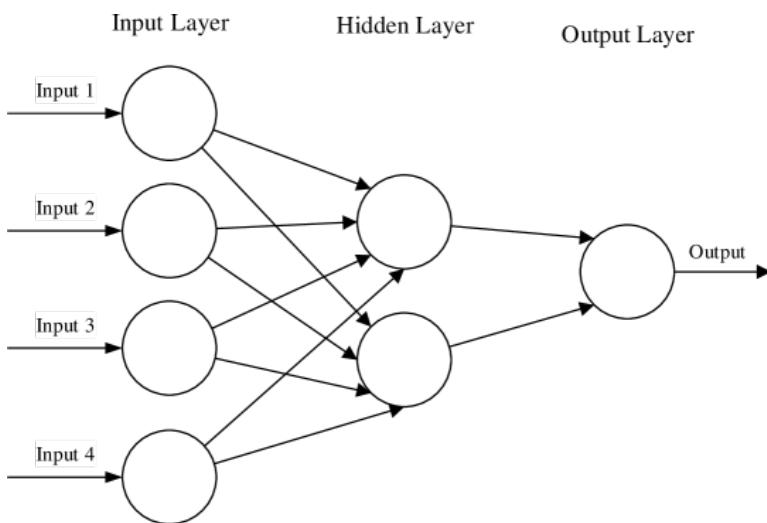


Figura 3.1: red neuronal *feedforward (FNN)* simple de tres capas, compuesta por una capa de entrada, una capa oculta y una capa de salida. [5]

Cada nodo individual se puede considerar como su propio modelo de regresión lineal, formado por datos de entrada x_i , ponderaciones w_i , un sesgo (o umbral) y una salida. La fórmula sería similar a la siguiente:

$$\sum w_i x_i + \text{sesgo} = w_1 x_1 + w_2 x_2 + w_3 x_3 + \text{sesgo}$$

La salida se determina de la siguiente manera:

$$\text{salida} = f(x) = \begin{cases} 1 & \text{if } \sum w_i x_i + b \geq 0 \\ 0 & \text{if } \sum w_i x_i + b < 0 \end{cases}$$

Épocas

Una época significa entrenar la red neuronal con todos los datos de entrenamiento durante un ciclo. En una época, utilizamos todos los datos exactamente una vez.^[6] En ese momento del proceso, el modelo realiza ajustes a sus parámetros internos tomando como referencia la diferencia entre los valores predichos y los reales.

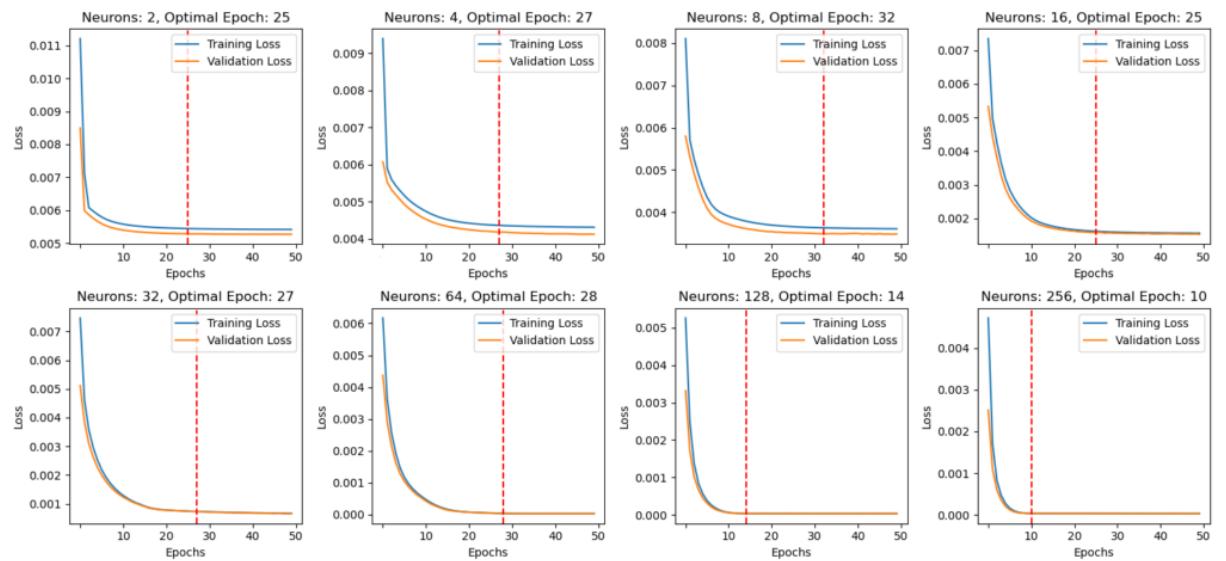


Figura 3.2: Comparativa del error según el número de épocas y neuronas [11]

Determinar la cantidad óptima de épocas es esencial por varias razones críticas:

- **Aprendizaje Incompleto:** Un número insuficiente de épocas podría no ser suficiente para que el modelo capture y aprenda efectivamente los patrones en el conjunto de datos de entrenamiento.
- **Sobreentrenamiento:** Un número excesivo de épocas puede llevar al modelo a memorizar los datos de entrenamiento, afectando negativamente su habilidad para generalizar a nuevos datos no observados.
- **Gasto Ineficiente de Recursos:** Asignar más épocas de las necesarias resulta en el uso innecesario de recursos computacionales, lo cual puede alargar el tiempo de entrenamiento sin proporcionar mejoras significativas en el rendimiento del modelo.

Tasa de aprendizaje

La tasa de aprendizaje es un hiperparámetro que controla cuánto cambiar el modelo en respuesta al error estimado cada vez que se actualizan los pesos del modelo. Elegir la tasa de aprendizaje es un desafío, ya que un valor demasiado pequeño puede resultar en un proceso de entrenamiento largo que podría atascarse, mientras que un valor demasiado grande puede resultar en aprender un conjunto de pesos subóptimo demasiado rápido o un proceso de entrenamiento inestable.^[9] Aquí se presentan algunas estrategias para configurarla:

- **Velocidad de aprendizaje estática:** Se mantiene constante a lo largo del entrenamiento, lo que puede resultar en un entrenamiento subóptimo.
- **Disminución de la Velocidad de Aprendizaje:** Reduce la velocidad de aprendizaje a lo largo del tiempo, lo cual puede ayudar en la convergencia del modelo.
- **Velocidad de aprendizaje adaptativa:** Métodos como Adagrad, RMSprop y Adam ajustan la velocidad individualmente por parámetro, basándose en la historia de los gradientes.
- **Programación de velocidad de aprendizaje:** Ajusta la velocidad según un cronograma establecido o en respuesta al rendimiento del modelo durante la validación.
- **Velocidad de aprendizaje cíclica:** Oscila entre dos valores, permitiendo tanto la exploración del espacio de pesos como la convergencia durante el entrenamiento.

La selección adecuada y el ajuste de la velocidad de aprendizaje son cruciales para la eficacia del modelo, su rapidez en converger y su habilidad para generalizar tras el entrenamiento.

Funciones de activación

Una vez que se determina una capa de entrada, se asignan ponderaciones. [23] A la hora de determinar la activación de una propia neurona esta utilizará funciones de activación. Hablaremos principalmente de las funciones de activación más utilizadas a la hora de la clasificación , siendo un resultado de positivo(activación) o negativo (no activación). Se destacan las siguientes:

- **Función de activación sigmoidea.** La función sigmoidea se usaba tradicionalmente para problemas de clasificación binaria (sigue la línea de "si $x \leq 0,5$, $y = 0$, si no, $y = 1$ "). Pero tiende a causar un problema de gradientes que desaparecen y, si los valores están demasiado cerca de 0 o +1, la curva o el gradiente es casi plano y, por lo tanto, el aprendizaje sería demasiado lento. [2]

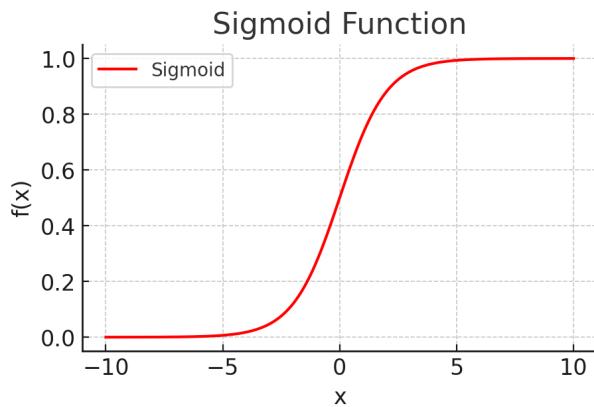


Figura 3.3: Función de activación sigmoidea

- **Función de activación Tanh.** Es diferente del sigmoide en el sentido de que está centrado en cero y, por lo tanto, restringe los valores de entrada entre -1 y +1. Es incluso más costoso desde el punto de vista computacional que sigmoide, ya que implica muchas operaciones matemáticas complejas, que deben realizarse para cada entrada e iteración, repetidamente.[2]

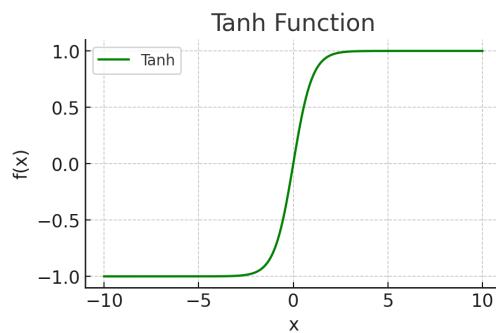


Figura 3.4: Función de activación Tanh

- **Función de activación ReLU.** ReLU es una función de activación no lineal famosa y ampliamente utilizada, que significa Unidad Lineal Rectificada (va más o menos como “si $x \leq 0$, $y = 0$ sino $y = 1$ ”). Por tanto, sólo se activa cuando los valores son positivos. ReLU es menos costoso computacionalmente que tanh y sigmoide porque implica operaciones matemáticas más simples. [2]

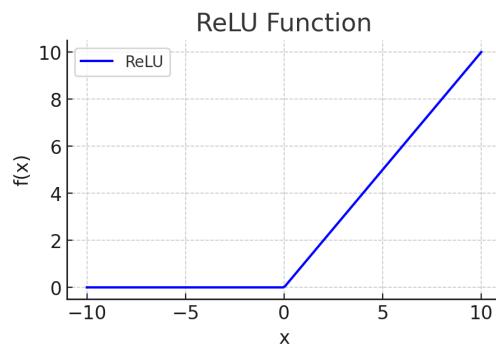


Figura 3.5: Función de activación ReLU

Función de perdida

La función de pérdida ayuda a determinar la eficacia con la que el algoritmo modela el conjunto de datos presentado. De manera similar, la pérdida es la medida que tiene el modelo de previsibilidad, los resultados esperados. Las pérdidas generalmente pueden clasificarse en dos categorías amplias relacionadas con problemas del mundo real: clasificación y regresión. Debemos predecir la probabilidad para cada clase a la que se refiere el problema. Sin embargo, en la regresión tenemos la tarea de pronosticar un valor constante para un grupo específico de características independientes.^[39] Viene expresada por la fórmula:

$$CE = - \sum_{i=1}^C t_i \log(s_i)$$

En resumidas evalúa los valores de salida esperados y los valores predichos; Evalúa qué tan efectivamente la red neuronal representa los datos de entrenamiento. Durante el proceso de entrenamiento, nuestro propósito es reducir al mínimo esta discrepancia entre las predicciones y las metas establecidas modificando así sus respectivos pesos.

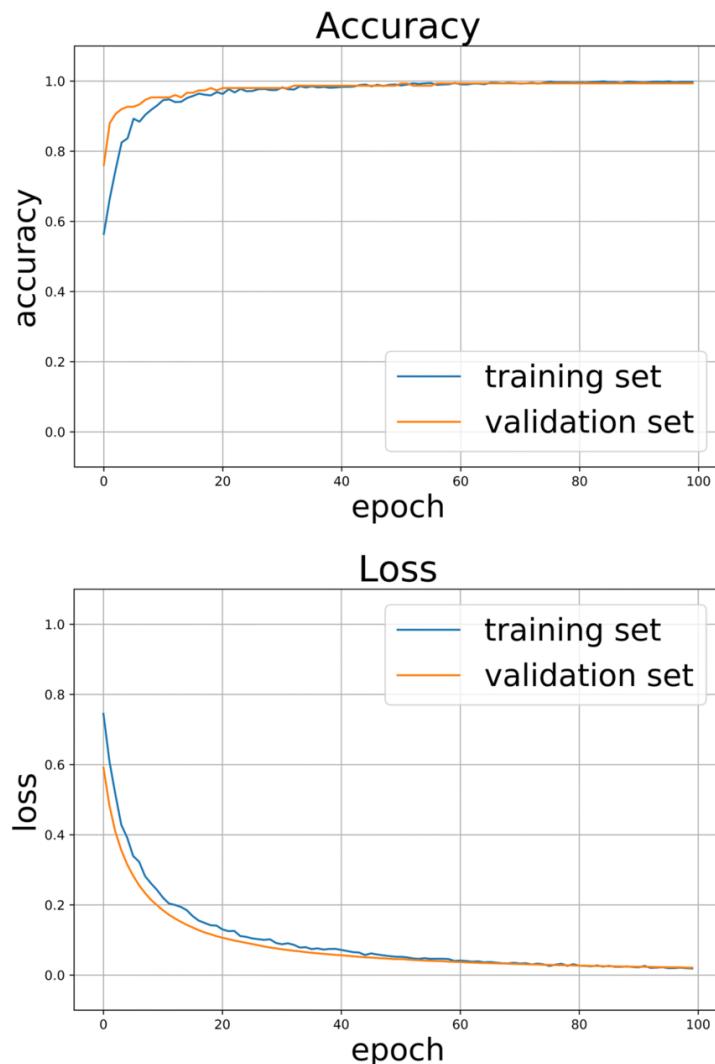


Figura 3.6: Gráfica comparativa precisión y perdida

[45]

Accuracy siendo el acierto del modelo en las predicciones, como podemos ver en la gráfica la comparativa entre el acierto con el conjunto de validación y prueba a medida que aumentan las épocas este se hace mas preciso mostrando mejores resultados con un error menor.

3.2. Redes Neuronales Convolucionales - CNNs

Una Red Neuronal Convolucional (CNN), también conocida como ConvNet, es un tipo especializado de algoritmo de aprendizaje profundo diseñado principalmente para tareas que requieren el reconocimiento de objetos, incluida la clasificación, detección y segmentación de imágenes. Las CNN se emplean en una variedad de escenarios prácticos, como vehículos autónomos, sistemas de cámaras de seguridad y otros.[13]

En la estrategia inicial de clasificación de imágenes, se solía emplear el uso directo de los píxeles para llevar a cabo la clasificación. Por ejemplo, para diferenciar entre diversas razas de perros, se podría utilizar la información de color a través del histograma de colores de los píxeles en la imagen, así como la forma de las orejas mediante la detección de bordes. A pesar de que este método puede arrojar resultados aceptables en situaciones simples, la extracción de características más complejas permite abordar desafíos más complicados.

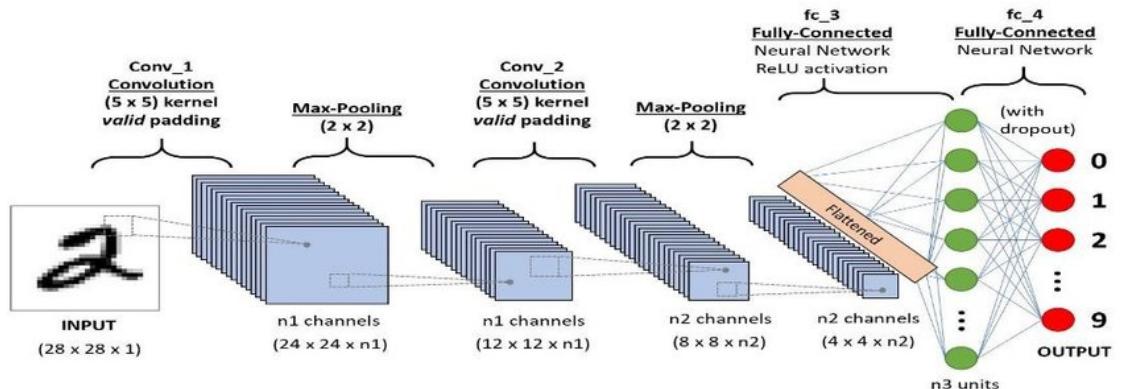


Figura 3.7: Secuencia de una CNN paso a paso [36]

Como podemos ver en la figura 3.7, corresponde a una CNN capaz de determinar, a partir de una imagen o dibujo de un número, qué número es este, teniendo unas posibles respuestas del 0 al 9 en la salida.

En nuestro caso, tendremos como entrada una imagen RGB, por lo tanto, una imagen con 3 matrices, en las que cada una de estas representará valores del 0 al 255 para las intensidades de los colores rojo, verde y azul (RGB). De la siguiente forma:

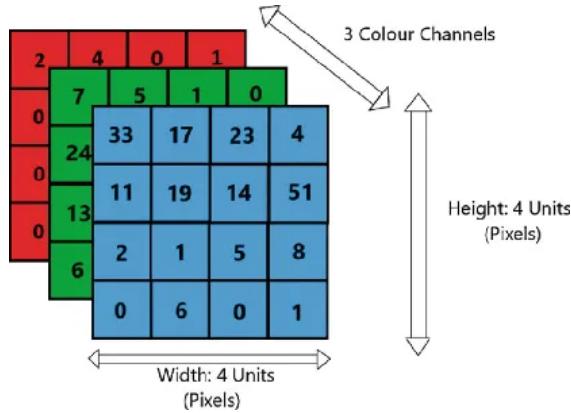


Figura 3.8: Secuencia de entrada RGB en CNN [33]

En cuanto a las CNN, es muy importante ser conocedor de dos acciones fundamentales que realizan estas: la aplicación de la *convolución* y el *pooling*.

- **Convolución:** Obtendrá información relevante a partir de la matriz de entrada multiplicando por los valores del filtro K .

Como podemos ver en la figura 3.10, el kernel es de 3x3 mientras que

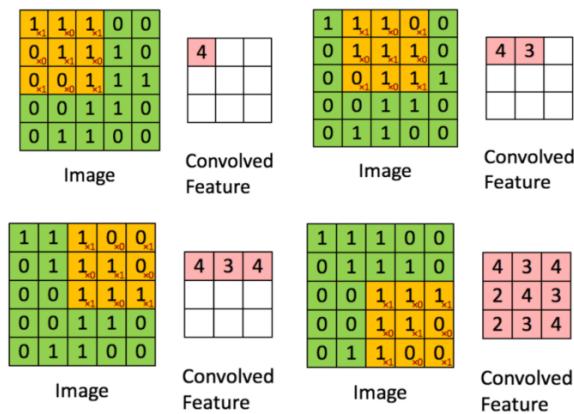


Figura 3.9: Aplicación de convolución [15]

la matriz de entrada es de 5×5 . A la hora de moverse, el kernel por la matriz de entrada se moverá en función del *stride* (longitud de paso), en este caso siendo de 1, moviéndose un total de nueve veces a la hora de recorrer la matriz de entrada. Con los siguientes valores del filtro K :

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

- **Pooling:** A partir del kernel podremos hacer dos tipos de agrupaciones:

1. **Average Pooling**: Agrupando conjuntos de celdas de la matriz, es capaz de calcular el promedio para cada uno de los conjuntos de celdas determinados, obteniendo así un número reducido conformado por el promedio de estas.
2. **Max Pooling**: Similar al Average Pooling, pero seleccionando las celdas mayores de los grupos de la matriz.

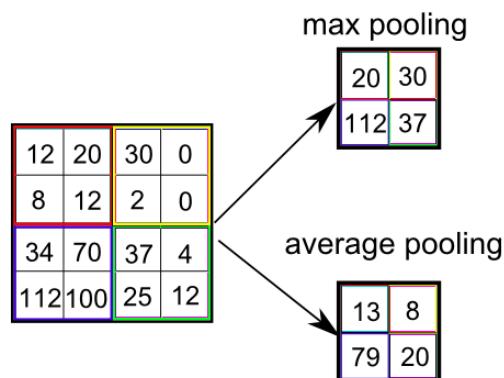


Figura 3.10: Demostración gráfica de Max Pooling y Average Pooling

Como podemos ver en la figura 3.10, el tamaño de la formación de los grupos estará definido por el valor de *stride*, que en este caso tendrá un valor de 2.

Convertida nuestra imagen de entrada a una forma adecuada para nuestro perceptrón multinivel, aplanaremos la imagen en un vector de columna. La salida aplanada se envía a una red neuronal de retroalimentación y se aplica retropropagación en cada iteración del entrenamiento. A lo largo de una serie de épocas, el modelo es capaz de distinguir entre características dominantes y ciertas características de bajo nivel.[37]

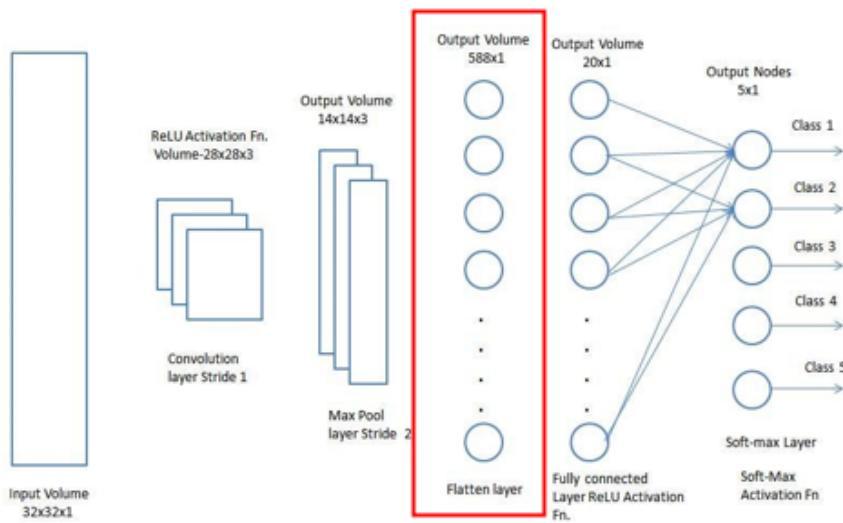


Figura 3.11: Capa flatten en la estructura de red CNN [35]

A partir de las características obtenidas en las diferentes capas de la red CNN, será capaz de obtener mediante la red neuronal el resultado final, activándose la neurona oportuna mediante su función de activación, como puede ser la función ReLU (1 activa, 0 desactivada), teniendo tantas neuronas en la última capa como clases a detectar en la imagen.

3.3. R-CNN – Región basada en una Red Convolucional

A diferencia de las CNN convencionales, las R-CNN utilizan la identificación de regiones, agrupando así las características en una misma región y luego extrayendo información de estas. Por lo tanto, se utiliza una CNN para cada una de las regiones propuestas.

Las fases son las siguientes:

1. Obtener la imagen de entrada.
2. Generar regiones propuestas en la imagen.
3. Aplicar una red CNN para la obtención de características.
4. Clasificar las regiones mediante una RNA y la activación de las neuronas.

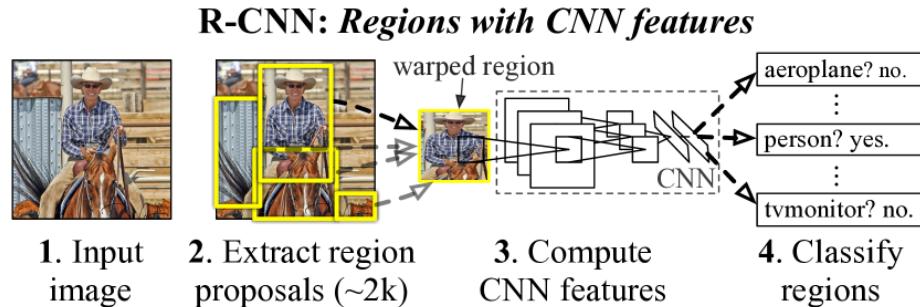


Figura 3.12: Descripción general de la estructura de red R-CNN [19]

3.3. R-CNN – REGIÓN BASADA EN UNA RED CONVOLUCIONAL21

Para la generación de las regiones, existen varios algoritmos que permiten la definición de estas regiones propuestas:

- **Algoritmo de búsqueda selectiva:** Este algoritmo combina recursivamente las regiones similares más pequeñas en otras más grandes, utilizando el algoritmo Greedy para combinar regiones similares y crear regiones más grandes.[16]

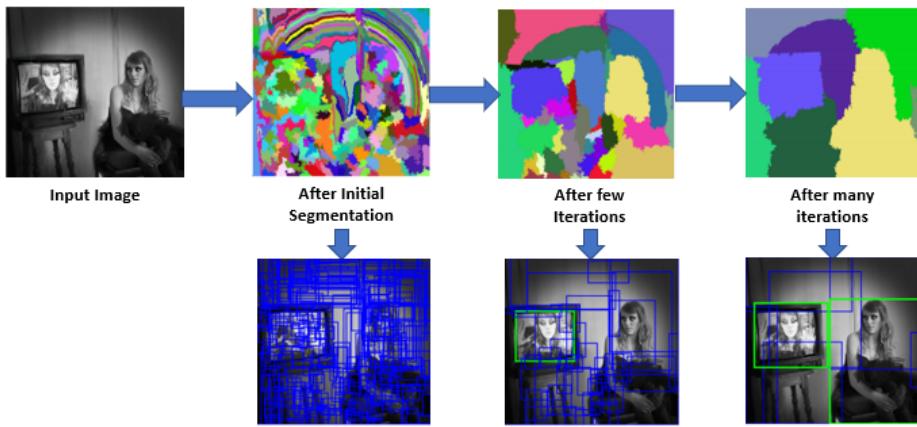


Figura 3.13: Representación gráfica del algoritmo Greedy en búsqueda selectiva [16]

- **Region Proposal Network:** Utiliza cajas de anclaje como referencias en múltiples escalas y relaciones de aspecto, lo que permite predecir eficientemente propuestas de región con un amplio rango de escalas y proporciones.[30]

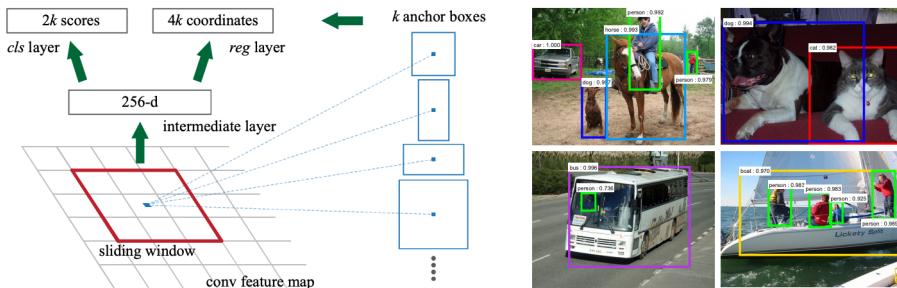


Figura 3.14: A la izquierda, la Red de Propuestas de Región (RPN). A la derecha, se presentan ejemplos de detecciones utilizando propuestas RPN [30]

3.4. Fast R-CNN – Red Convolucional basada en Regiones Rápidas

Tras la aparición de R-CNN, se lograron mejoras que dieron lugar al modelo Fast R-CNN. En lugar de extraer características de la CNN de forma independiente para cada región de interés, Fast R-CNN las agrega en un único paso hacia adelante sobre la imagen; es decir, las regiones de interés de la misma imagen comparten cálculo y memoria en los pasos hacia adelante y hacia atrás.[18]

El sistema está compuesto por dos módulos. El primer módulo es una red convolucional profunda que propone regiones, y el segundo módulo es el detector Fast R-CNN que utiliza las regiones propuestas. Todo el sistema es una única red unificada para la detección de objetos. Utilizando la terminología recientemente popular de redes neuronales con mecanismos de “atención”, el módulo RPN le indica al módulo Fast R-CNN dónde mirar.

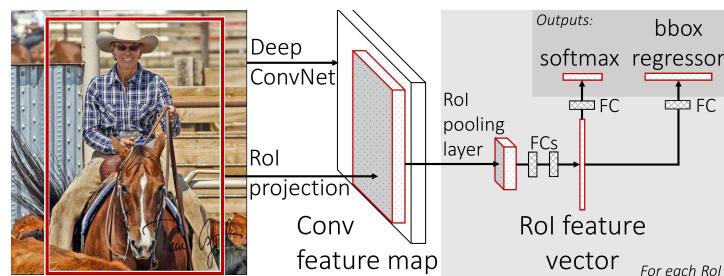


Figura 3.15: Modelo Fast R-CNN [17]

Explicación de los elementos para entender su mejor funcionamiento:

- **Arquitectura ConvNet profunda:** La esencia de este enfoque radica en el uso de una red convolucional de múltiples capas que procesa de manera exhaustiva la imagen entera, destilando un conjunto de mapas de características. Estos mapas actúan como representaciones detalladas que capturan los atributos visuales cruciales dispersos por toda la imagen.
- **Proyección de Región de Interés (RoI):** Cada región demarcada como de interés es mapeada hacia el espacio de características convolucional previamente obtenido. La ubicación precisa de cada RoI es esencial y se determina dentro del contexto del mapa de características.
- **Capa de pooling RoI:** Posterior a la proyección, sigue la aplicación de la capa de pooling sobre las RoIs. El propósito de esta operación es condensar las dimensiones de los atributos de las RoIs a una escala uniforme, simplificando el tratamiento subsecuente. Este procedimiento se replica para cada RoI identificada en el mapa.
- **Vector de características RoI:** Los atributos de cada RoI, una vez comprimidos, se transforman en un vector único de características. Este vector funciona como un compendio eficiente de la información visual inherente a cada región de interés seleccionada.
- **Salidas del modelo:** El flujo de información se conduce hacia una serie de capas densamente conectadas (FC), culminando en la producción de dos resultados por cada RoI:
 - **Regresor de cajas delimitadoras:** Funciona ajustando los contornos de las cajas delimitadoras proyectadas para coincidir con la localización exacta de objetos dentro de la RoI.
 - **Clasificador Softmax:** Este clasificador estima la probabilidad de que cada RoI contenga un objeto y, en caso afirmativo, procede con su clasificación.

3.5. Mask R-CNN

Una vez comprendidos los términos y modelos anteriores, podemos dar paso al aprendizaje sobre Mask R-CNN, la cual es la que utilizará este proyecto.

Mask R-CNN es una arquitectura avanzada de red neuronal diseñada para llevar a cabo la identificación y delimitación precisa de elementos individuales en imágenes, a través del proceso conocido como segmentación de instancias. Esta técnica es esencial para diferenciar y segmentar múltiples elementos de una misma categoría dentro de una imagen, proporcionando una comprensión detallada de su contexto y relaciones espaciales.[40]

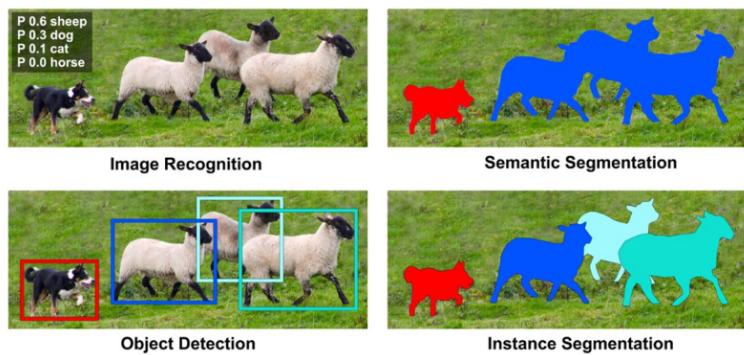


Figura 3.16: Comparativas entre *Image recognition*, *Semantic segmentation*, *Object detection* y *Instance segmentation* [38]

La Figura 3.1 muestra las distinciones entre la detección de objetos, la segmentación semántica y la segmentación de instancias. La segmentación de instancias fusiona los dos últimos enfoques para ofrecer una identificación y separación minuciosa de cada elemento individual.

Básicamente, Mask R-CNN es una extensión de Fast R-CNN pero prediciendo una máscara binaria para cada clase de forma independiente, sin competencia entre clases, basándose en la rama de clasificación RoI de la red para predecir la categoría.

Máscaras

Al recibir una imagen como entrada, se generarán un *bounding box* y una *máscara* dentro de este *bounding box* para cada una de las regiones propuestas (*RPN*).



Figura 3.17: Primera máscara sobre un objeto a detectar

La máscara generada por primera vez tendrá un valor de 0 conformando todo el espacio del *bounding box* hasta el límite de este, como podemos ver en la Figura 3.17.

Una vez que el modelo es entrenado, será capaz de establecer la máscara a los bordes de la figura a detectar, dando un valor de 0 a la forma en la máscara y de 1 al resto del espacio entre la figura y el *bounding box*.

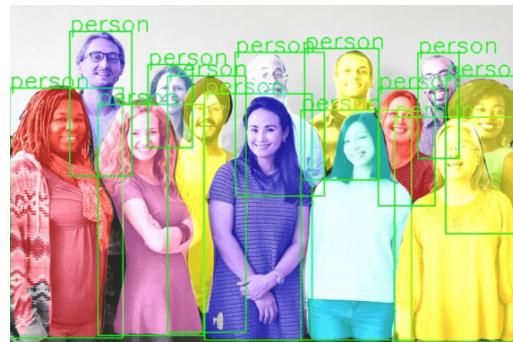


Figura 3.18: *Bounding boxes* y máscaras generadas por un modelo Mask R-CNN [27]

Como podemos ver en la figura, el modelo guardará el *bounding box* y la máscara de cada uno de los elementos detectados, en este caso personas.

RoIAlign

RoIAlign es particularmente beneficioso porque evita la cuantización de las coordenadas de la región de interés durante el proceso de pooling. Como podemos ver en el siguiente caso, perderíamos información al hacer el *Pooling*.

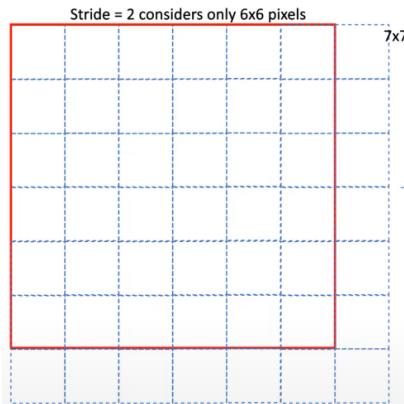


Figura 3.19: Pérdida al hacer *Max Pooling*

Como podemos ver en el siguiente caso, se toman cuatro muestras dentro de cada bin utilizando interpolación bilineal, lo que resulta en un alineamiento exacto y conservación de detalles. Esto se traduce en una segmentación más precisa a nivel de píxel.

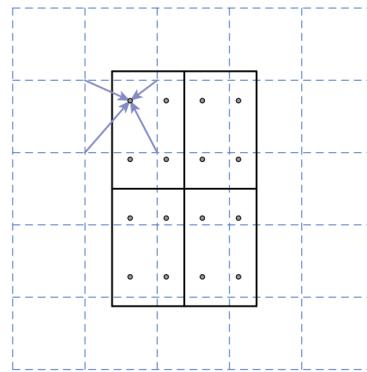


Figura 3.20: Ejemplo de stride de utilizando interpolación bilineal [20]

Arquitectura

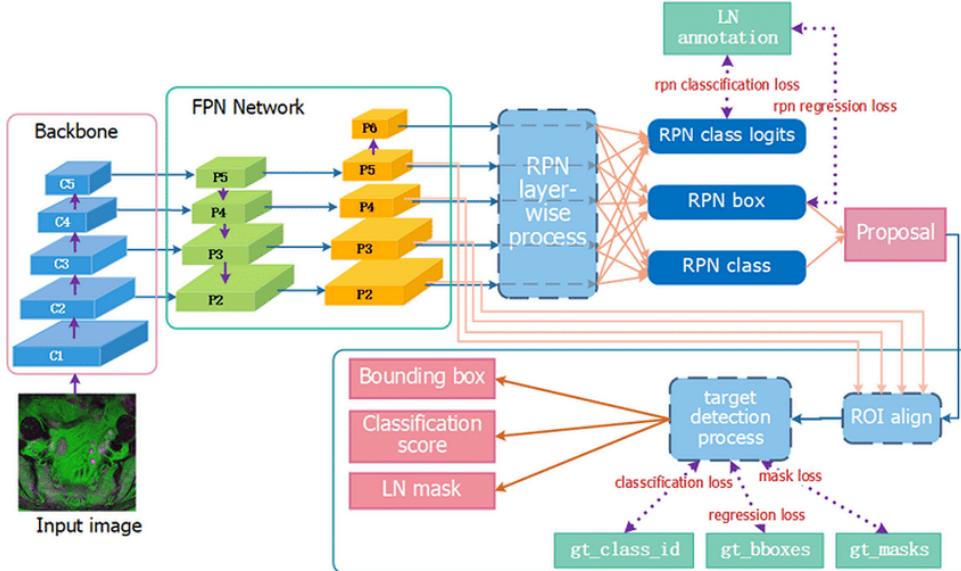


Figura 3.21: Arquitectura Mask R-CNN

Esta arquitectura de Mask R-CNN se compone de varias partes clave:

1. **Backbone:** Es la base de la red neuronal que procesa la imagen de entrada y extrae las características.
2. **FPN Network:** Toma las características extraídas del *backbone* y las procesa a diferentes escalas para capturar detalles a varios niveles.
3. **RPN Layer-wise Process:** La Red de Propuestas de Región (RPN) utiliza las características de FPN para proponer candidatos a objetos que podrían estar presentes en la imagen.
4. **Proposal:** Las propuestas generadas se pasan para su procesamiento.
5. **RoI Align:** Esta etapa alinea precisamente las regiones de interés con las características de la imagen.
6. **Bounding Box, Classification Score, LN Mask:** Estos son los outputs finales que incluyen las cajas delimitadoras para cada detección, los puntajes de clasificación para cada objeto detectado y las máscaras que segmentan los objetos a nivel de píxel.

3.6. Cuatro fases fundamentales

1. **Pre-procesamiento:** Esta fase involucra la preparación inicial de las imágenes recolectadas(dataset). Se realizan ajustes y mejoras, como la normalización del tamaño de las imágenes.
2. **Codificación:** En esta fase, las imágenes son transformadas de su forma original a un formato que el modelo de IA puede procesar eficientemente. Esto generalmente implica la conversión de imágenes a un conjunto de características o tensores que sirven como entrada al modelo de redes neuronales convolucionales (CNN), es importante la redimensión de las imágenes de entrada a un tamaño "común" siendo el mismo para todas ellas.
3. **Detección y Clasificación:** Aquí es donde el modelo de IA entra en acción. Utilizando los datos procesados, el modelo realiza la identificación y clasificación de los insectos presentes en las imágenes. Esta fase es crucial y es el núcleo del proceso de detección de insectos.
4. **Post-procesamiento:** Finalmente, los resultados generados por el modelo de IA pueden necesitar ajustes adicionales para mejorar la clasificación de las especies detectadas. En esta etapa, se corrigen posibles errores y se perfeccionan los resultados antes de su presentación final.

3.7. Herramientas y tecnologías

A continuación se muestra una tabla con las herramientas en las diferentes partes del proyecto AppPython ,API REST, BD y Memoria

Tabla 3.1: Herramientas y tecnologías utilizadas en cada parte del proyecto

Herramientas	App	Python	Memoria
HTML		X	
CSS		X	
FLASK		X	
Python		X	
JavaScript		X	
Socket.io		X	
TensorFlow		X	
Keras		X	
JSON		X	
GitHub	X		X
GitBash		X	X
Overleaf			X
TEXMaker			X
Batch Scripting	X		
Conda		X	
Spyder		X	

4. Técnicas y herramientas

Técnicas, metodologías y herramientas de desarrollo

Dentro del espectro de metodologías a disposición, sobresalen las Redes Neuronales Convolucionales (CNN), Mask R-CNN, YOLO (You Only Look Once) y SSD (Single Shot MultiBox Detector), cada una distinguiéndose por sus características específicas y sus respectivas áreas de aplicación y restricciones. Este apéndice procede a realizar un análisis comparativo de dichas técnicas, culminando en la explicación de por qué se prefirió Mask R-CNN para este estudio en particular.

- **CNN:** En el ámbito de la segmentación de imágenes, han cumplido con los propósitos para los que se utilizaban; sin embargo, en determinados casos se las considera “antiquadas” debido a una serie de razones como:
 1. Gran cantidad de datos: Para un buen funcionamiento de las CNN necesitaremos una gran cantidad de datos para las diferentes fases del entrenamiento y validación. Esto, en algunos casos, puede suponer un gran coste de recursos y tiempo.
 2. Alta complejidad computacional: Es común una gran cantidad de capas e interconexiones entre estas, lo que implica una alta complejidad computacional, limitándose a equipos con grandes recursos de procesamiento, no disponibles para dispositivos como sistemas embebidos.
 3. Dificultades para adaptarse a nuevas situaciones no vistas durante el entrenamiento, lo cual limita su eficacia en la localización visual en espacios no previamente mapeados.^[32]
 4. Evolución de otras arquitecturas: A lo largo de los últimos años han surgido nuevas arquitecturas que han demostrado mejores resultados respecto a las CNN, quedando estas en segundo plano.
 5. Confusión con el fondo: Las CNN confunden los parches de fondo de una imagen con objetos porque no pueden ver el contexto más amplio.

Por lo general, las CNN cumplen con su objetivo, pero a día de hoy pueden ser mejoradas con nuevas arquitecturas que ofrecen un mejor rendimiento.

- **YOLO:** Se entrena con imágenes completas y optimiza directamente el rendimiento de la detección. Este modelo unificado tiene varias ventajas sobre los métodos tradicionales de detección de objetos, como:[29]

1. Extremadamente rápido: Simplemente ejecutamos nuestra red neuronal en una nueva imagen en el momento de la prueba para predecir las detecciones.
2. Razonamiento global sobre la imagen: Ve la imagen completa durante el entrenamiento y el tiempo de prueba, por lo que codifica implícitamente la información contextual sobre las clases, así como su apariencia.
3. Altamente generalizable: Es menos probable que se rompa cuando se aplica a nuevos dominios o entradas inesperadas.

YOLO todavía está por detrás de los sistemas de detección de última generación en cuanto a precisión, presentando las siguientes desventajas:[29]

1. Dificultad en datos nuevos o inusuales.
2. Dificultad en la predicción de objetos pequeños.
3. Limitaciones significativas en el espacio de las predicciones de los marcos delimitadores, identificando solo una categoría por celda.

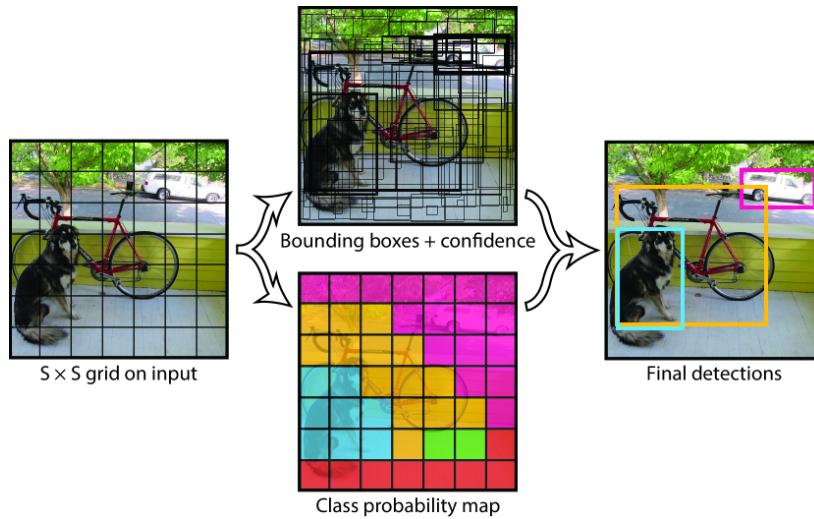


Figura 4.1: Proceso de detección de objetos utilizado por la arquitectura de red neuronal YOLO [29]

- **SSD:** Otro método de detección de objetos que estuvimos considerando es la SSD (Single Shot MultiBox Detector). Este es el segundo más conveniente ya que es mejor que YOLO por motivos como los siguientes:[22]
 1. Mayor rapidez: Un detector de disparo único para múltiples categorías que es más rápido que el estado del arte anterior para detectores de disparo único.
 2. Mayor precisión: Para la rapidez que tiene en la detección de objetos, tiene bastante buena precisión, manteniendo una cierta calidad en la relación rapidez/precisión.
 3. Entrenamiento simple de un extremo: Incluso en imágenes de entrada de baja resolución, lo que mejora aún más la relación entre velocidad y precisión.
 4. Mayor portabilidad: Al tratarse de un sistema tan rápido y que necesita pocos recursos, es más portable, permitiendo implementarse en sistemas embebidos o dispositivos móviles.

Una vez destacados los puntos positivos, mencionaremos los negativos que nos llevaron a la conclusión de utilizar Mask R-CNN:

- 1. Detección de objetos pequeños: Al igual que le pasa a YOLO, tiene dificultades en la detección de elementos pequeños.
 - 2. Mala asignación de las *bounding boxes*: Un mal establecimiento de las cajas delimitadoras puede suponer gran pérdida de información.
 - 3. Baja sensibilidad en variación de datos: La eficacia de este sistema puede verse comprometida en caso de tener un dato de entrada real en el que la iluminación o texturas sean diferentes.
 - 4. Implementación óptima: Para que SSD sea eficaz y merezca la pena, requerirá una implementación compleja.
- **Mask R-CNN:** Tras realizar un estudio previo para la elección del sistema a utilizar, nos decantamos por Mask R-CNN ya que este permite una mejor precisión en la detección de elementos que son de tamaño reducido, como es nuestro caso, ya que estamos tratando de identificar insectos. El entrenamiento y proceso de detección es mayor porque requiere un análisis de segmentación a nivel de píxel, por lo tanto, nos hemos enfocado más en la eficiencia que en la rapidez.
Para más información sobre Mask R-CNN acudir al 3.5.

5. Aspectos relevantes del desarrollo del proyecto

Esta sección se describen desde la exposición del ciclo de vida utilizado hasta los detalles más importantes de las fases de análisis, diseño e implementación del proyecto.

Para el desarrollo de el proyecto se ha de revisar antes el proceso de instalación de *python* , *CUDA toolkit*(ficheros dll para utilización de la GPU), *Mask R-CNN*, *IDE* , etc ... Para ello acudir al anexo mas concretamente en el apartado de *Manual de Usuarios*.

5.1. Datos

A la hora de buscar los datos con los que entrenaremos nuestro modelo, tuvimos en cuenta principalmente que tendría que ser un dataset de insectos cuya presencia es común en plagas para ello tuvimos varias opciones:

1. IP102 dataset. Dataset muy grande con 7500 imágenes contando con 102 categorías diferentes(insectos distintos) [43].
Este fue un posible dataset ya que tenía gran cantidad de imágenes, al final no fue elegido por el hecho de que las imágenes tenían distintos fondos y elementos que podrían ser incovenientes a la hora del entrenamiento y detección de insectos.Para acceder al dataset acceder a:
<https://github.com/xpwu95/IP102>
2. 4tu.ResearchData. Es una reconocida infraestructura integral dedicada al manejo de datos de investigación en las áreas de ciencia, ingeniería y diseño.[1]
Gracias a 4tu.ResearchData encontramos el dataset ideal para nuestro caso ya que las imágenes están tomadas sobre un adhesivo amarillo el cual hace distintivos a las formas de los insectos.Este además nos facilita el etiquetado con su correspondiente image.

Para acceder al dataset podremos hacerlo mediante el enlace:

https://data.4tu.nl/articles/dataset/Raw_data_from_Yellow_Sticky_Traps_with_insects_for_training_of_deep_learning_Convolutional_Neural_Network_for_object_detection/12707066

Adecuación de los Datos

Antes de realizar el entrenamiento de nuestro modelo, es necesario adecuar los nombres de los datos, ya que en nuestro código estos corresponderán con el ID de imagen, yendo desde el 0 al 284. Cada imagen en formato .jpg debe tener su correspondiente archivo de extensión .xml con el mismo nombre.

Para ello, utilizamos un script llamado `rename_file_to_numbers`. Este script asigna los nombres correctos a los archivos, asegurando que estén numerados secuencialmente del 0 al 284.

Además, utilizamos otro script llamado `reordenar.py`, que reordena las posiciones de las imágenes y sus etiquetas de manera aleatoria por cada época. Este reordenamiento aleatorio es crucial para evitar que el modelo utilice siempre los mismos datos en las mismas posiciones durante el entrenamiento, reduciendo así el riesgo de sobreajuste (overfitting)¹.



Figura 5.1: Ejemplo de la imagen 001.jpg y su etiquetado 001.xml

¹El overfitting es el proceso en el que el modelo aprende las predicciones de memoria, lo que lo hace inadecuado para datos reales

Como podemos ver en la figura 5.1, tendremos una etiqueta por cada imagen y en la propia etiqueta los objetos presentes en esta. Cada objeto tiene cuatro parámetros: xmin, xmax, ymin, ymax. Esto se debe a que la etiqueta se hizo formando rectángulos, por lo tanto, esta será su caja delimitadora.

Uno de los problemas que presenta el dataset es que no están correctamente definidas las características de ancho y alto de las imágenes con su correspondiente etiqueta. Este problema produce una carga incorrecta de las cajas delimitadoras que identifican a cada uno de los elementos, no definiendo correctamente las máscaras.

Para solucionar este problema, se ha realizado un script llamado `ajustarTamanosXML` que asegura que todas las imágenes y sus etiquetas tengan las mismas dimensiones (ancho y alto), garantizando así la coherencia y precisión en el procesamiento de los datos.

Etiquetado con Makesense

Aunque el dataset es apropiado, pudimos ampliar las etiquetas con la herramienta *makesense*, pudiendo realizar el etiquetado de más insectos en la imagen que no estaban etiquetados, incluso dándole un etiquetado *poligonal* el cual, al tener más puntos, permitirá realizar predicciones más exactas.²

Para utilizar la herramienta, acudiremos a:

<https://www.makesense.ai/>

²El proyecto está programado para usar los 4 puntos de las cajas delimitadoras (etiquetado rectangular), en caso de querer realizarse con etiquetado poligonal, ha de adecuarse.

5.2. COCO - Common Objects in Context

A la hora de realizar el entrenamiento de un modelo, no crearemos un modelo desde cero ya que esto nos llevaría más tiempo y recursos. En su lugar, utilizaremos un modelo preentrenado con una gran cantidad de datos y con pesos ya establecidos previamente.

Para poder descargar este modelo, lo haremos a través del siguiente enlace: [mask r-cnn.h5](#).

Una vez que lo hemos descargado, nos tendremos que asegurar de que cuenta con la extensión .h5, la cual es la que utilizan todos los modelos y nos indicará que es un archivo que maneja una gran cantidad de datos.

5.3. Objetos

Atributos A lo largo de el entrenamiento y identificación de los insectos tendremos varios atributos que hacen clave la ejecución del programa, estos son los siguientes:

- **class_ids.** Este es un array que se utiliza para guardar los identificadores de cada uno de los insectos detectados. Guardara n elementos comprendidos entre el 1 y el 3 siendo estos:
 1. Mosca Blanca. WF
 2. Nesidicoris. NS
 3. Macrolophus. MC
- **bbox - Bounding Boxes.** Correspondiente a las esquinas de las cajas delimitadoras siendo 2 puntos en el *plano(x,y)*:
 - xmin. Vértice inferior izquierda
 - xmax. Vértice superior derecha
 - ymin. Vértice inferior izquierda
 - ymax. Vértice superior derecha
- **image_id** Correspondiente a el identificador de la imagen, tendrá valores posibles del 1 al n (n = número de imágenes).
- **mask.**Corresponde al array de las máscaras de los objetos habiendo tantas máscaras como insectos,estas en la interacción 0 será todo 0 luego irán rellenándose el fondo de la caja delimitadoras con valor de 1 al contorno.
- **class_id_counter.** Corresponde el numero de elementos identificados durante la predicción.
- **score.** Este atributo correspondiente a cada objeto siendo la probabilidad de que tan seguro esta nuestro modelo de haber identificado a ese objeto,es un valor en coma flotante del 0 al 1. A continuación mostraremos el ejemplo de dos insectos detectados:



Figura 5.2: Ejemplo de una imagen de entrada a Mask R-CNN

Como podemos ver en la figura 5.2 es una entrada correspondiente al dataset en la que a cada uno de los objetos etiquetados se le asigna la mascara y un color único además de ser representada las máscaras de todos ellos.

Mientras que en la figura 5.3 corresponde a una imagen tras haber hecho predicciones sobre ella



Figura 5.3: Ejemplo de una imagen tras haber realizado predicciones

Para tener una mejor comprensión sobre cada uno de los insectos detectados y sus correspondientes atributos durante el proceso de predicción, se muestra un pequeño diagrama con sus correspondientes atributos a cada uno de ellos.

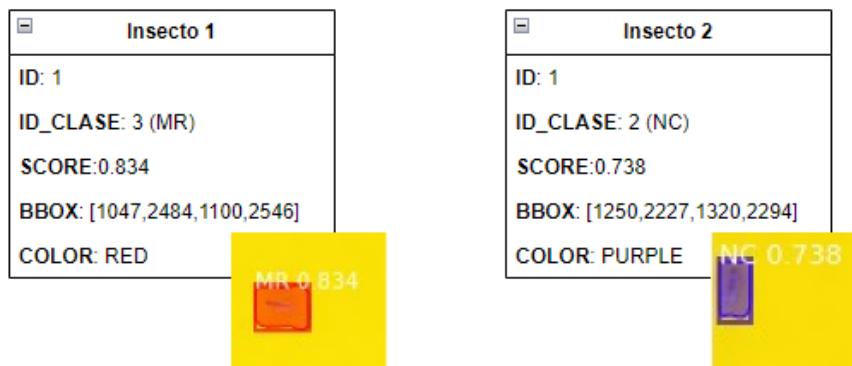


Figura 5.4: Figura correspondiente a los atributos de 2 insectos detectados

5.4. Arquitectura de la red neuronal

Una de las mejoras que presenta este proyecto frente a otros que utilizan Mask R-CNN es la utilización de redes residuales *ResNet*, las cuales son más profundas³ que las tradicionales.

³Contienen un número mayor de capas

ResNet

La clave de ResNet es el uso de conexiones residuales o "skip connections", que permiten que la entrada de una capa se sume directamente a la salida de una capa posterior basándose en bloques residuales.[21]

Bloque residual Un bloque residual es aquel que incluye dos o más capas convolucionales con una conexión directa, la cual puede saltar estas capas y sumar la entrada original a la salida.

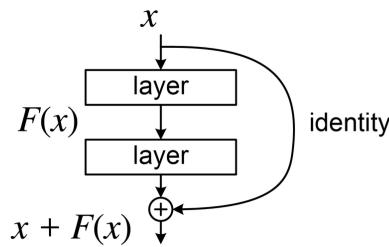


Figura 5.5: Figura correspondiente a un bloque residual [41]

Como podemos ver, el bloque residual se define como:

$$y = F(x) + x \quad (5.1)$$

Donde x es la entrada, y es la salida y $F(x)$ ⁴ es la función de transformación, aprendiendo así las diferencias entre la entrada y la salida y mejorando el problema de la degradación del gradiente⁵.

En la aplicación se podrá entrenar un modelo propio con ResNet50 y ResNet101, las cuales son dos variantes conocidas de las redes residuales y tienen un número de 50 capas y 101 capas respectivamente.

⁴Consiste en una serie de operaciones no lineales aplicadas a la entrada x , incluyendo dos o más capas convolucionales seguidas de funciones de activación, como ReLu 3.5

⁵Derivadas del error con respecto a los pesos de la red, los cuales se vuelven muy pequeños o muy grandes [7]

Ventajas de ResNet

- **Mejor precisión:** Las redes neuronales residuales (ResNet) han mostrado mejorar significativamente la precisión en diversas tareas de reconocimiento de imágenes, superando a las redes neuronales profundas convencionales.[14, 42]
- **Convergencia más rápida:** Las conexiones de salto (skip connections) facilitan un flujo de gradiente más efectivo durante el entrenamiento, lo que permite una convergencia más rápida y, en consecuencia, un entrenamiento más eficiente.[14, 42]
- **Generalización mejorada:** La estructura de ResNet ayuda a que los modelos aprendan estructuras generales en los datos en lugar de enfocarse en características específicas del conjunto de datos de entrenamiento, mejorando así la capacidad de generalización del modelo.[12, 42]
- **Transferencia de aprendizaje:** Las ResNet son populares en aplicaciones de transferencia de aprendizaje debido a su eficacia y rendimiento, permitiendo que modelos preentrenados se adapten fácilmente a nuevas tareas con menos datos adicionales.[12, 10]

Desventajas de ResNet

- **Complejidad aumentada:** Las conexiones de salto incrementan la complejidad del modelo, lo que puede traducirse en mayores requerimientos de cómputo y memoria.[12]
- **Sobreajuste:** En conjuntos de datos pequeños, las redes residuales pueden sobreajustarse debido a su estructura compleja, lo que requiere técnicas de regularización adecuadas para mitigar este problema.[12, 42]
- **Interpretabilidad limitada:** Al igual que otras redes neuronales profundas, las ResNet pueden ser difíciles de interpretar debido a su estructura compleja y ramificada, lo que complica la comprensión de cómo y por qué toman decisiones específicas.[12]

5.5. Optimizadores

SGD - Descenso de Gradiente Estocástico SGD es una variante del algoritmo de descenso de gradiente que, en lugar de calcular el gradiente promedio o la suma de los gradientes de todas las muestras en cada iteración, utiliza una única muestra aleatoria para estimar el gradiente. Esto reduce los cálculos redundantes y ahorra memoria.[31]

Algorithm 1 SGD[31]

```

Require: learning rate  $\eta_t > 0$ 
    Initialize  $w_0 \in \mathbf{R}^d$ ,  $t = 0$ 
    while Stop Criteria is True do
        Sample  $\xi_t \sim P$  with  $i \in \{1, 2, \dots, n\}$ 
         $\zeta_t = \nabla f_i(w_t; \xi_t)$ 
         $w^+ = w - \eta_t \cdot \zeta_t$ 
         $t = t + 1$ 
    end while

```

RMSprop - Root Mean Square Propagation RMSprop es un algoritmo de optimización adaptativa ampliamente utilizado en el entrenamiento de redes neuronales profundas debido a su capacidad para manejar gradientes de magnitudes variadas y mejorar la eficiencia computacional. Fue desarrollado por Geoffrey Hinton y ha mostrado mejorar la convergencia en comparación con otros métodos como el descenso de gradiente estocástico (SGD).[28]

Algorithm 2 RMSprop [31]

```

Require:  $\alpha$ : Tasa de aprendizaje
Require:  $\beta$ : Coeficiente de decaimiento exponencial
Require:  $\epsilon$ : Valor pequeño para evitar divisiones por cero
Require:  $f(\theta)$ : Función objetivo estocástica con parámetros  $\theta$ 
Require:  $\theta_0$ : Vector de parámetros inicial
1:  $m_0 \leftarrow 0$  (Inicializar el vector de momento)
2:  $t \leftarrow 0$  (Inicializar el paso de tiempo)
3: while  $\theta_t$  no haya convergido do
4:    $t \leftarrow t + 1$ 
5:    $g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1})$  (Obtener los gradientes respecto a la función objetivo estocástica en el paso de tiempo  $t$ )
6:    $m_t \leftarrow \beta \cdot m_{t-1} + (1 - \beta) \cdot g_t^2$  (Actualizar la estimación del segundo momento)
7:    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \frac{g_t}{\sqrt{m_t} + \epsilon}$  (Actualizar los parámetros)
8: end while
9: return  $\theta_t$  (Parámetros resultantes)

```

Adam Adam es computacionalmente eficiente y tiene bajos requerimientos de memoria. Ajusta las tasas de aprendizaje de manera adaptativa para cada parámetro, utilizando estimaciones de los momentos de primer y segundo orden de los gradientes.

Combina las ventajas de AdaGrad, que maneja bien los gradientes dispersos, y RMSProp, que es eficaz en configuraciones no estacionarias⁶.

Algorithm 3 Adam[25]

Require: α : Stepsize

Require: $\beta_1, \beta_2 \in [0, 1]$: Exponential decay rates for the moment estimates

Require: $f(\theta)$: Stochastic objective function with parameters θ

Require: θ_0 : Initial parameter vector

- 1: $m_0 \leftarrow 0$ (Initialize 1st moment vector)
 - 2: $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
 - 3: $t \leftarrow 0$ (Initialize timestep)
 - 4: **while** θ_t not converged **do**
 - 5: $t \leftarrow t + 1$
 - 6: $g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
 - 7: $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
 - 8: $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
 - 9: $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
 - 10: $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
 - 11: $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$ (Update parameters)
 - 12: **end while**
 - 13: **return** θ_t (Resulting parameters)
-

⁶Situaciones en las que las propiedades estadísticas de los datos cambian con el tiempo se conocen como *configuraciones no estacionarias*[44]

5.6. Resultados

Los resultados iniciales no fueron satisfactorios debido a la inadecuada configuración de parámetros, como el número de pasos (steps) durante el entrenamiento, el número de pasos durante la validación y el algoritmo de optimización seleccionado.

Para abordar esta situación, realizamos múltiples experimentos ajustando diversos parámetros y evaluando su impacto mediante gráficos y análisis de errores. Este proceso iterativo nos permitió identificar una configuración óptima.

Para determinar el número adecuado de steps por entrenamiento, consideramos el tamaño del dataset y la cantidad de imágenes procesadas en cada step. En este caso, tenemos un total de 284 imágenes en el dataset y procesamos 2 imágenes por cada step. Utilizamos la siguiente fórmula para calcular el número de steps necesarios:

$$\text{Número total de steps} = \frac{\text{Número total de imágenes}}{\text{Imágenes por step}}$$

Aplicando los valores específicos de nuestro caso:

$$\text{Número total de steps} = \frac{284}{2} = 142$$

Por lo tanto, un número apropiado de steps por epoch sería 142. Esto garantiza que todas las imágenes del dataset se utilicen durante cada epoch de entrenamiento, promoviendo un aprendizaje completo y evitando el sobreajuste (overfitting) al no repetir las mismas imágenes en cada epoch.

SGD

Inicialmente, utilizamos el optimizador Stochastic Gradient Descent (SGD) y variamos sistemáticamente el número de pasos durante la validación para determinar su efecto en el rendimiento del modelo.

En las siguientes figuras, se presentan las gráficas de pérdida de entrenamiento y pérdida de validación para dos configuraciones diferentes del algoritmo SGD.

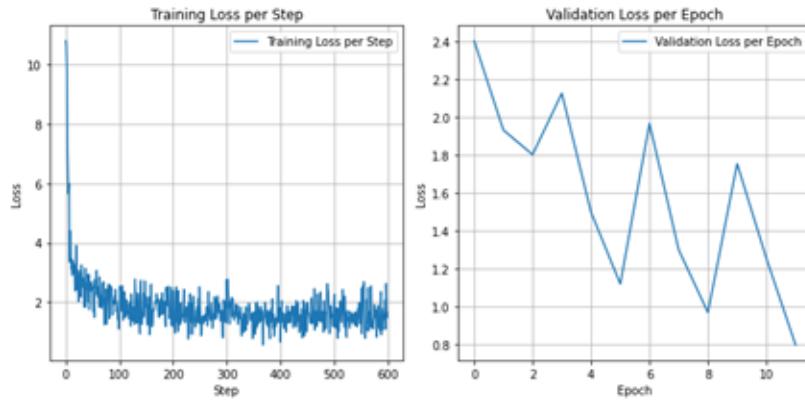


Figura 5.6: Pérdida de entrenamiento por paso (izquierda) y pérdida de validación por época (derecha) después de 600 pasos y 12 épocas usando SGD.

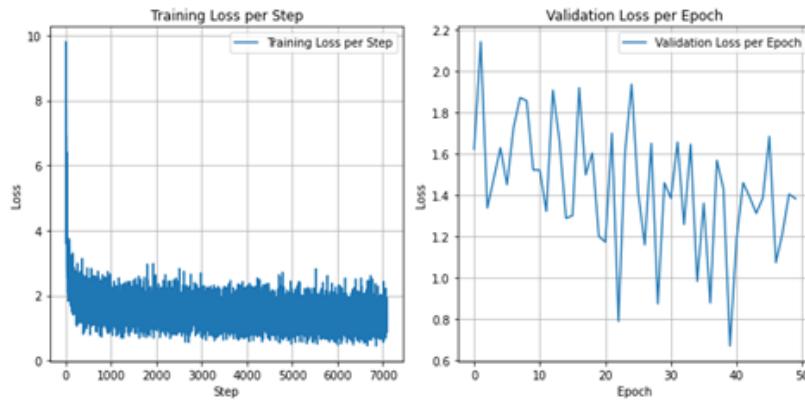


Figura 5.7: Pérdida de entrenamiento por paso (izquierda) y pérdida de validación por época (derecha) después de 7000 pasos y 50 épocas usando SGD.

En la Figura 5.6, observamos que la pérdida de entrenamiento disminuye significativamente en los primeros 100 pasos y luego se estabiliza, aunque con fluctuaciones. La pérdida de validación muestra una tendencia general a la baja, pero con variabilidad significativa entre épocas, indicando inestabilidad en la validación inicial.

En la Figura 5.7, la pérdida de entrenamiento también muestra una disminución rápida inicial y una estabilización posterior, aunque con fluctuaciones más constantes a lo largo de los 7000 pasos. La pérdida de validación, aunque presenta variabilidad, muestra una tendencia general más estable.

Estos resultados sugieren que la segunda configuración, con un mayor número de pasos y épocas, proporciona una mayor estabilidad y precisión en el modelo.

A continuación, se muestran los resultados obtenidos mediante la aplicación desarrollada utilizando el algoritmo SGD.



Figura 5.8: Primer resultado de la aplicación usando SGD.



Figura 5.9: Segundo resultado de la aplicación usando SGD.

El número total esperado de insectos es de 47, desglosado en 11 macrolophus y 36 whiteflies.

Como se observa en los resultados, la configuración inicial del algoritmo SGD no fue efectiva, detectando solo 23 de los 47 insectos. En contraste, la segunda configuración mostró una mayor precisión, logrando detectar 43 de los 47 insectos esperados.

RMSprop

Dado que se ha observado una mejora en las detecciones con respecto a otros optimizadores, se decidió comprobar los resultados con el optimizador RMSprop para ver si presentaba mejoras respecto a SGD. Tras realizar un entrenamiento de 10 y 144 pasos por epoch, se obtuvieron los siguientes resultados:

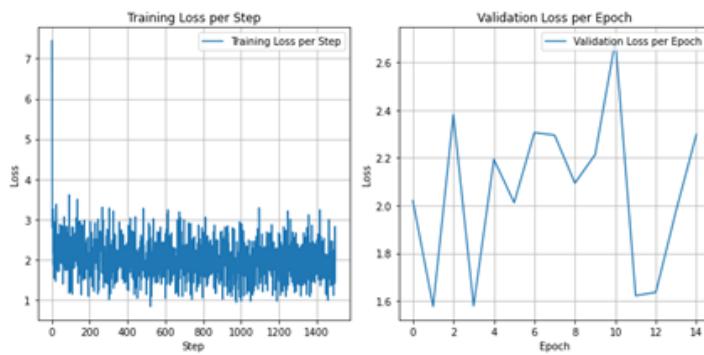


Figura 5.10: Segundo resultado de la aplicación usando RMSprop.

Como podemos ver inicialmente, la pérdida es alta y desciende rápidamente en los primeros pasos. Sin embargo, la pérdida de validación no muestra una tendencia clara a disminuir, sino que fluctúa de manera notable. Hay picos altos en ciertas épocas (por ejemplo, en las épocas 4, 10 y 13), lo que indica que el modelo está sobreajustándose en algunos puntos o que la validación no es completamente estable.

En general no presenta una mejoría respecto a SGD por lo tanto se decidió continuar comparando resultados con el optimizador Adam

Adam

A continuación, comparamos los resultados con el optimizador Adam, buscando una mejora en cuanto a la validación y mitigar el problema de las variaciones producidas durante el entrenamiento.

Para ello, se decidió disminuir hiperparámetros como la tasa de aprendizaje para así tener menor variación mediante el uso de un programador de tasa de aprendizaje (*learning rate scheduler*). Este programador reduce la tasa de aprendizaje en un 5 % cada 2 épocas.

Se ha utilizado el callback `LearningRateScheduler` de Keras para implementar este programador en el proceso de entrenamiento, más concretamente en la función `train`.

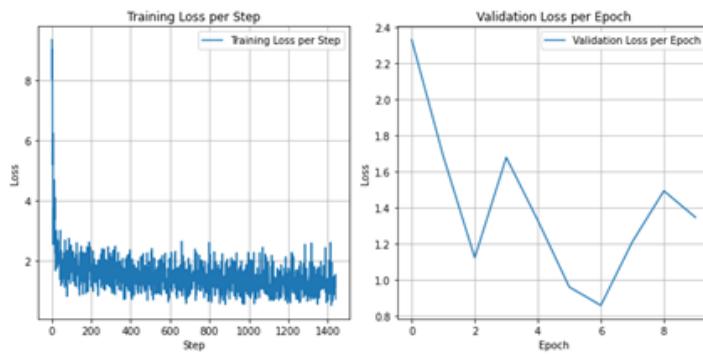


Figura 5.11: Segundo resultado de la aplicación usando Adam.

Tras realizar un entrenamiento de 10 y 144 steps cada epoch, se obtuvieron los siguientes resultados:

- Al igual que en los resultados anteriores, la pérdida de entrenamiento disminuye rápidamente al inicio. Después de la caída inicial, la pérdida se estabiliza más rápidamente y con menos fluctuaciones comparado con los resultados sin el programador, teniendo un entrenamiento más estable y controlado.
- En cuanto al gráfico de la validación, todavía muestra algunas variaciones, pero estas son menos pronunciadas y ocurren de manera más controlada.

5.7. Mejoras

Para realizar las mejoras, primero se analizaron los resultados con los distintos optimizadores. Se determinó que el más apropiado era *Adam* en comparación con *RMSprop* y *SGD*. Además, se ajustaron ciertos parámetros como la *tasa de aprendizaje*, las proporciones para *test* y *validación*, y el número de *steps* por entrenamiento.

Para visualizar los resultados más detalladamente, consulte la sección [Resultados](#).

Problema de sobreajuste

Nos enfrentamos a uno de los problemas más comunes en redes neuronales: el sobreajuste. Este problema se debe en parte al tamaño del dataset, que no es muy grande, compuesto por 285 imágenes, cada una de las cuales contiene varios insectos.

Para mitigar este problema, utilizamos una herramienta de etiquetado para etiquetar más insectos en las imágenes existentes del dataset, ya que no todos los insectos estaban etiquetados inicialmente. Para ello, utilizamos la herramienta [Makesense](#).

Otra solución que aplicamos para evitar este problema fue un script llamado *reorganización*, el cual reorganiza el dataset de forma aleatoria en cada epoch, evitando que las mismas imágenes se utilicen consecutivamente durante el entrenamiento.

Adicionalmente, se implementaron técnicas de regularización como la de *batch normalization*⁷ y *dropout*⁸. Estas no llegaron a presentar una mejoría clara sobre los resultados.

⁷técnica que normaliza las activaciones de una capa en mini-lotes durante el entrenamiento[24]

⁸Elimina unidades aleatoriamente (junto con sus conexiones) de la red neuronal durante el entrenamiento.[34]

Uso previo de un modelo entrenado en COCO

Motivación

Entrenar un modelo de detección de objetos desde cero puede ser extremadamente costoso en términos de tiempo y recursos computacionales. Para abordar este desafío y mejorar el rendimiento del modelo, se optó por utilizar un modelo previamente entrenado en el conjunto de datos COCO (Common Objects in Context) como punto de partida.

Ventajas del uso de un modelo preentrenado

1. *Aceleración del Entrenamiento:* Los modelos previamente entrenados ya han aprendido características generales a partir de un gran volumen de datos, lo que permite que el proceso de entrenamiento sea más rápido y eficiente en comparación con entrenar desde cero.
2. *Mejora del Rendimiento:* Utilizar un modelo preentrenado ayuda a mejorar la precisión del modelo final, ya que las características aprendidas en el conjunto de datos COCO pueden transferirse a nuestro conjunto de datos específico de insectos, facilitando la detección y segmentación de objetos.
3. *Optimización de Recursos:* El uso de modelos preentrenados reduce la necesidad de grandes volúmenes de datos anotados, lo cual es beneficioso cuando se tiene un conjunto de datos limitado en tamaño y anotaciones.

Implementación

Para esta tarea, se seleccionó un modelo Mask R-CNN previamente entrenado en el conjunto de datos COCO, el cual contiene 80 categorías de objetos comunes y cuenta con millones de imágenes anotadas. Los pasos realizados fueron los siguientes:

1. *Selección del modelo:* Se optó por el modelo Mask R-CNN con una arquitectura de backbone ResNet-50 y ResNet-100, conocida por su balance entre precisión y eficiencia computacional.
2. *Carga del modelo preentrenado:* Se cargó el modelo previamente entrenado en COCO utilizando bibliotecas como Detectron2 o mmdetection, que facilitan la integración de modelos preentrenados y su ajuste (fine-tuning).
3. *Ajuste del modelo (Fine-Tuning):* El modelo se ajustó a nuestro conjunto de datos específico de insectos. Esto implicó ajustar las capas superiores del modelo para que aprendieran características específicas de los insectos, mientras que las capas inferiores conservaban las características generales aprendidas del conjunto de datos COCO.

Resultados

El uso del modelo previamente entrenado en COCO demostró ser altamente beneficioso, evidenciado por:

1. *Reducción del Tiempo de Entrenamiento:* El tiempo necesario para alcanzar una precisión óptima fue significativamente menor en comparación con el entrenamiento desde cero.
2. *Mejora de la Precisión:* Se observó un aumento en la precisión de detección y segmentación de los insectos, debido a la transferencia de conocimientos del modelo preentrenado.
3. *Estabilidad en el Entrenamiento:* El entrenamiento fue más estable, con una convergencia más rápida y menor riesgo de sobreajuste, gracias a las características robustas aprendidas por el modelo preentrenado.

En conclusión, la utilización de un modelo previamente entrenado en el conjunto de datos COCO proporcionó una base sólida sobre la cual construir y ajustar nuestro modelo específico de detección y segmentación de insectos, resultando en mejoras significativas en términos de eficiencia y precisión.

6. Trabajos relacionados

En los campos de segmentación de imágenes y visión por computadora, más concretamente en el de detección de insectos, se han realizado numerosos estudios y proyectos que tienen algunas características relacionadas con este proyecto, ya sea en la utilización de la arquitectura en la red neuronal aplicada o bien en los propósitos y algoritmos utilizados en este.

Picture Insect Esta app sirvió como inspiración ya que es una aplicación móvil diseñada para la identificación de insectos a través del uso de tecnologías avanzadas de visión por computadora e inteligencia artificial, permitiendo a los usuarios tomar una foto de un insecto y obtener información detallada sobre la especie.

El siguiente enlace corresponde a su descarga: <https://apps.apple.com/us/app/picture-insect-spiders-bugs/id1461694973?mt=8>

Mask R-CNN for Object Detection and Segmentation Este proyecto, además de ser un claro ejemplo de la aplicación del Mask R-CNN en la vida real para la detección de elementos (llevando así a una mejor comprensión), ha servido como punto de partida para ciertas configuraciones de Mask R-CNN, como ha sido en los aspectos de visualización y compilación, entre otros.

Para poder acceder al repositorio, acudir al siguiente enlace: https://github.com/matterport/Mask_RCNN

7. Conclusiones y Líneas de trabajo futuras

En este estudio, hemos desarrollado y analizado un modelo de detección y segmentación de objetos utilizando Mask R-CNN, enfocado en la identificación de insectos en imágenes. A lo largo del proyecto, se implementaron varias técnicas y estrategias para optimizar el rendimiento del modelo y abordar los desafíos relacionados con el tamaño limitado del conjunto de datos. Las principales conclusiones son las siguientes:

1. **Eficacia del modelo preentrenado:** Utilizar un modelo preentrenado en COCO resultó ser una estrategia efectiva, permitiendo reducir significativamente el tiempo de entrenamiento y mejorar la precisión del modelo final.
2. **Reducción del sobreajuste:** La aplicación de técnicas como el ajuste fino (Fine-Tuning), el aumento de datos y la regularización ayudaron a reducir el problema del sobreajuste, mejorando la capacidad del modelo para generalizar a datos no vistos.
3. **Optimización de hiperparámetros:** La búsqueda y optimización de hiperparámetros fueron cruciales para maximizar el rendimiento del modelo. Herramientas como Grid Search y Random Search permitieron encontrar combinaciones óptimas que mejoraron la precisión y estabilidad del modelo.
4. **Importancia de la calidad de los datos:** La calidad y cantidad de los datos anotados tuvieron un impacto significativo en el rendimiento del modelo. La mejora en las anotaciones y el uso de técnicas

de aumento de datos fueron fundamentales para obtener resultados precisos.

5. **Transferencia de aprendizaje y arquitecturas:** Experimentar con diferentes arquitecturas de backbone y técnicas avanzadas de transferencia de aprendizaje proporcionó una comprensión más profunda de cómo estas variantes afectan el rendimiento del modelo, destacando la importancia de seleccionar la arquitectura adecuada para la tarea específica.

7.1. Líneas de trabajo futuras

A partir de los resultados obtenidos y las conclusiones derivadas de este trabajo, se identifican varias líneas de trabajo futuras que pueden contribuir a mejorar aún más el modelo desarrollado y explorar nuevas aplicaciones:

1. **Ampliación del conjunto de datos:** Recolectar y anotar un conjunto de datos más grande y diverso para mejorar la capacidad del modelo para generalizar a diferentes tipos de insectos y escenarios.
2. **Ensamblar modelos:** Desarrollar un conjunto de modelos Mask R-CNN con diferentes configuraciones y backbones para mejorar la precisión y robustez de las predicciones mediante la combinación de resultados.
3. **Optimización para tiempo real:** Optimizar el modelo para aplicaciones en tiempo real, reduciendo la latencia de inferencia y mejorando la eficiencia computacional para su uso en dispositivos móviles y sistemas embebidos.
4. **Exploración de nuevas arquitecturas:** Investigar y experimentar con nuevas arquitecturas de redes neuronales y técnicas de aprendizaje profundo, como EfficientDet, para comparar su rendimiento con Mask R-CNN y potencialmente adoptar nuevas soluciones.
5. **Aplicación offline y en dispositivos móviles:** Optimizar el modelo para permitir su funcionamiento en modo offline, incluso desde dispositivos móviles, asegurando así su disponibilidad en entornos sin conexión a internet.
6. **Incremento en el número de insectos detectados:** Ampliar el modelo para incluir un mayor número de especies de insectos, mejorando así su capacidad para identificar y clasificar una variedad más amplia de insectos.
7. **Integración con Sistemas de Información Geográfica (SIG):** Explorar cómo se puede integrar el modelo de detección y segmentación de insectos con los Sistemas de Información Geográfica (SIG) para monitorizar y analizar espacialmente las poblaciones de insectos, proporcionando herramientas avanzadas para la gestión y toma de decisiones en contextos ecológicos y agrícolas. Este enfoque permite un control global de las poblaciones de insectos y la gestión efectiva de especies invasoras.

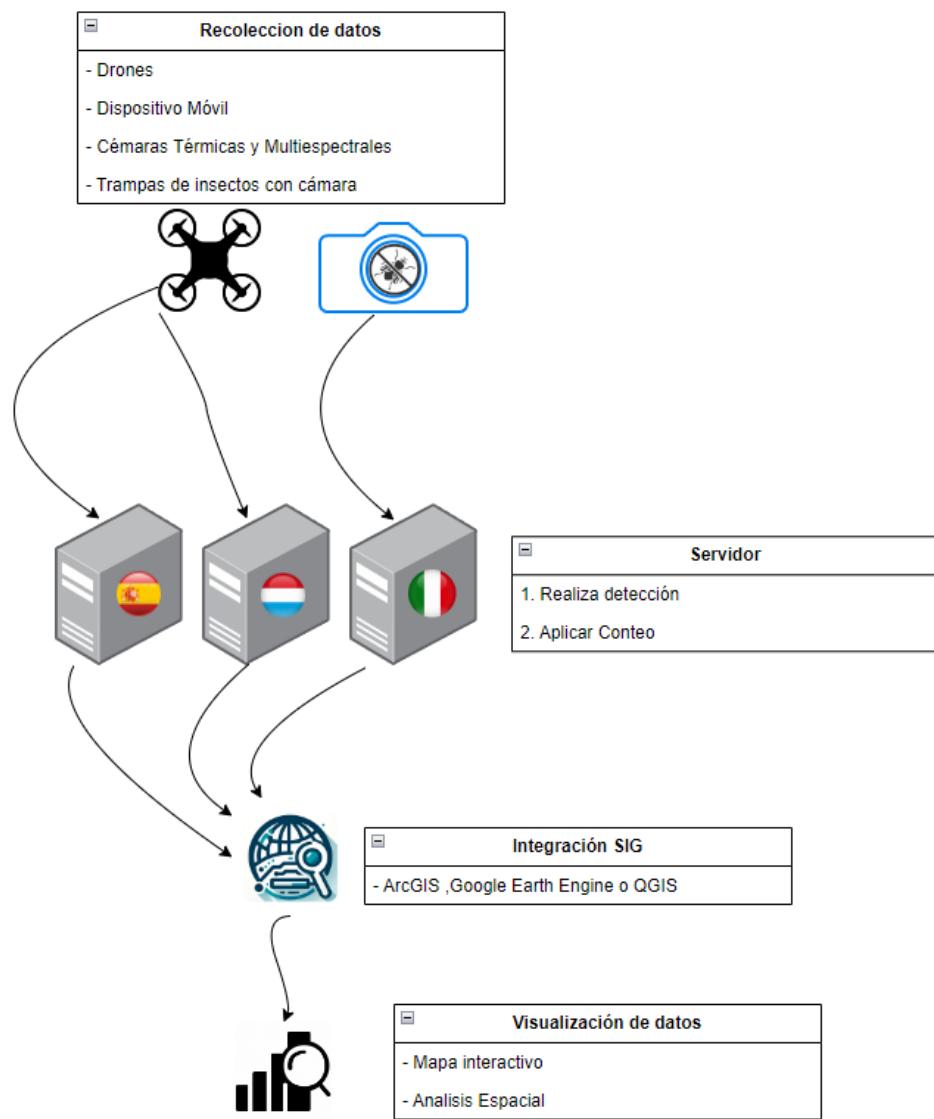


Figura 7.1: Posible sistema SIG

Bibliografía

- [1] 4TU.ResearchData Community. 4tu.researchdata community portal. <https://community.data.4tu.nl/>, 2024. Último acceso: [Fecha de tu último acceso].
- [2] Fritz AI. Exploring activation and loss functions in machine learning. <https://fritz.ai/exploring-activation-and-loss-functions-in-machine-learning/>, 2023. Accedido en 2024.
- [3] Luis Amador Hidalgo. *Inteligencia artificial y sistemas expertos*. Universidad de Córdoba, Servicio de Publicaciones, 1996.
- [4] Appen. Recent developments in neural networks. <https://www.appen.com/blog/recent-developments-neural-networks>, 2023. <https://www.appen.com/blog/recent-developments-neural-networks>.
- [5] Varios Autores. An Introduction to Convolutional Neural Networks. *arXiv:1511.08458*, 2015.
- [6] Baeldung. Epoch in neural networks, 2023. Último acceso: 2024.
- [7] David Balduzzi, Marcus Frean, Lennox Leary, JP Lewis, Kurt Wan-Duo Ma, and Brian McWilliams. The shattered gradients problem: If resnets are the answer, then what is the question? *arXiv preprint arXiv:1702.08591*, 2017.
- [8] Julio Berbel. La inteligencia artificial en la agricultura: perspectivas de los sistemas expertos. 1989.
- [9] Jason Brownlee. Understand the dynamics of learning rate on deep learning neural networks. <https://machinelearningmastery.com/>.

- [com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/](https://www.cibebuchi.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/), 2019. Último acceso: 2024.
- [10] Built In. ResNet architecture and its variants. [Online; accessed 15-May-2024].
 - [11] Cibebuchi. Optimizing epochs in neural networks: A practical approach to stability and performance. <https://blog.cibebuchi.com/optimizing-epochs-in-neural-networks-a-practical-approach-to-stability-and-performance/>, 2023. Último acceso: 2024.
 - [12] Data Basecamp. ResNet advantages and disadvantages. [Online; accessed 15-May-2024].
 - [13] DataCamp. Introduction to convolutional neural networks (cnns), 2021. Último acceso: [Fecha de tu último acceso].
 - [14] DataGen. ResNet advantages and disadvantages. [Online; accessed 15-May-2024].
 - [15] Fouriering. Procesamiento digital de imágenes, 2020. Available online: <https://www.fouriering.com/post/procesamiento-digital-de-imagenes> (Accessed on 21 October 2021).
 - [16] Barcelona Geeks. Búsqueda selectiva para la detección de objetos: R-cnn. <https://barcelonageeks.com/busqueda-selectiva-para-la-deteccion-de-objetos-r-cnn/>, 2023.
 - [17] Ross Girshick. Fast r-cnn. <https://paperswithcode.com/method/fast-r-cnn>, 2015. Acceso: 9 de Abril de 2024.
 - [18] Ross Girshick. Fast r-cnn. <https://paperswithcode.com/method/fast-r-cnn>, 2015. Acceso: 9 de Abril de 2024.
 - [19] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *arXiv preprint arXiv:1311.2524*, 2014.
 - [20] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. *arXiv preprint arXiv:1703.06870*, 2017. Accessed: [Insert Date Here].
 - [21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

- [22] Christian Heipke. *Crowdsourcing of Geospatial Data*, volume XLI-B6, pages 83–90. Springer International Publishing, Cham, 2016.
- [23] IBM. Neural networks. <https://www.ibm.com/es-es/topics/neural-networks>, 2023. Último acceso en 2024.
- [24] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 448–456, 2015.
- [25] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [26] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.
- [27] LearnOpenCV. Mask r-cnn. <https://learnopencv.com/tag/mask-r-cnn-2/>, 2022. Último acceso: [Fecha de acceso].
- [28] Huan Li and Zhouchen Lin. On the $o(\frac{\sqrt{d}}{T^{1/4}})$ convergence rate of rmsprop and its momentum extension measured by ℓ_1 norm: Better dependence on the dimension. *arXiv preprint arXiv:2402.00389*, 2024.
- [29] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *arXiv preprint arXiv:1506.02640*, 2016.
- [30] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. <https://paperswithcode.com/method/rpn>, 2015.
- [31] Herbert Robbins and Sutton Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22(3):400–407, 1951.
- [32] Torsten Sattler, Qunjie Zhou, Marc Pollefeys, and Laura Leal-Taixe. Understanding the limitations of cnn-based absolute camera pose regression. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3302–3312, 2019.
- [33] Prince Kumar Singh. Pytorch: Tensors and its operations. <https://www.analyticsvidhya.com/blog/2023/03/pytorch-tensors-and-its-operations/>, March 2023.

- [34] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.
- [35] Ljubiša Stanković and Danilo Mandic. Convolutional neural networks demystified: A matched filtering perspective-based tutorial. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2023.
- [36] Kazheen Ismael Taher and Adnan Mohsin Abdulazeez. Deep learning convolutional neural network for speech recognition: a review. *International Journal of Science and Business*, 5(3):1–14, 2021.
- [37] Towards Data Science. A comprehensive guide to convolutional neural networks — the eli5 way. <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a5>, 2018. Último acceso: 9 de Abril de 2024.
- [38] User:X93ma. User contributions - statwiki, 2022. [Online; accessed 7-April-2024].
- [39] Analytics Vidhya. Understanding loss function in deep learning. *Analytics Vidhya Blog*, June 2022.
- [40] Mrinal Walia. Semantic segmentation vs. instance segmentation: Explained. <https://blog.roboflow.com/difference-semantic-segmentation-instance-segmentation/>, 2022. Accessed: 2024-04-07.
- [41] Wikipedia contributors. Red neuronal residual — Wikipedia, The Free Encyclopedia, 2024. [Online; accessed 15-May-2024].
- [42] Wisdom ML. Understanding ResNet-50. [Online; accessed 15-May-2024].
- [43] Xiaoping Wu, Chi Zhan, Yukun Lai, Ming-Ming Cheng, and Jufeng Yang. Ip102: A large-scale benchmark dataset for insect pest recognition. In *IEEE CVPR*, pages 8787–8796, 2019.
- [44] Ailing Zeng, Yicheng He, Chen Zhang, et al. Non-stationary transformers: Exploring the stationarity in time series forecasting, 2022.
- [45] Ce Zheng, Xiaolin Xie, Longtao Huang, Binyao Chen, Jianling Yang, Jiewei Lu, Tong Qiao, Zhun Fan, and Mingzhi Zhang. Detecting glaucoma based on spectral domain optical coherence tomography

imaging of peripapillary retinal nerve fiber layer: a comparison study between hand-crafted features and deep learning model. *Graefe's Archive for Clinical and Experimental Ophthalmology*, 258:577–585, 2020.