Universidad Internacional de la Rioja (UNIR)

ESIT

Máster Universitario en Inteligencia Artificial

Sistema *edge computing* para la detección de *Lobesia Botrana* mediante imágenes

Trabajo Fin de Máster

presentado por:

José Luis Mira Serrano

Dirigido por:

Antoni Manel Ferragut Amengual

Jesús Barba Romero

Ciudad: Toledo

Fecha: 23 de Febrero de 2022

Resumen

En la actualidad, la recogida de datos, que permiten estimar la población de insectos, se realiza por observación directa y recuento manual de los mismos. Los técnicos de prevención de plagas cruzan esta información obtenida con otras variables de tipo ambiental (temperatura, humedad, etc.), alimentando de esta forma un modelo de predicción el cual permite adelantarse a épocas de riesgo para poder aplicar las medidas preventivas correspondientes. Otro beneficio a tener en cuenta de la aplicación de estos modelos es la capacidad de ajustar los tratamientos fitosanitarios con el consiguiente ahorro para los agricultores e impacto positivo en el medio ambiente.

La tarea de recuento requiere de un gran esfuerzo y cantidad de mano de obra, lo que también lleva en ocasiones a limitar su aplicabilidad debido a los altos costes del proceso.

Este TFM pretende desarrollar un algoritmo de inteligencia artificial y visión por computador, que sera desplegado en un sistema Raspberry Pi. El resultado supone un primer paso para automatizar este tipo de tareas.

Palabras Clave: Control de Plagas, Inteligencia Artificial, Sistemas *on the edge*, Visión por Computador.

Abstract

Nowadays, the collection of data that enables the estimation of the insect population is done by direct observation and the counting of individuals is manual. Pest prevention technicians cross-reference this information with other environmental variables (temperature, humidity, etc.), thus feeding a prediction model which allows to anticipate peaks of risk in order to apply the corresponding preventive measures. Another benefit to be taken into account in the application of these models is the ability to adjust phytosanitary treatments with consequent savings for farmers and positive impact on the environment.

The counting task requires a lot of effort and labour, which also sometimes leads to limiting its applicability, due to the high costs of the process.

This project aims to develop an artificial intelligence and computer vision algorithm, wich will be deployed in a Raspberry Pi system. The result aims to be a first step to automate this kind of tasks.

Keywords: Artificial Inteligence, Computer Vision, On the edge systems, Pest Control

Índice de Contenidos

Índice de Ilustraciones

Índice de Tablas

Capítulo 1

Introducción

Vivimos en plena era de la información. A día de hoy la humanidad está sufriendo uno de los mayores cambios en su historia, equiparable a los cambios sociales que supuso la aparición de la agricultura o la revolución industrial. Las tecnologías de la información han revolucionado por completo nuestra forma de comportarnos e interactuar con nuestro entorno, tanto individual como colectivamente.

Los diferentes sectores económicos avanzan y evolucionan rápidamente hacia un mundo dominado por la tecnología. Este proceso de innovación hace que todos deban adaptarse para evitar quedarse obsoletos y mejorar la calidad de vida de las personas que se dedican a éstos. Sin embargo, a veces hay lugares donde la utilización de estas tecnologías no está demasiado arraigada, ya sea por la falta de iniciativa para implantarlas o por el desconocimiento de las mismas. En esta situación se encuentra, aunque cada vez menos, la mayor parte del mundo rural.

Entre los sectores más olvidados tenemos el agrícola, ganadero, pesca, silvicultura... y entre ellos también encontramos al vitivinícola, sector entorno al que girará este proyecto. Todos los mencionados anteriormente son pertenecientes al sector primario, piedra fundamental de nuestra economía, es por ello que se debería tener más en cuenta a la hora de invertir en avances tecnológicos ya que dicha inversión puede garantizar una reducción en los gastos de producción, aumentando así su rentabilidad y potenciando la economía, además de mantener activa una parte tan amplia cómo es el mundo rural que se mantiene gracias a empleos relacionados con el sector primario.

La Unión Europea es el principal productor de vino del mundo concentrando el 45 % de la superficie vitícola mundial, el 65 % de la producción y además el 70 % de las expor-

taciones. Dentro de la Unión Europea, los países más importantes son Francia, Italia y España [6].

1.1. Motivación

Las técnicas actuales relacionadas con la detección y prevención de una plaga de insectos conllevan un despliegue en el cual se deben instalar una serie de trampas, denominadas cromáticas, con el fin de analizar el volumen de insectos presentes en el medio. Posteriormente, esas mismas trampas se recogen y analizan en un laboratorio por un técnico contando y clasificando los insectos de manera totalmente manual. El número de trampas crece en relación a la superficie a cubrir y a la cantidad de densidad de las mismas por metro cuadrado. En la Figura 1.1 se puede observar una trampa cromática desplegada en un viñedo.

Figura 1.1: Trampa cromática y captura recogida

Este proceso manual, en los casos en los que quiere analizar una gran cantidad de terreno o zonas con una alta densidad de trampas por metro cuadrado, puede llegar a convertirse en una tarea muy tediosa para los técnicos agrícolas. Además, el tiempo que transcurre entre el despliegue y la obtención de resultados puede ser demasiado elevado para poder actuar con rapidez contra una plaga, esto repercute con la aparición de enormes pérdidas del producto que desencadenará una serie de problemas, por ejemplo, en los acuerdos de exportaciones con otros países.

Con tal de modernizar estas prácticas, este TFM tratará el desarrollo de un sistema, dentro del marco de *Smart Farming*, el cuál sea capaz de identificar de forma automática el número de insectos en una trampa mediante técnicas de segmentación y análisis de imagen.

1.2. Planteamiento del Trabajo

Para comprender la solución final que propone este TFM, es necesario desarrollar previamente el proceso a seguir hasta llegar a dicho objetivo.

En una primera instancia, se va a realizar una investigación de las soluciones tecnológicas ya aplicadas en algunos viñedos o en otros sectores agrícolas con el mismo problema. De hecho, uno de los proyectos que se tendrán muy en cuenta es mi Trabajo de Fin de Grado, el cuál se centraba en el control de plagas de la mosca del olivo mediante el procesamiento de imágenes. Este trabajo previo no hacía uso de técnicas de inteligencia artificial, aplicando técnicas de procesamiento de imagen "clásicas".

Continuando tras la fase de investigación de soluciones y tecnologías, se elegirán aquellas que utilizaremos para el desarrollo de este proyecto. Siguiendo con un estudio de las arquitecturas neuronales involucradas en la casuística de este proyecto.

Una vez definida la metodología de gestión del proyecto y la planificación de este, se va a comenzar con la implementación y evaluación de resultados de éste. Por último, se realizará la adaptación del sistema para su correcto funcionamiento en dispositivos *on the edge*.

1.3. Estructura del Documento

El presente documento se compone de distintos capítulos, en cada uno de los cuales se aborda el siguiente contenido:

Capítulo 1: Introducción. En este capítulo se aborda el problema a resolver en este Trabajo Fin de Máster, así como la relevancia para el sector afectado.

Capítulo 2: Contexto y Estado del Arte. Desarrollo del contexto en el que se desplegará el proyecto además de las tecnologías utilizadas para alcanzar el resultado final.

Capítulo 3: Objetivos y Metodología del Trabajo. Explicación de los objetivos y funcionalidades básicas del proyecto desarrollado en el marco de este TFM, así como la metodología utilizada para la gestión de su desarrollo.

Capítulo 4: Desarrollo del Trabajo. En este capítulo se presentan y desglosan las distintas fases que se han llevado a cabo en el ciclo de vida del proyecto.

Capítulo 5: Conclusiones y Trabajo Futuro. Por último, se incluirá una reflexión sobre los conocimientos adquiridos a lo largo del desarrollo de este TFM, así como una justificación de los objetivos completados y las posibles mejoras o líneas de continuación que deriven del presente TFM.

Capítulo 2

Contexto y Estado del Arte

Para comprender la solución final que propone este TFM es necesario desarrollar previamente el proceso a seguir hasta llegar a dicho objetivo.

En una primera instancia, se va a realizar una investigación de las soluciones tecnológicas ya aplicadas en algunos viñedos o en otros sectores agrícolas para el mismo problema. De hecho, uno de los proyectos que se tendrán muy en cuenta para la elaboración de éste, es mi trabajo de fin de grado, el cuál se centraba en el control de plagas de la mosca del olivo mediante el procesamiento de imágenes, pero que carecía de la utilización de inteligencia artificial.

2.1. Lenguajes de programación utilizados en proyectos de IA y visión por computador

Podemos destacar dos lenguajes de programación empleados habitualmente en el desarrollo de proyectos de inteligencia artificial y visión por computador.

2.1.1. Python

Python es un lenguaje de programación que fue creado en 1991 por Guido Van Rossum. Se trata de un lenguaje de alto nivel, interpretado, orientado a objetos y con semántica dinámica. Su sintaxis hace hincapié en la legibilidad del código y facilita por tanto la productividad del usuario. Al tratarse de un lenguaje interpretado hace que sea mucho mas fácil portarlo a diferentes sistemas y dispositivos, aunque por el contrario sufre cierta penalización en la velocidad de ejecución. El hecho de tener una curva de aprendizaje

bastante suave ha hecho que sea el lenguaje favorito para implementar las automatizaciones de diversas tareas de una forma sencilla [1].

Estas características - sumadas a la disponibilidad de diversas librerías de herramientas científicas, numéricas, de herramientas de análisis y estructuras de datos o de algoritmos de *machine learning* - además de ser *open source*, han hecho de Python un candidato ideal para tomar la delantera a lenguajes de análisis de datos como SAS o R. De las librerías más utilizadas podemos destacar NumPy, ScPy, Matplotlib, Pandas o Keras que están orientadas al *Data Science* [8].

Por todo lo que se ha mencionado anteriormente, Python es una alternativa ideal para poder realizar modificaciones y diversos prototipos de un sistema de una forma rápida, sencilla y poco costosa.

Figura 2.1: Logo de Python [8]

2.1.2. C++

C++ es un lenguaje de programación de alto nivel diseñado en 1979 por Bjarne Stroustrup como una extensión del lenguaje C, uno de los lenguajes básicos en el sector tecnológico. Con este lenguaje se añadió la capacidad de tener los mecanismos necesarios para poder manipular objetos. Al igual que en su antecesor, en C++ es posible realizar programación de bajo nivel, pudiendo llegar a la manipulación directa de la memoria. Se trata también de un lenguaje compilado, con lo cual el debe generarse código objeto a partir del código fuente, previamente a su despliegue en un determinado dispositivo. Esto, añadido a una curva de aprendizaje un tanto elevada y una depuración del código costosa hace que este lenguaje no sea apto para todo tipo de programadores [24].

C++ se utiliza en diversos ámbitos, aunque sin duda destaca en el diseño de sistemas que requieren de un alto rendimiento (High Performance Computing) y en sistemas em-

potrados con recursos limitados gracias a su alta eficiencia y las posibilidades de acceder a una programación a bajo nivel mencionadas anteriormente [12]. De igual manera este lenguaje cuenta con la implementación de numerosas librerías de código abierto, entre ellas OpenCV y Tensorflow, de las cuales profundizaremos más adelante. En la Figura 2.2 se puede observar la comparativa de productividad del programador y rendimiento en tiempo de ejecución del lenguaje entre Python y C++.

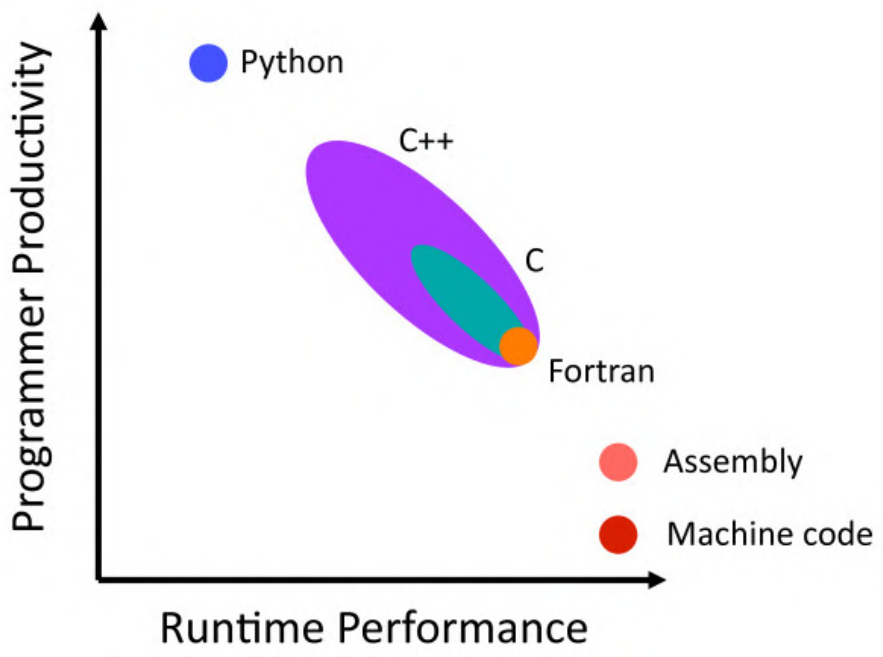


Figura 2.2: Comparativa entre Python y C++ [12]

2.2. Librerías Utilizadas en proyectos de IA y visión por computador

2.2.1. OpenCV

OpenCV, también conocida como *Open Computing Vision* es una de las librerías más utilizadas para la implementación de sistemas de visión por computador. Incluye varios cientos de funciones y algoritmos que son utilizados habitualmente para modelar este tipo de problemas. OpenCV esta disponible para la mayoría de sistemas operativos actuales como GNU/Linux, OS X, Windows, Android, etc.La primera implementación de esta librería se realizó en C pero debido a su popularidad en la versión 2.0 se migró el código de ésta a C++, además de añadirle nuevas funciones. A día de hoy la librería dispone de

una interfaz completa para su uso en otros lenguajes de programación como Java, Python o Matlab [11].

Como intento de maximizar el rendimiento de las tareas de visión, OpenCV da soporte a diversas tecnologías para ayudar a acelerar el flujo de trabajo.

- Multithreading utilizando **Threading Building Blocks**, facilita que un programa explote las capacidades de paralelismo de procesadores con arquitectura multinúcleo.
- Interfaces para el procesamiento en GPU, utilizando **CUDA** o **OpenCL**.

Las aplicaciones de OpenCV cubren diversas áreas como la segmentación y el reconocimiento, identificación de objetos, rastreo de movimientos, transformaciones de la imagen o detección de bordes. En la Figura 2.3 se puede observar la comparativa de aplicar distintos algoritmos para la detección de gradientes.

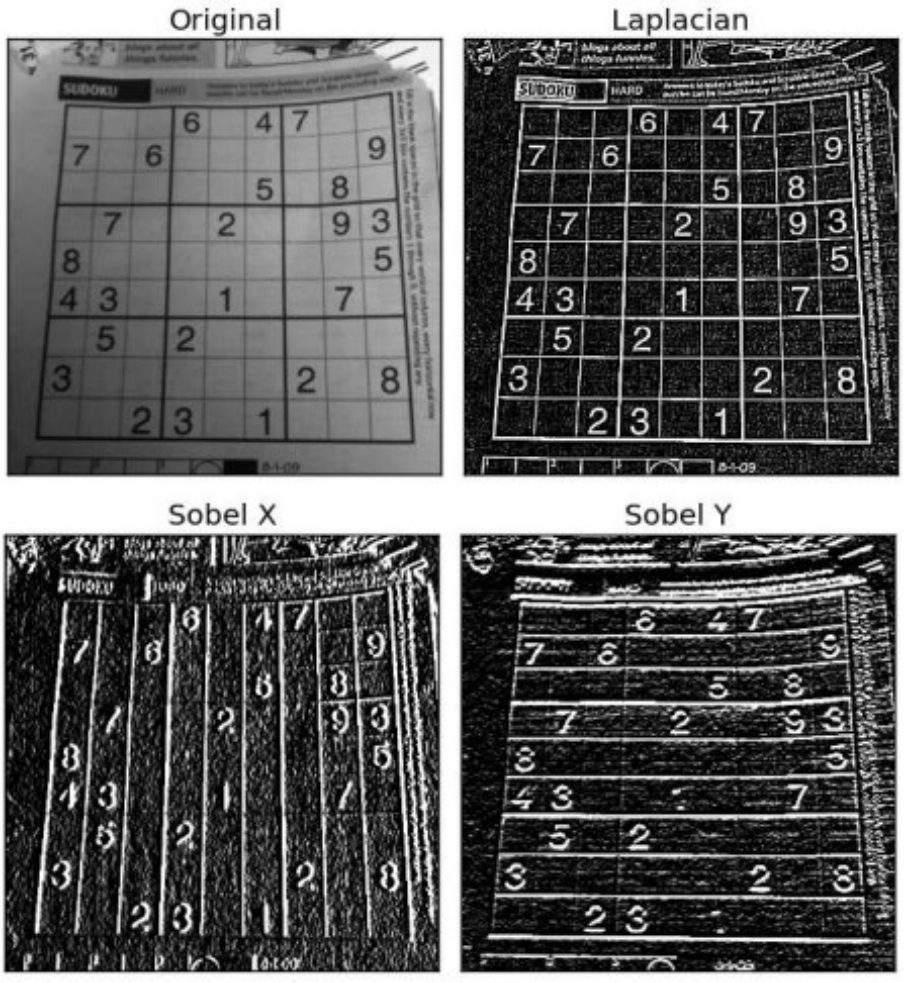


Figura 2.3: Detección de gradientes con varios métodos en OpenCV [4]

OpenCV está compuesta a su vez de varios módulos, los cuales contienen funciones relativas a un tipo de procesamiento de la imagen concreto. Podemos destacar *imgproc* para procesamiento y transformaciones geométricas de las imágenes o *features2d* para la detección de características 2d.

2.2.2. Tensorflow

Tensorflow es una biblioteca de código abierto para tareas de aprendizaje automático que fue desarrollada por Google para satisfacer sus necesidades en cuanto a la construcción y entrenamiento de redes neuronales.

En 2011 Google Brain, equipo de investigación de *deep learning* de Google, construye un sistema propietario de aprendizaje automático llamado DistBelief. El rápido crecimiento del uso de este sistema llevo a Google a asignar a distintos científicos de la computación como Jeff Dean para simplificar y reconstruir el código base de DistBelief en una biblioteca más rápida y robusta que terminó resultando en Tensorflow. En 2015 el código de esta biblioteca fue liberado como *open-source* y desde entonces se ha convertido en un ecosistema software muy completo, con una gran comunidad de usuarios. Se utiliza principalmente con la API en Python aunque también está disponible en otros lenguajes como C++ y Java. A pesar de esto, la mayor parte de las operaciones y el núcleo de TensorFlow están programados en C++ para una mayor velocidad de ejecución [25].

El nombre de Tensorflow deriva de las operaciones que tales redes neuronales realizan sobre arrays multidimensionales de datos conocidos como tensores. La estructura de la red neuronal se dispone en forma de grafo computacional donde cada nodo realiza una serie de operaciones a partir de los datos de los nodos anteriores. En la Figura 2.4 se puede observar una posible organización de un grafo de computación.

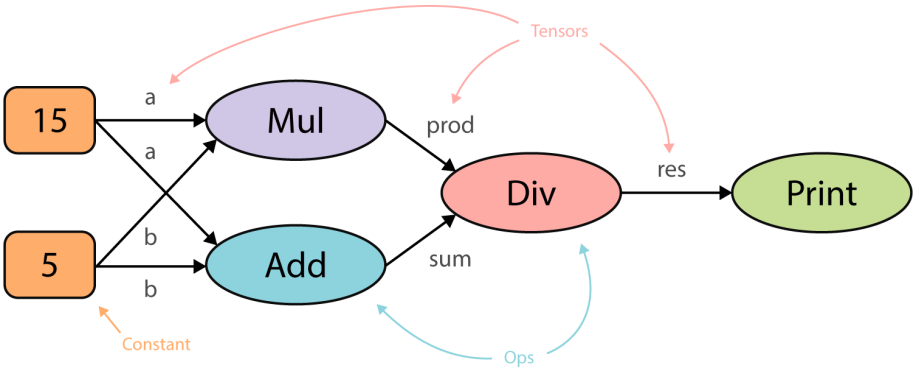


Figura 2.4: Grafo de computación [5]

La utilización de este framework para el diseño de redes neuronales simplifica en gran medida diversos aspectos de cara al programador:

- **Escalabilidad**: Mayor simplicidad en el diseño de redes neuronales con multitud de capas.
- **Cálculo de parámetros de manera automática**: Cálculo de la función de coste, backpropagation, actualización de pesos, etc...
- **Facilidad de ejecución y entrenamiento**: Abstracción al programador para la ejecución de la red en entornos distribuidos o con diversos tipos de aceleración basada en GPU.
- **Código optimizado**: El código que conforma la librería ha sido testeado y optimizado de forma exhaustiva.

2.3. Redes Neuronales Artificiales

Las redes neuronales artificiales son sistemas de computación inspirados en el funcionamiento y organización de las neuronas biológicas que constituyen los cerebros en el reino animal. Una red neuronal artificial es, por tanto, una colección de neuronas artificiales las cuales se interconectan entre ellas organizándose en capas.

La unidad mínima de estas redes son las neuronas, también conocidas como perceptrones. El perceptrón fue inventado en 1958 por Frank Rosenblatt y en un primer lugar su implementación estaba pensada para ser una máquina en vez de un programa, esta máquina estaba diseñada para hacer reconocimiento de imagen de manera que interconectaba un array de 400 células fotosensibles con la entrada de las neuronas. En una primera instancia a pesar de que la utilización del perceptrón parecía prometedora, no se tardo mucho en demostrar que este no podía ser entrenado para reconocer varias clases de patrones. La utilización de perceptrones de manera individual solo permite que estos sean capaces de aprender patrones de separación lineales [19]. Esta limitación en la tecnología hacía que estos perceptrones fueran incapaces de resolver el problema de clasificación de la función XOR. Esto provocó que la investigación en este campo se estancara durante muchos años, hasta la aparición del perceptrón multicapa, con el cual estructurando estos perceptrones individuales en diversas capas interconectadas se podía conseguir la clasificación de problemas más complejos.

Cada neurona o perceptrón Figura 2.5, de manera individual esta formada por:

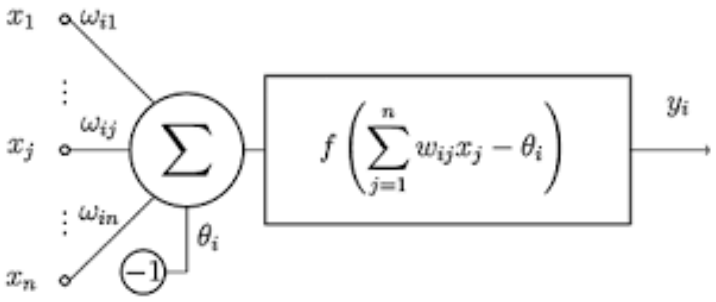


Figura 2.5: Perceptrón [26]

- **inputs** (X_i): Valores de entrada que recibe la neurona.
- **weights** (W_i): Están asignados individualmente a cada uno de los valores de entrada de la neurona. Suponen la influencia relativa de esa entrada para la neurona.
- **bias** (θ_i): También conocido como sesgo, es un valor de la neurona que determina como de predispuesta está esta de generar un 0 o un 1.
- **función de activación** ($f(x)$): La función de activación es una función que recibe como entrada una suma ponderada de la entrada anterior más el sesgo y la transforma en base a sus parámetros.
- **salida** (Y_i): Resultado de la función de activación con los parámetros de la neurona.

La organización de estas redes de neuronas artificiales se suele dar de forma que existe una de entrada, una de salida y una o varias capas ocultas. La capa de entrada recibe y procesa las señales de entrada y envía la señal de salida a la siguiente capa de neuronas de la red. Cada una de estas neuronas se pueden conectar a las neuronas de una de las capas adyacentes, como se ilustra en la Figura 2.6.

2.3.1. Funciones de activación

La función de activación es una función encargada de romper la linealidad en la salida de la neurona [15]. Habitualmente nos encontramos con los siguientes tipos de funciones:

- **Sigmoid**: Esta fue el primer tipo de función de activación en ser usada en los perceptrones.

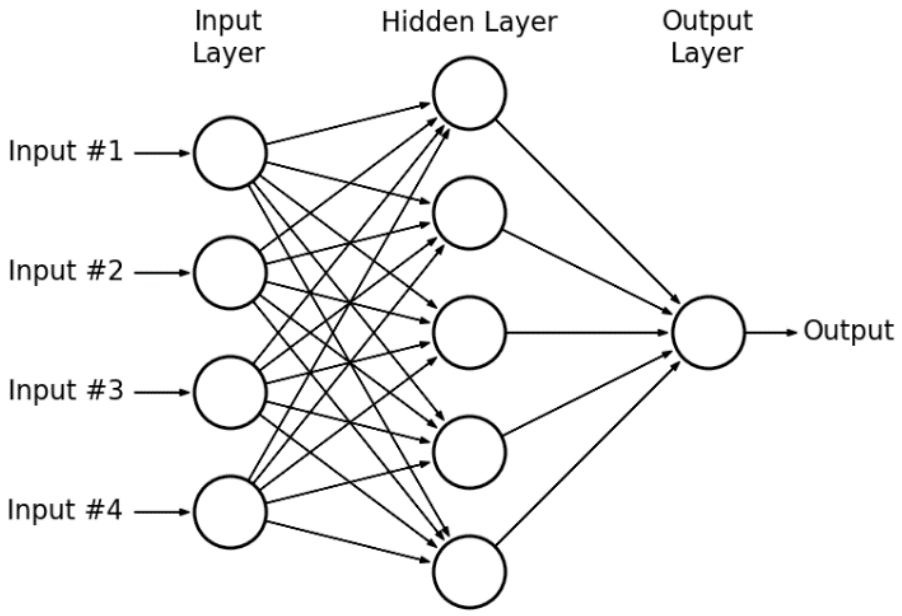


Figura 2.6: Perceptrón multicapa

Como ventaja podemos destacar que son muy útiles para modelar valores de probabilidad en valores de salida dado que el rango de esta función se encuentra contenido entre 0 y 1.

Por el contrario su empleo en las capas intermedias ha caído en desuso. Esta función de activación mata los gradientes para valores muy cercanos a sus extremos, además de que siempre toma valores estrictamente positivos, estas dinámicas pueden suponer un problema en la propagación de conocimiento por la red. Otra desventaja que se podría destacar, es la utilización del operador exponencial para obtener el valor de salida, lo cual no la convierte en una función computacionalmente eficiente.

- **Tanh**: Presenta una forma similar a la activación sigmoid, sin embargo podemos señalar varias diferencias.

  El rango de valores en los que actúa esta comprendido entre 1 y -1 por tanto no tenemos el problema de la no existencia de valores negativos propagándose por la red. Por el contrario sigue apareciendo el problema de saturación de gradientes para los valores cercanos a los extremos de su rango de actuación. Además sigue siendo ineficiente de calcular computacionalmente al tener que resolver cálculos exponenciales.

- **ReLu**: También conocida como Rectified Linear Unit, es una de las funciones más

utilizadas en la actualidad. La función devuelve 0 para valores inferiores a 0 y la función identidad para valores superiores.

Como ventajas con respecto a las anteriores podemos decir que no satura los gradientes para valores positivos, es computacionalmente muy eficiente y que supone una aceleración de hasta 6 veces en la convergencia de redes neuronales.

Como desventajas decir que este operador es frágil y puede morir durante el entrenamiento, si algún cambio en los pesos de la función provoca que no se pueda volver a sobrepasar el valor umbral 0. Este problema es posible controlarlo al utilizar *learning rates* pequeños durante el entrenamiento.

- **Variaciones de ReLu**: Existen varias alternativas similares a ReLu pero con la intención de solucionar el problema de la muerte neuronal. Los valores inferiores a 0 son multiplicados por un número. En el caso de la activación Leaky ReLu este valor es 0,01 y en el caso de la activación PReLu este valor es ajustado durante el entrenamiento de manera dinámica. Con esto se soluciona el problema de propagación de valores nulos por la red.

En la Figura 2.7 se puede observar la forma geométrica de estas funciones de activación.

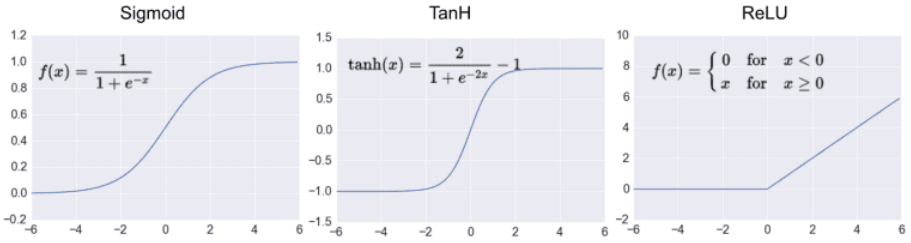


Figura 2.7: Funciones de activación

2.3.2. Entrenamiento de Redes Neuronales

Las redes neuronales son herramientas muy potentes a la hora de resolver problemas de clasificación o de búsqueda de patrones. Para llegar a obtener resultados, éstas son entrenadas de manera previa con un conjunto de ejemplos con los cuales ajustan los valores internos de sus neuronas. Este conjunto de datos debe contener un número de ejemplos suficiente y de calidad, es decir, que se ajusten al tipo de problema que se pretende resolver para poder asegurar un buen funcionamiento de la red.

Para conocer la eficacia del entrenamiento existen diversas métricas las cuales permiten conocer cómo de bien está actuando la red ante el problema que se esta tratando de solucionar. Cabe destacar que estas métricas se deben calcular en base a un subconjunto de ejemplos con el cual la red no haya sido entrenada, al que denominamos conjunto de *test*.

No obstante, durante el entrenamiento se debe emplear algún tipo de métrica para estimar el error que comete la red durante el entrenamiento. Este error de la predicción de la red con respecto al esperado, debe ser propagado a las neuronas en un proceso de asignar responsabilidades a cada una de ellas para que puedan ajustar sus valores con el objetivo de reducir dicho error; este proceso se llama *backpropagation*. Principalmente para calcular este error se utilizan funciones basadas en la diferencia entre dos distribuciones de probabilidad tanto para problemas binarios, donde destacamos *binary crossentropy*, o para problemas multiclase *categorical crossentropy*.

Existen otras métricas centradas en problemas de clasificación o de regresión con las que podemos estimar cómo de buenas son las predicciones de nuestro modelo. El empleo de un tipo de métrica u otra depende de si la variable objetivo a predecir es discreta o continua.

- **Clasificación**:
  - **Accuracy**: Representa la proporción del número de predicciones correctas entre el número total de predicciones.
  - **Precision**: Representa la proporción de ejemplos realmente positivos entre todos los positivos predichos.
  - **Recall**: Representa la capacidad del modelo para predecir ejemplos positivos, es decir, es el ratio de los ejemplos positivos correctamente clasificados.
  - **F1**: Esta métrica combina tanto precisión como *recall* utilizando la media armónica. Se utiliza con mucha frecuencia puesto que simplifica el rendimiento de un algoritmo de clasificación a una sola métrica.
- **Regresión**:
  - **MAE**: Error absoluto medio (del inglés *Mean Average Error*), es la diferencia en valor absoluto entre el valor real y el valor predicho.

- **MSE**: Error cuadrático medio (del inglés *Mean Squared Error*), es la media de la diferencia entre el valor real y el valor predicho al cuadrado.
- **RMSE**: Raíz del error cuadrático medio (del inglés *Mean Root Square Error*), es la raíz cuadrada de la media de la diferencia entre el valor real y el valor predicho al cuadrada.

Regularización de parámetros

Las redes neuronales son modelos con un gran poder de representación. Al contar con una alta cantidad de parámetros es posible que la red memorice de una manera bastante ajustada los datos de entrenamiento con los que se está entrenando, llegando a clasificarlos a la perfección. Este fenómeno se conoce como *overfitting* y se produce cuando un modelo pierde capacidad de generalizar en las predicciones con ejemplos con los que no ha sido entrenada. Existe también el caso en el cual la red no es capaz de aprender un patrón ya sea por razones de cantidad o calidad de los ejemplos que se le proporcionan o del diseño de la propia arquitectura de la misma. En la Figura 2.8 se puede ver cómo existe un punto óptimo en el error en la red para el conjunto de datos de prueba con el que no ha sido entrenado. Para controlar este fenómenos existe diversos métodos, de los cuales nos vamos a centrar en el *dropout*.

El *dropout* es una técnica mediante la cual, durante el entrenamiento de una red neuronal, se desactiva un número aleatorio de neuronas Figura 2.9. Estas neuronas que se desactivan van cambiando a lo largo de todo el entrenamiento. La contribución de las neuronas desactivadas no influye a la hora de estimar la predicción de la red en un determinado momento, por lo cual ésta se ve forzada a aprender diversas maneras de clasificar los datos de manera correcta. El efecto es que la red se vuelve menos sensible a valores específicos en las neuronas y una mayor generalización en las predicciones dadas [23].

2.4. Redes Neuronales Convolucionales

Dentro de las redes neuronales artificiales existes diversas variantes enfocadas a resolver problemas de distinto tipo. Las redes neuronales convolucionales son igual que las redes neuronales artificiales en cuanto a que están compuestas por neuronas que se van auto calibrando durante el entrenamiento, estas redes está inspiradas en la organización de los campos receptivos de las neuronas de la corteza visual primaria (v1) de un cerebro

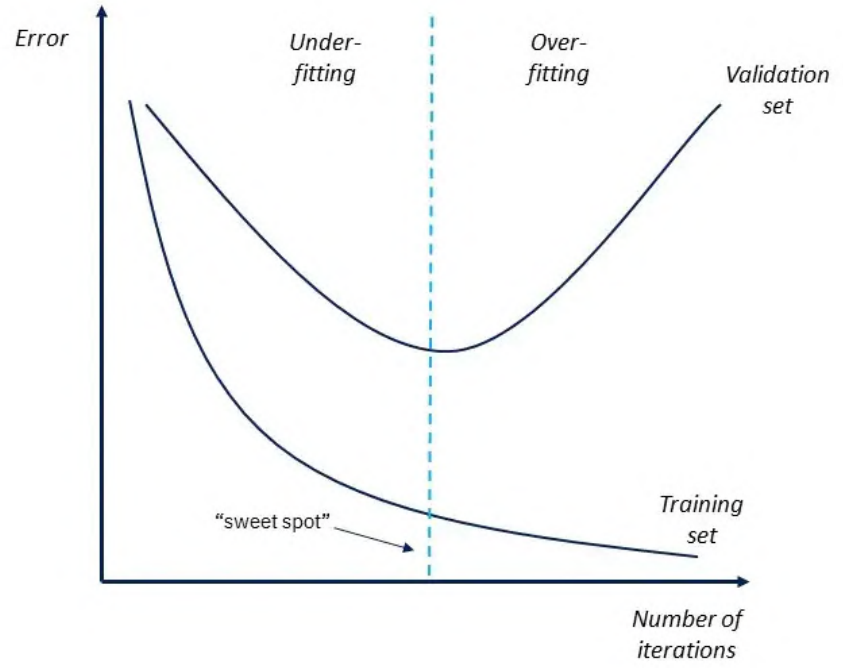


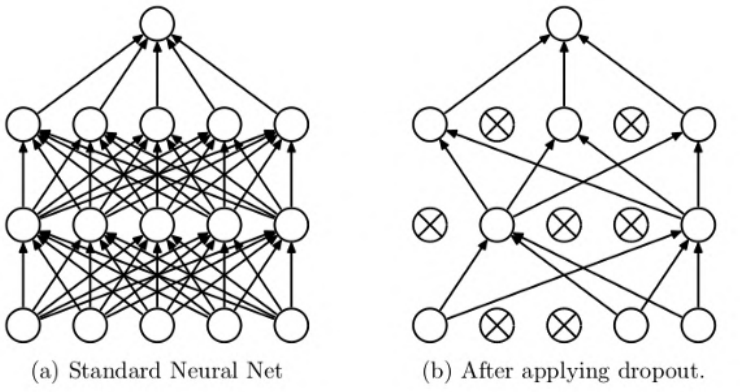
Figura 2.8: Underfitting y Overfitting

biológico humano. Es por esto que son utilizadas principalmente en aplicaciones de visión artificial.

El nombre de convolucionales le viene de la operación matemática lineal entre matrices homónima. En esta operación existen tres elementos: una matriz de entrada, un filtro o kernel y una matriz de salida. En la operación el kernel se desplaza a través de los valores de la matriz de entrada realizando una operación de multiplicación entre los valores correspondientes. Estos valores conformaran el resultado de la matriz de salida [16]. En la Figura 2.10 se puede observar esta operación. Las imágenes son entendidas como matrices en las que los valores de estas representan los valores de color de los píxeles de estas.

En los últimos años este tipo de redes ha tenido un gran crecimiento en campos relacionados con el reconocimiento de patrones, del procesamiento de imágenes al reconocimiento de voz.

Una de las ventajas que presenta la utilización de estas redes en lugar de los clásicos perceptrones multicapa es que estas redes son capaces de escalar con respecto al tamaño de las imágenes de entrada sin que suponga un esfuerzo computacional demasiado elevado, por ejemplo para procesar una imagen de 1920x1080 píxeles con una red tradicional se



++

Figura 2.9: Proceso de dropout [23]

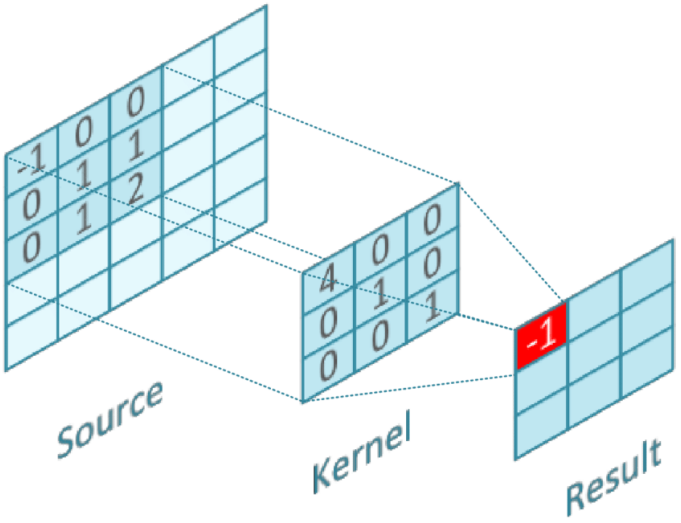


Figura 2.10: Convolución sobre una imagen [27]

necesitarían 2.077.440 neuronas sólo en la capa de entrada (hay que tener en cuenta que este valor se multiplicaría por la profundidad de color de la imagen [16]). Por otro lado en las redes neuronales convolucionales los parámetros de éstas están ligados a la cantidad de neuronas que conforman los kernels de convolución, número que es bastante menor y que no está ligado al tamaño de la imagen de entrada.

Otra gran ventaja es que la extracción de características no está ligada a una posición concreta de la imagen, como sí que lo estaría en un perceptron multicapa, lo que además también ayuda a reducir el *overfitting* de la red. Cada kernel de convolución es el encargado de la extracción de un tipo de característica distinta de la entrada.

Por lo general, el aprendizaje del conocimiento en este tipo de redes hace que las primeras capas que conforman la red aprendan patrones poco complejos - como formas

geométricas - haciéndose sucesivamente en las capas siguientes más complejos. En la Figura 2.11 pueden verse las características aprendidas en cada una de las capas para el reconocimiento de caras.

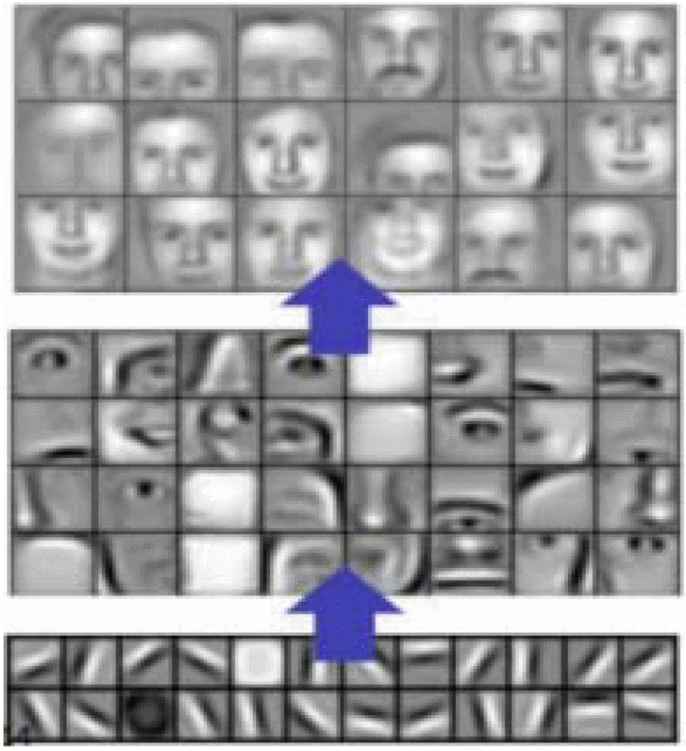


Figura 2.11: Características aprendidas para el reconocimiento de caras [27]

En las redes neuronales tradicionales sólo existe un tipo de capa. Sin embargo, en las redes neuronales convolucionales existen distintos tipos, entre las que destacan:

- **Conv2D**: Esta capa crea un kernel de convolución que aplica dicha operación a los datos de entrada para producir un tensor de salida. En ella se especifican diversos parámetros como la cantidad de kernels utilizados, el tamaño de éstos, el tipo de función de activación para configurar los valores del kernel, la forma en que se desplaza el kernel por la imagen...Podríamos decir que ésta es la capa por excelencia que le da su nombre a este tipo de redes.
- **Dense**: Ésta es la capa de neuronas tradicionales que podíamos ver en el perceptron multicapa, a la cual se le especifica la cantidad de neuronas y el tipo de activación de estas.
- **Flatten**: Esta capa se utiliza para transformar la forma del tensor de entrada a un vector unidimensional de salida con el mismo número de valores.

- **MaxPool2D**: Esta capa se utiliza para reducir el tamaño de la entrada en función de unos parámetros determinados. Estos parámetros se entienden como una ventana de un alto y ancho determinado la cual va a ir recorriendo la imagen de entrada y seleccionando únicamente el valor más alto contenido dentro de la misma. Se relacionan de forma directa por el tamaño de ventana. Si esta ventana es grande, también lo es la reducción espacial de la imagen resultante. Esta técnica es utilizada principalmente por varias razones: seleccionar los píxeles de la imagen con más información, reducir el ruido que pueda contener la misma y reducir la complejidad de los cálculos en siguientes etapas.

La combinación de este tipo de capas es lo que define la arquitectura de éstas. Generalmente en estas redes se intercalan capas Convolucionales con capas de MaxPooling de manera sucesiva, incrementando además el numero de kernels en cada capa convolucional. Para terminar se realiza un aplanamiento del tensor obtenido y se pasan esos valores a una serie de capas densas de las cuales su salida es el resultado de la clasificación. En la Figura 2.12 se puede observar la sucesión de capas que se dan en una arquitectura del tipo AlexNet, una de las arquitecturas referentes en la competición de clasificación de imágenes Imagenet.

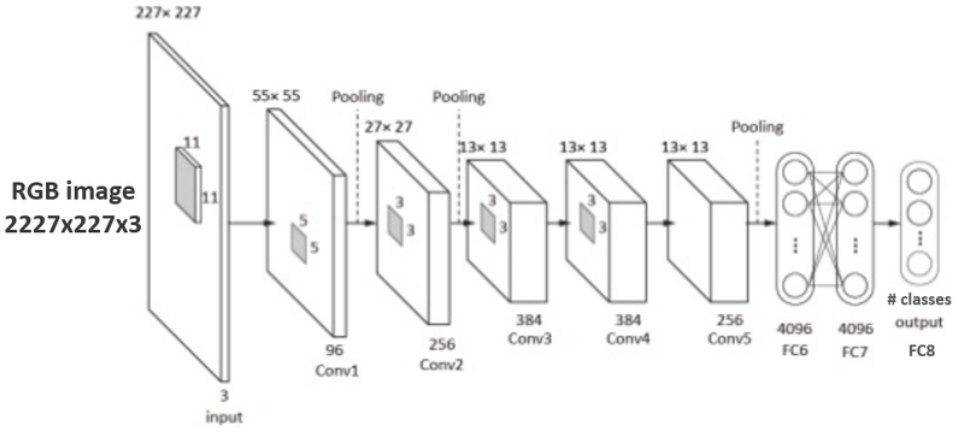
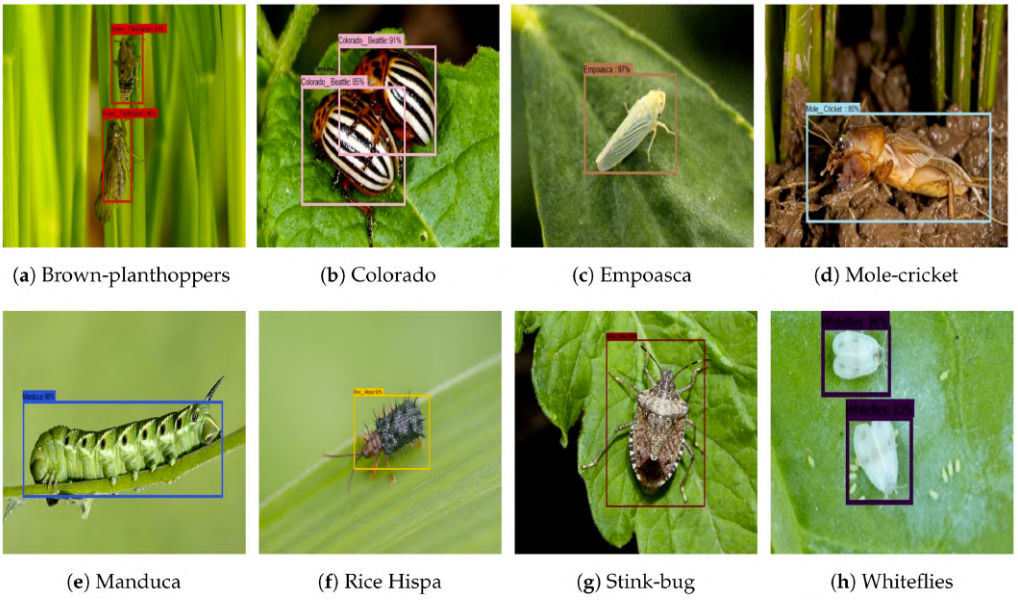


Figura 2.12: Arquitectura convolucional AlexNet

2.4.1. Casos de estudio

Redes neuronales empleadas en la clasificación de insectos

Las redes neuronales convolucionales han sido utilizadas de manera generalizada para la clasificación de diferentes tipos de imágenes. Entre estos tipos también encontramos problemas relacionados con el mundo de la agricultura como la identificación automática de diversas especies de insectos [21] [2] [13] [18] o de enfermedades relacionadas con los cultivos [2]. Estos trabajos pretenden resolver el problema en el que mediante la captura de imágenes se pueda identificar la presencia de determinados elementos de interés como insectos nocivos Figura 2.13 o enfermedades botánicas. Cabe destacar las dificultades en la adquisición de imágenes en este campo debido a que el cambio en las condiciones meteorológicas que pueden influir en las características lumínicas de las mismas.



(a) Brown-planthoppers (b) Colorado (c) Empoasca (d) Mole-cricket

(e) Manduca (f) Rice Hispa (g) Stink-bug (h) Whiteflies

Figura 2.13: Clasificación de insectos con CNN [18]

Redes neuronales empleadas en segmentación semántica

Otro de los campos de estudio de las redes neuronales convolucionales es la segmentación semántica de imágenes. La segmentación semántica es un proceso que busca asignar una etiqueta categórica a cada uno de los píxeles contenidos en una imagen [10]. Estas redes neuronales convolucionales se diferencian de las arquitecturas habituales de clasificación de imágenes en que su resultado tiene las mismas características espaciales que la entrada. Así, se definen entre otras arquitecturas del tipo Encoder-Decoder en la que la primera

parte de la red se encarga de encontrar patrones cada vez más complejos en la imagen al mismo tiempo que se va reduciendo en el plano espacial. La segunda parte de este tipo de redes hace el procesamiento contrario deshaciendo el proceso de reducción de la imagen para dar como resultado un mapa de segmentación de las características espaciales de la imagen de entrada Figura 2.14. El hecho de eliminar las capas densas de neuronas de la red hace que ésta contenga menos parámetros de entrenamiento y por tanto tarde menos en este procesamiento.

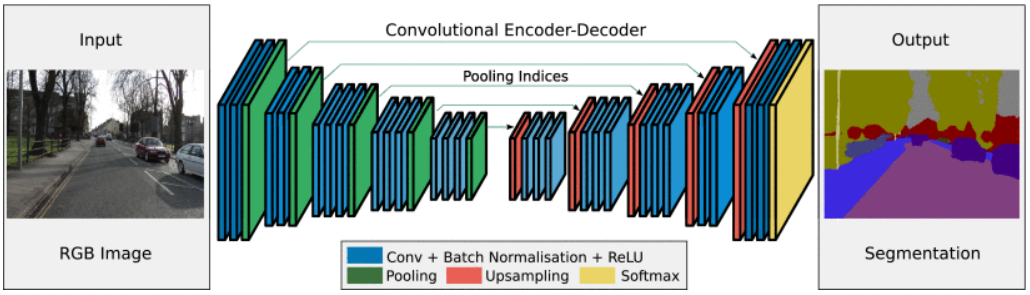


Figura 2.14: Arquitectura convolucional SegNet [3]

2.5. IoT y Smart Farming

El IoT es un concepto que hace referencia a la interconexión, haciendo uso de Internet, de distintos objetos cotidianos, los cuales contienen un pequeño procesador con que mejorar sus capacidades y funcionalidades. Las aplicaciones introducidas en estos dispositivos se centran principalmente en recoger datos del entorno, para procesarlos y realizar una estimación óptima de una serie de recursos o bien poder realizar un control remoto de estos dispositivos. Uno de los dispositivos IoT más famosos y utilizados son las placas basadas en un SoC ARM fabricadas por la compañía Raspberry Pi Foundation [9].

Con todo este ecosistema de tecnologías interconectadas se puede proporcionar un sistema de apoyo a la toma de decisión que podría suponer una mejora en el tiempo, recursos y costes empleados [7].

Por lo general, los dispositivos que funcionan bajo esta filosofía están enfocados a ser usados en lugares aislados, lo cual hace que sea poco probable que cuente con un suministro de energía eléctrica continuo. Estas características hacen que la opción más viable para poder alimentarlo sea mediante algún tipo de fuente de energía renovable la cual cargue una batería. De esta forma, no importarían las condiciones externas o la localización del nodo.

Al combinar estas baterías con un generador de energía renovable se puede conseguir que el sistema sea capaz de operar de forma completamente autónoma, realizando cargas parciales de la batería en momentos favorables (por ejemplo: generación fotoeléctrica de día o generación eólica en condiciones de viento) y supliendo las demandas energéticas del nodo en momentos no favorables gracias al uso de la misma. Aun así, por las condiciones climáticas que pueden ser cambiantes (p.ej. falta de viento o días nublados) esta energía ha de ser utilizada con la máxima eficiencia posible.

En este punto se nos plantean dos estrategias arquitecturales en cuanto a dónde procesar la información obtenida de los nodos IoT que recogen información del entorno para una determinada aplicación: de forma centralizada, en un centro de procesado de datos (p.ej. Cloud Computing) o en el propio nodo (*Edge Computing*).

Una arquitectura basada en Cloud Computing implica que los nodos no realizan ningún tipo de procesamiento de las imágenes que capturan del medio, teniendo que transmitir de esta forma la imagen al completo. Por el contrario una arquitectura basada en *Edge Computing* supondría realizar el procesamiento de la imagen *in situ* y transmitir únicamente los resultados obtenidos del propio procesamiento [17].

La ubicación de estos sistemas se suele ubicar en mitad del campo en los cuales no se puede garantizar el acceso continuo y de calidad a una red de comunicaciones. Este hecho, sumado al hecho de que en un sistema de este estilo alrededor del 80 % de la energía del mismo se destina a la transmisión de datos, hace que sea un requisito clave realizar una reducción de la cantidad de los mismos. Reduciendo los datos a transmitir además aseguramos que el nodo cumpla su función en un mayor número de ubicaciones de trabajo al no ser tan dependiente de la red de comunicaciones.

Capítulo 3

Objetivos y metodología de trabajo

En este capitulo se expondrán los objetivos que se pretenden lograr con la realización de este proyecto así como la metodología empleada en el mismo.

3.1. Objetivo general

El objetivo general de este proyecto consiste en el desarrollo de un sistema que implemente un algoritmo de procesamiento de imágenes que, mediante técnicas de segmentación e IA, realice una estimación del número de insectos atrapados en trampas cromáticas. Siendo más concretos, el proyecto se engloba dentro de la construcción de un sistema de mayor alcance, el cual va destinado a realizar un control sobre la plaga de la polilla de la vid.

Esta implementación se desplegaría en un sistema empotrado con capacidad de comunicación limitada y alimentado por batería. El procesamiento realizado reduciría la cantidad de información a transmitir, esto es, sólo zonas de interés de la imagen o datos numéricos sobre el tipo y número de insectos de un determinado tipo encontrados.

3.2. Objetivos específicos

3.2.1. Análisis de las tecnologías potenciales a utilizar

Teniendo en cuenta cuáles son los requisitos funcionales y no funcionales que tiene que cumplir nuestro sistema, se tiene que analizar qué tecnologías serían las más adecuadas

para su implementación en el proyecto. Es necesario evaluar el rendimiento proporcionado por estas tecnologías en relación con el rendimiento ideal que vamos a necesitar.

3.2.2. Estudio de las técnicas de análisis de imagen mediante IA

Estudio de la viabilidad de aplicación de diferentes tecnologías basadas en IA para poder realizar la segmentación y análisis de las imágenes obtenidas de las trampas cromáticas.

A partir de estas imágenes se deberá realizar un procesamiento que proporcione como resultado una estimación de la presencia o no de *Lobesia Botrana* en las imágenes.

Python es un lenguaje de programación ampliamente conocido tanto por su versatilidad, como por su fácil curva de aprendizaje. Estas características lo convierten en un candidato ideal para poder llevar a cabo de forma rápida un prototipo del propio sistema, pudiendo variar los parámetros del mismo e ir haciendo comprobaciones de los resultados obtenidos mediante el método de ensayo y error.

3.2.3. Adquisición de imágenes y creación del conjunto de datos de entrenamiento

Los algoritmos de IA que sirven para la clasificación y segmentación de imágenes suelen ser implementados a partir de un entrenamiento supervisado con un conjunto de ejemplos que le permitan aprender una serie de patrones dados.

Por esto es clave obtener una serie de muestras representativas y generar un conjunto de datos para entrenar estos algoritmos. Las técnicas de *data augmentation* suelen ser claves en la consecución de este paso.

3.2.4. Adaptación del algoritmo a un sistema empotrado

Python es una buena opción por su rapidez a la hora de poder prototipar, pero no destaca por su velocidad ni eficiencia energética. C++ es un lenguaje ampliamente utilizado en sistemas empotrados por su eficiencia y capacidad de adaptación.

En nuestro caso es un requisito crítico que el sistema consuma poca energía. Por lo que tendremos que adaptar la funcionalidad obtenida en el prototipo para que desempeñe su labor con la máxima eficiencia posible.

3.2.5. Optimización del modelo de IA para su utilización en un sistema empotrado

Los sistemas basados en inteligencia artificial suelen ejecutarse con el apoyo de aceleración hardware como GPU debido a la alta cantidad de parámetros que contienen éstos y que hacen que el esfuerzo computacional requerido y necesidades de memoria sean extraordinarios. Para realizar la inferencia de un modelo a partir de una entrada se deben realizar una alta cantidad de operaciones en punto flotante (Float16 y Float32) sobre matrices. Las CPU (como la presente en una Raspberry Pi) utilizadas habitualmente en la implementación de sistemas empotrados, no están preparadas para realizar operaciones con este tipo de datos de manera eficiente.

Es clave la adaptación de este modelo en base a técnicas de cuantificación que transformen los tipos de datos que se utilizan internamente de punto flotante a entero.

3.2.6. Analizar desempeño y valoración de requisitos no funcionales como consumo o recursos utilizados

Por ultimo, hemos de tener en cuenta los recursos utilizados en nuestro modelo, así como los tiempos de ejecución. Y en caso de que alguno de esos parámetros no sean satisfactorios volveríamos a la etapa de optimización del algoritmo.

3.3. Metodología

Para las cuestiones organizativas del proyecto se ha optado por escoger un desarrollo iterativo-incremental debido a la posibilidad de dividir el proyecto en distintas fases. Gracias a esta división, el coste de cada iteración es menor que el de asumir el proyecto completo, además de ofrecer una visión más clara a los desarrolladores sobre a qué tareas se debe asignar una prioridad más alta [22].

Esta metodología de desarrollo se creó en respuesta a las debilidades del modelo tradicional en cascada. La idea principal es llevar a cabo la gestión del proyecto de manera incremental, permitiendo al desarrollador ganar ventaja con el aprendizaje progresivo durante el desarrollo, incrementando poco a poco la funcionalidad de versiones resultantes del sistema Figura 3.1.

Los pasos claves en el proceso son: comenzar con una implementación simple de los requisitos del sistema y sucesivamente mejorar la secuencia evolutiva de las versiones hasta

que el sistema completo esté implementado. En cada iteración se realizan cambios en el diseño y se agregan nuevas funcionalidades y capacidades al sistema. De esta forma se consiguen proporcionar resultados anticipados y orientar el proyecto hacia los objetivos reales deseados. Los objetivos se pueden priorizar en base a criterios como: el valor de negocio, la cantidad de esfuerzo necesario para realizar una iteración o el riesgo posible.
La metodología iterativa-incremental consta de 3 etapas:

Etapa de inicialización: En esta etapa inicial se crea una "versión" del sistema, es decir, se establece como objetivo la creación de un producto con el que el usuario pueda interactuar. El fin es obtener información (p.ej. del usuario final) para retroalimentar con posibles mejoras el mismo proceso. Debe ofrecer una muestra de los aspectos claves del problema y proveer una solución lo suficientemente simple para ser comprendida e implementada fácilmente. Para guiar el proceso de iteración se crea una lista de control del proyecto la cual contiene todas las tareas que deben ser realizadas. En la fase de análisis de cada una de las iteraciones esta lista de control puede ser revisada y modificada en función de requisitos cambiantes en el sistema.

Etapa de iteración: Esta etapa involucra el diseño e implementación de cada tarea de la lista de control del proyecto. La meta aquí es que la implementación sea capaz de soportar el rediseño o la adición de una nueva etapa. Cuanto mas avancen las iteraciones, mas fáciles deben de ser las modificaciones que se añadan al proyecto. De no ser así, existe un problema en el diseño general del mismo.

Lista de control de proyecto: En esta etapa se revisa la lista de control y se establece si se han completado todas las tareas establecidas.

Dentro de cada iteración podemos diferenciar 4 fases:

Análisis: En esta fase se realizará una descripción de requisitos, la cual ayude al equipo de desarrollo en las siguientes etapas.

Diseño: En esta fase se realizará el diseño de la solución correspondiente a los requisitos anteriormente identificados. Como resultado de esta fase, es posible que se generen múltiples soluciones al mismo problema. Es importante tener en cuenta que durante el ciclo de desarrollo pueden producirse cambios en los requisitos que no se tuvieron en cuenta en la fase de análisis.

Desarrollo: En esta fase se producirá una implementación del diseño realizado anteriormente. Consideraremos la solución como valida si se consigue una solución satisfactoria que cumpla todos y cada uno de los requisitos planteados.

Pruebas: El objetivo de esta fase es tratar de encontrar fallos en la implementación antes de dar comienzo a la siguiente iteración. Como esta fase supone el final de una iteración, es necesario realizar pruebas sobre partes de iteraciones anteriores en las cuales podría haber cambiado de manera potencial el resultado.

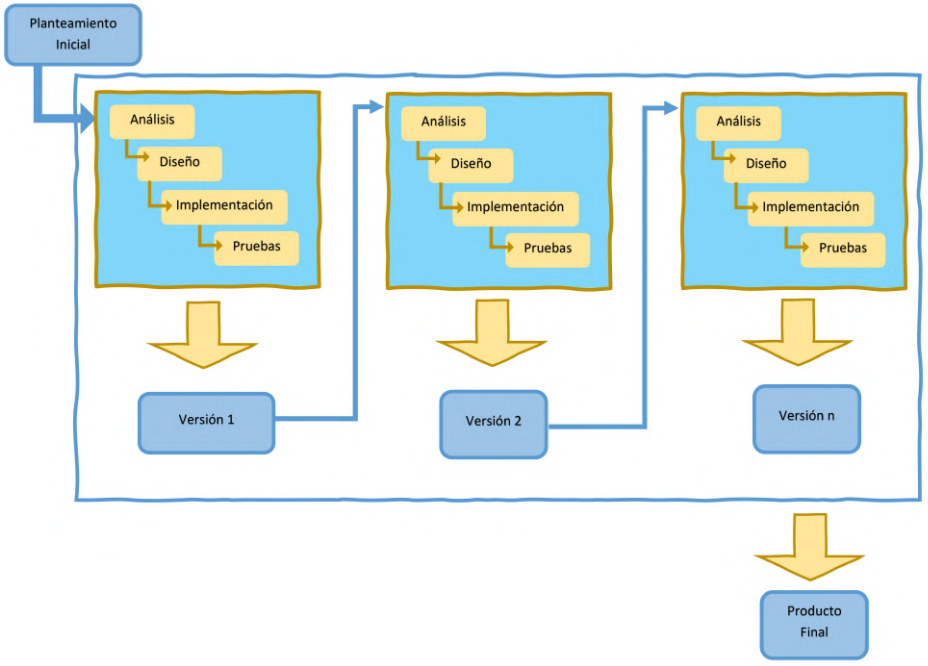


Figura 3.1: Metodología Iterativa Incremental

3.4. Herramientas utilizadas

3.4.1. Herramientas Hardware

- **Ordenador**: AMD Ryzen 7 3700X, Nvidia Geforce RTX 3070, 64 GB RAM @ 3200MHz, 1 TB SSD. Equipo empleado en el prototipado y entrenamiento del modelo de IA.
- **Raspberry Pi 3B**: Equipo empleado en las pruebas de despliegue del sistema.
- **Raspberry Pi 4 4GB**: Equipo empleado en las pruebas de despliegue del sistema.

- **Raspberry Pi Zero 2 W**: Equipo empleado en las pruebas de despliegue del sistema.
- **USB voltaje meter**: Dispositivo de medición de consumo eléctrico USB empleado en la estimación de consumo de las placas Raspberry Pi.

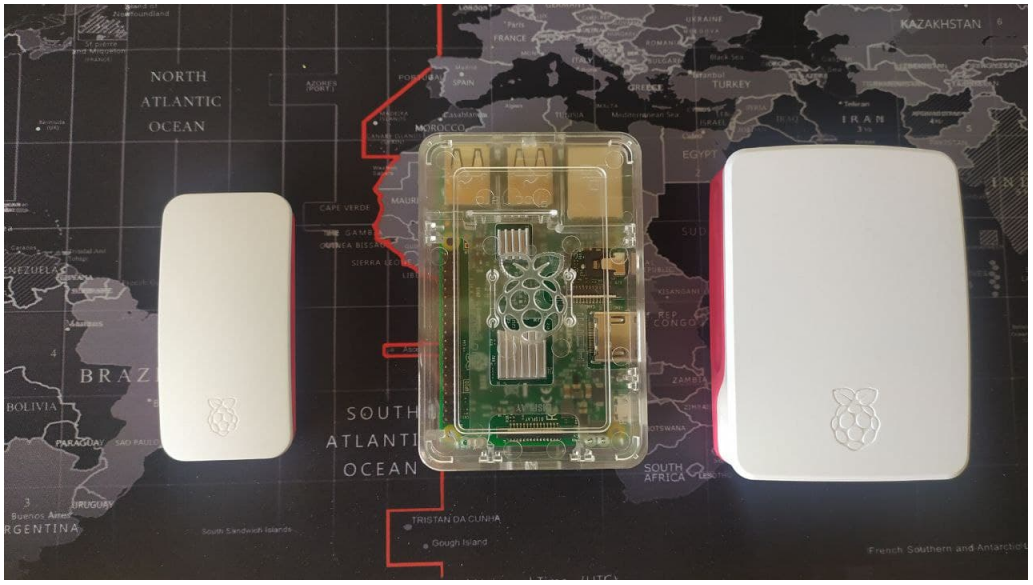


Figura 3.2: Sistemas Raspberry Pi utilizados en las pruebas y validación

3.4.2. Herramientas, lenguajes y tecnologías Software

- **PyCharm Community Edition**: IDE centrado en el lenguaje de programación Python y gestion de entornos virtuales.
- **Visual Studio Code**: IDE multilenguaje utilizado para el desarrollo del prototipo en C++.
- **OpenCV**: Librería multilenguaje centrada en la manipulación de imágenes.
- **Tensorflow**: Librería multilenguaje centrada en la creación de modelos de IA.
- **Gimp**: Software de edición fotográfica.
- **Overleaf**: Editor en línea de proyectos LaTeX.
- **www.generadordegraficos.com**: Pagina web para realizar gráficos descriptivos.

3.4.3. Herramientas de gestión de proyectos

- **Bitbucket**: Servicio de alojamiento basado en web para proyectos que utilizan sistemas de control de versiones que puede estar basado en Mercurial o en Git.
- **Trello**: Software de administración de proyectos con interfaz web basado en la metodología kanban.
- **Microsoft Project**: Software de administración de proyectos simplificando el seguimiento y la estimación temporal de los mismos.
- **Microsoft Teams**: Plataforma unificada de comunicación y colaboración que combina chat persistente, reuniones en vídeo, almacenamiento de archivos e integración de aplicaciones.
- **SmartGit**: Cliente Git para la gestión de repositorios locales y remotos.

Capítulo 4

Desarrollo del trabajo

En esta sección se describirá la aplicación del método de trabajo presentado en la Sección 3.3, mostrando los elementos (modelos, diagramas, especificaciones, etc.) más importantes.

Este apartado explica cómo el empleo de la metodología permite satisfacer tanto el objetivo principal como los específicos planteados en el TFM, así como los requisitos exigidos (según exposición en Sección 3.1).

4.1. Planteamiento del Proyecto

En esta etapa se van a estudiar las características del proyecto que pretendemos abordar con tal de determinar los requisitos funcionales y no funcionales.

4.1.1. Captura de Requisitos

En primera instancia vamos a realizar una captura de requisitos sobre el problema que vamos a solventar. Con esto podremos conocer con más precisión el alcance del proyecto.

Requisitos Funcionales

1. **Detección de *Lobesia Botrana* basado en imágenes**: El nodo IoT en el que se integrará el componente a desarrollar estará provisto de una cámara que realizará una captura del estado de la trampa cada cierto tiempo. El sistema utilizará estas imágenes como entrada para realizar una estimación de la cantidad de *Lobesia Botrana* atrapada e identificar las zonas de interés de la imagen.

2. **Estimación precisa**: El sistema debe ser capaz de realizar una estimación lo más precisa posible de la cantidad de insectos en las imágenes. Para ello se evaluarán diferentes alternativas con las que enfrentarnos al problema y se elegirá la más adecuada.

3. **Reducción de la cantidad de información a transmitir**: La comunicación (envío/recepción) de datos es la tarea que más consume en un nodo IoT. Por lo tanto, y siguiendo la filosofía de una arquitectura basada en *Edge Computing*, se procurará reducir al mínimo posible la cantidad de información a enviar, para que esta sea posteriormente analizada en mayor profundidad en unos servidores centrales.

Requisitos No Funcionales

1. **Eficiencia energética**: El nodo IoT, al estar alimentado por baterías, debe hacer un uso eficiente de los recursos. Para esto deben plantearse diversas alternativas y evaluar la relación eficiencia-rendimiento de las mismas.

2. **Utilización de recursos hardware reducidos**: Los dispositivos IoT cuentan con recursos hardware limitados, por tanto se deben realizar las optimizaciones necesarias en la complejidad computacional del sistema para poder ser ejecutado en estos dispositivos.

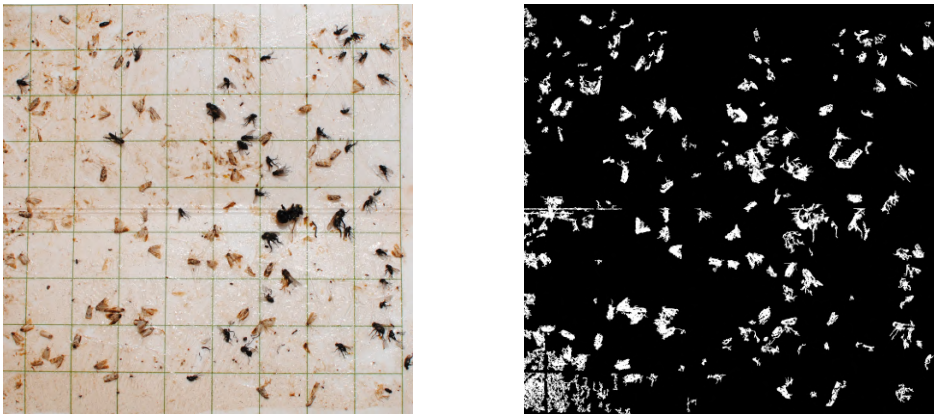
3. **Coste reducido de los dispositivos**: La viabilidad de implantación del sistema para su uso agrícola a gran escala depende del coste del mismo. Los dispositivos y sistemas empleados en el proyecto deben ser asequibles.

4.1.2. Planificación del proyecto y división del trabajo

Una vez recopilados los requisitos, el siguiente paso consistirá en investigar las técnicas y tecnologías actuales relacionadas con la detección e identificación de insectos utilizando técnicas de visión por computador. Después de esto, podremos realizar una división de los hitos que irán guiando el proyecto y una primera estimación del tiempo necesario para la realización de los mismos en función de los recursos que tenemos a nuestra disposición.

Previamente se han explorado varias alternativas para poder solucionar el problema de conteo de insectos de manera automática. En primer lugar se planteó la posibilidad de establecer unos parámetros fijos de procesamiento en la imagen mediante el uso de combinaciones de diferentes operaciones como conversiones del espacio de color, ecualizaciones

del histograma, funciones de binarización y operaciones morfológicas. La idea de este *pipeline* de procesamiento era la de diseñar un algoritmo que fuese capaz de segmentar la imagen, identificando las regiones de interés que pudieran contener *lobesia botrana*. A estas regiones posteriormente se aplicaran técnicas de machine learning para determinar de manera definitiva cuáles de estas regiones se tendrían en cuenta para el conteo de insectos final [20]. Sin embargo tras varias pruebas se desechó esta manera de afrontar el problema debido a la presencia de diversos tipos de insectos y polvo adheridos en la trampa cromática, los cuales suponían una componente de ruido que imposibilitaba el establecimiento de unos parámetros comunes a las imágenes para el diseño del filtro Figura 4.1.



(a) Imagen original **(b)** Segmentación manual con OpenCV

Figura 4.1: Ejemplo de segmentación de imágenes con operaciones en OpenCV

Tras estos resultados se decidió realizar el procesamiento del problema mediante la aplicación de redes neuronales convolucionales centradas en la segmentación de imágenes. Por tanto, el primer paso a llevar a cabo consiste en la creación de un conjunto de datos con el cual poder entrenar una red de este estilo.

4.2. Preparación de los datos

El sistema está diseñado para funcionar a partir de una serie de imágenes de entrada provenientes de trampas cromáticas de insectos, desplegadas en una explotación agrícola. Estas características hacen que al estar en un medio no controlado, las imágenes de entrada puedan presentar diferencias en los colores y la intensidad de la luz en función de la meteorología de cada momento.

Con respecto a las imágenes, se ha establecido que su resolución de entrada al sistema sea fija, con tal de establecer una cadena de procesado de datos la cual no sea variable

en el tiempo. Gracias a estos parámetros fijos podemos definir unos recursos uniformes en el procesado de todas las imágenes. Las imágenes han sido tomadas desde una cámara Nikon D300 a una distancia de 50 centímetros de las trampas cromáticas, obteniendo una resolución de 3872x2592 píxeles. El conjunto de datos a la hora de afrontar el problema esta formado por 7 imágenes de trampas cromáticas desplegadas en una explotación de vid de Castilla La Mancha Figura 4.2.

(a) Conjunto de trampas cromáticas disponible

(b) Imagen capturada de una trampa cromática

Figura 4.2: Ejemplo de imágenes del conjunto de datos

4.2.1. Recorte de las imágenes

Una de las decisiones tomadas a la hora de realizar el procesamiento de los datos ha sido el recorte de las imágenes de entrada de manera que solo quedase presente en ellas

la región relativa a la propia trampa cromática. La eliminación de partes externas a la trampa cromática en la imagen repercute en una disminución de la confusión a la hora de entrenar la red neuronal para realizar la segmentación.

La resolución de entrada elegida para el recorte de las imágenes ha sido de 1800x1800 píxeles, la selección de una imagen con el mismo número de píxeles de ancho y de alto supone una simplificación en los parámetros de diseño de las capas de la red neuronal. Esta resolución ofrece un compromiso entre calidad y precisión de la detección. Cabe destacar que el tamaño de las imágenes repercute de forma directa sobre la cantidad de recursos necesarios (tiempo de ejecución y memoria) para el entrenamiento e inferencia de la red. Otro punto a tener en consideración es que la calidad de inferencia de la red está vinculada al tamaño que ocupen las áreas correspondientes en la imagen a los insectos, por tanto realizar una predicción sobre imágenes con tamaños relativos distintos a los del conjunto de entrenamiento puede suponer un mal funcionamiento del sistema. Más adelante se expondrá la diferencia entre la selección de distintos tamaños de imagen para entrenar al sistema.

En la Figura 4.3 se puede observar el resultado de la aplicación de este recorte a la zona de interés.

4.2.2. Ground Truth

Uno de los requisitos necesarios para poder entrenar una red neuronal es contar con un conjunto de datos etiquetado. Para los problemas de segmentación semántica este valor se conoce como *ground truth*, el cual es simplemente el resultado que se espera obtener para cada uno de los píxeles de la imagen de entrada. Es por esto que el *ground truth* debe tener las mismas dimensiones espaciales que la imagen de entrada a excepción de la profundidad, la cual estará determinada por el número de posibles etiquetas para cada píxel. Nuestro problema es de clasificación binaria, pertenencia o no pertenencia a la clase *lobesia botrana*, con lo cual el *ground truth* tendrá una sola dimensión, indicando con un valor de 0 o 1 la pertenencia a la clase objetivo.

Tras consultar con un experto sobre el tema para determinar cuáles son los parámetros que caracterizan a la *lobesia botrana*, se ha generado un mapa de segmentación para cada imagen con el software de edición fotográfica Gimp (ver Figura 4.4). Aquí se han seleccionado las regiones pertenecientes a este tipo de insecto estableciendo el color representativo de sus píxeles en blanco y el del resto en negro Figura 4.5.

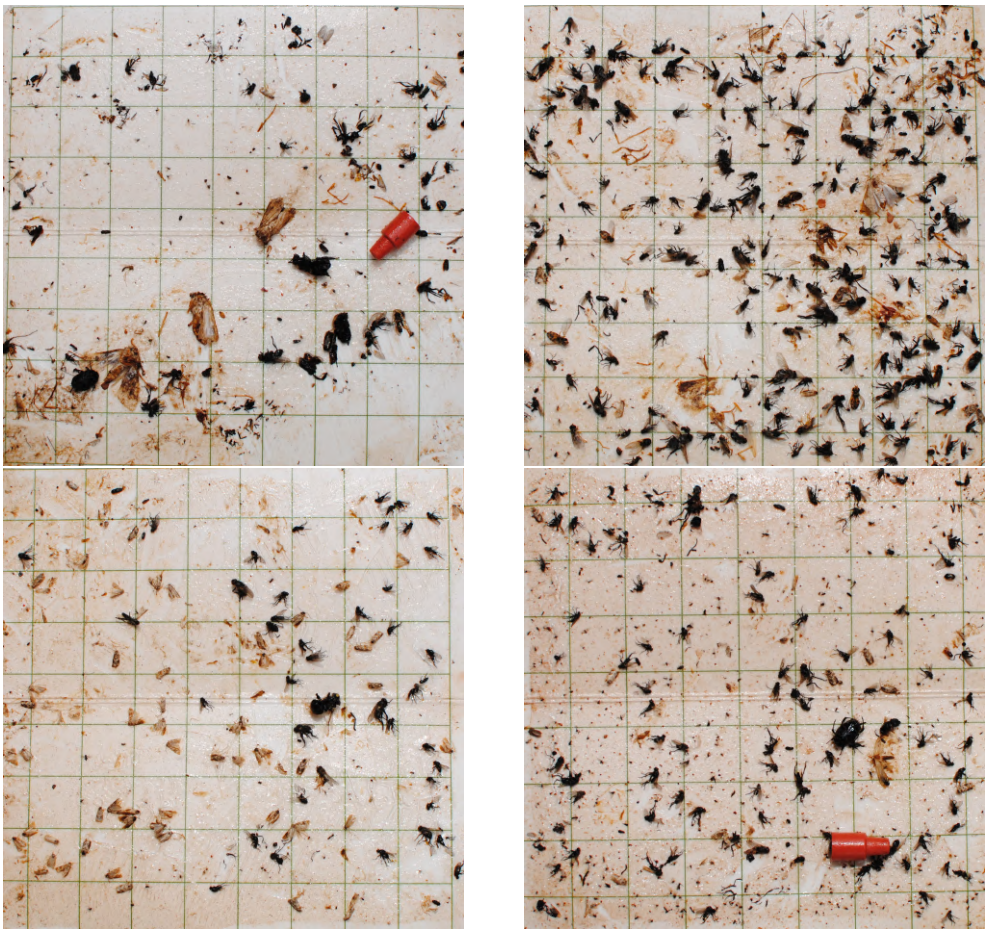
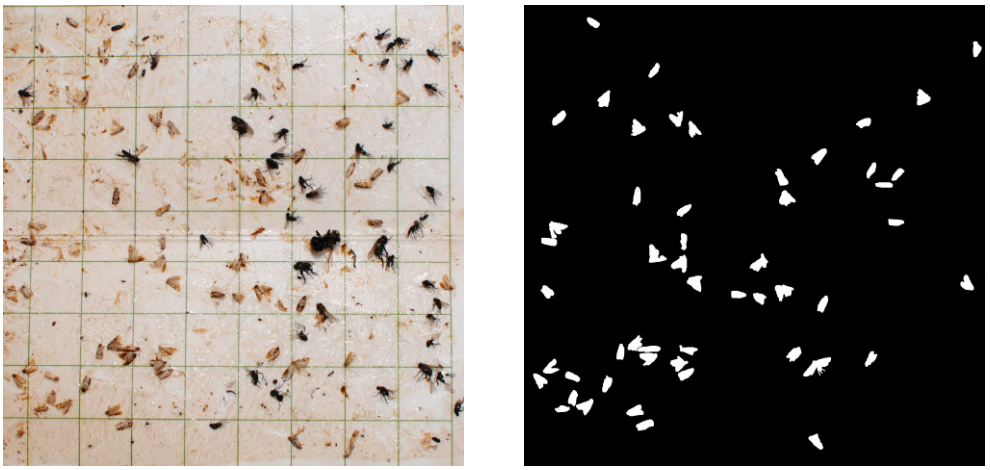


Figura 4.3: Ejemplos de imágenes tras recorte

Figura 4.4: Creación del *ground truth* con Gimp



(a) Imagen original **(b)** *Ground truth*

Figura 4.5: *Ground truth* de una imagen

4.2.3. *Data Augmentation*

El proceso de entrenamiento de redes neuronales requiere de una alta cantidad de ejemplos con los que alimentar a la red para que ésta vaya realizando ajustes en sus parámetros y conseguir obtener el resultado esperado. Al no contar con un amplio conjunto de datos con el cual realizar este proceso se debe recurrir a técnicas de ampliación mediante la generación sintética de estos. En el caso de conjuntos de datos de imágenes se suele recurrir a transformaciones sobre las mismas. Podemos destacar algunas operaciones como las de espejo o las de rotación. En las operaciones de espejo se cambia la posición dentro de un eje de los píxeles de la imagen, de manera que los que se encontraban en el límite superior pasan a ubicarse en el inferior y viceversa. Por otra parte las operaciones de rotación, como su propio nombre indica, consisten en rotar la imagen el número de grados que se le especifique.

Cabe mencionar que al realizar una operación de rotación de la imagen, esta sufre un desplazamiento de modo que parte de los píxeles se quedan fuera de la propia región de la imagen dejando otras partes con píxeles en negro sin valor. Esta falta de información en parte de la imagen puede suponer una entrada de ruido al proceso de entrenamiento. Para evitar esta situación, se ha optado por aplicar el criterio de expansión del primer píxel con valor encontrado al resto de píxeles de su misma columna, de esta manera se evita que existan zonas de la imagen con valores atípicos.

Para incrementar el conjunto de datos se ha tomado como criterio una combinación de las operaciones de espejo y rotación. De esta manera, por cada imagen se realiza la

operación de espejo en cada uno de sus ejes además de rotarlas en intervalos de 3º. Este incremento del conjunto de datos provoca un desacoplamiento entre la orientación espacial de los insectos en la imagen con respecto a su identificación. El factor de incremento obtenido es de 240 imágenes generadas por cada imagen original. En la Figura 4.6 se pueden ver algunos ejemplos de la generación de datos sintéticos a partir de una imagen.

Este proceso debe ser realizado de manera análoga con los *ground truths* generados, de manera que para cada imagen se tiene su mapa de segmentación correspondiente.

(a) Imagen original **(b)** Imágenes generadas

Figura 4.6: *Data Augmentation*

4.3. Diseño red neuronal

Tras establecer el conjunto de datos con el que vamos a entrenar la red neuronal hay que definir la arquitectura que le va a dar forma. El diseño de las redes neuronales convolucionales está determinado por la utilización de diferentes capas interconectadas entre sí. Cada una de estas capas esta especializada en un tipo de operación diferente que realiza sobre los datos de entrada. Uno de los pilares básicos en este diseño es la combinación de operaciones de reducción espacial y de convolución de manera consecutiva, aunque existen más operaciones que intervienen en este proceso. La selección de los parámetros que van a definir las operaciones que se realicen en cada una de estas capas determinarán la potencia de la red neuronal para ajustarse a las características de los datos de entrada y por tanto tener la capacidad de realizar buenas predicciones sobre los mismos.

La necesidad principal de este proyecto es la segmentación de imágenes, por tanto se ha tomado la decisión de escoger la utilización de una arquitectura del tipo Encoder-Decoder

(ver Sección 2.4.1). Además, el diseño de la arquitectura de la red está inspirado por otras que ya han sido probadas en el campo de la clasificación y segmentación de imágenes con buenos resultados como pueden ser AlexNet y SegNet. Otra de las fuentes de inspiración de esta arquitectura es la utilización de este tipo de redes neuronales en dispositivos IoT con recursos limitados como pueden ser LeNet-5 o MobileNet.

A pesar de esto se ha desechado la idea de utilizar técnicas de transferencia de conocimiento a partir de otras redes neuronales ya entrenadas debido a que ninguno de estos modelos se ha entrenado con algún dataset de imágenes parecido al que tenemos en este proyecto. Ésta es la razón por la que se piensa que el empleo de estas técnicas no hubiesen aportado ningún beneficio.

En la Figura 4.7 se puede observar la arquitectura que se ha seleccionado para llevar a cabo la implementación del sistema desde una perspectiva en 2 dimensiones. En rosa encontramos las capas Conv2D, en verde las capas MaxPooling2D, en azul claro las capas UpSampling2D y en azul oscuro las capas Conv2DTranspose. La implementación de la red se ha realizado utilizando Keras y TensorFlow debido a su simplicidad a la hora de definir la estructura de esta.

Podemos diferenciar dos partes que componen la arquitectura de la red, que se definen a continuación:

- **Encoder**: Esta parte de la arquitectura se encarga de iniciar la extracción de características a partir de las imágenes de entrada. En ella se alternan capas de convolución y capas de reducción espacial, de manera que los datos generados van incrementando progresivamente sus dimensiones a la par que reducen su tamaño espacial.

  - **Capas Conv2D**: Capas convolucionales de la red, son un total de 5, las principales características que podemos destacar de estas son los tamaños de los filtros empleados en las mismas y la profundidad de los datos que se generan en cada una. Se ha escogido una activación del tipo "relu" para ajustar los parámetros de los filtros convolucionales.

    El tamaño de los filtros, expresado en píxeles, sigue un formato cuadrado en los cuales las dimensiones de alto y ancho tiene el mismo valor. De esta forma el tamaño empleado por los filtros en cada una de las capas de manera progresiva se corresponde con la siguiente serie: 7x7, 5x5, 3x3, 3x3 y 3x3. Por otro lado la profundidad en los datos generada por cada uno de estos filtros, entendiendo

profundidad como el número de matrices bidimensionales o imágenes generadas, sigue la siguiente distribución: 16, 32, 64, 128 y 256.

- **Capas MaxPooling2D**: Estas capas sirven para reducir el tamaño de las imágenes que van atravesando la red. La reducción de tamaño sirve para potenciar las características más importantes dentro de las imágenes a la vez que se disminuye el uso de memoria en capas posteriores. Estas capas se encuentran intercaladas con las capas convolucionales, y cada una de ellas se encarga de reducir en un 75 % el tamaño espacial de las imágenes provenientes de la capa anterior sin modificar su profundidad. En esta reducción, la imagen se divide en grupos de 4 píxeles, teniendo altura y anchura 2, de los cuales solo permanece el que muestre el valor más alto.

- **Decoder**: Esta parte de la arquitectura empieza en la zona de menor tamaño espacial de las imágenes para ir realizando las operaciones inversas a las realizadas en el Encoder. Las imágenes van incrementando su tamaño espacial a la par que reduciendo el número de dimensiones que las componen. Esta zona es la encargada de realizar el proceso de segmentación de la imagen.

  - **Capas Conv2DTranspose**: Son un total de 4 y funcionan de forma análoga a las capas convolucionales en el Encoder, a excepción de que la aplicación de estas de forma progresiva revierte las características dimensionales y espaciales de las imágenes que fluyen por la red. Al igual que en las capas convolucionales del Encoder, se ha escogido una activación del tipo "relu" para ajustar los parámetros de los filtros.

    El tamaño empleado para los filtros en cada una de estas de manera respectiva es: 3x3, 3x3, 5x5 y 7x7. Por otro lado la profundidad generada en cada una de ellas es: 128, 64, 32 y 16. Por tanto podríamos decir, que estas capas se encargan de "revertir" las operaciones realizadas en el Encoder.

  - **Capas UpSampling2D**: Al contrario que las capas MaxPooling, en estas se incrementa el tamaño espacial de las imágenes en el mismo factor que se han ido reduciendo previamente, por tanto en cada una de estas se incrementa el tamaño espacial de los datos en un 75 %.

  - **Capas Add**: Estas capas se encuentran entre las capas de convolución transpuesta y UpScaling sirven para establecer una conexión entre las capas de igual

tamaño en el Encoder y el Decoder. Su función principal es mantener una coherencia entre la extracción de características y la generación del mapa de segmentación final entre capas con las mismas características espaciales y de profundidad.

- **Output**: Es la capa de salida de la red, esta realiza una última operación de convolución con la que se reduce la imagen a una única dimensión. Dicha dimensión tiene las mismas características espaciales que la imagen de entrada, cada uno de los píxeles que la forma expresa en forma de probabilidad en el rango 0-1, si este pertenece a una región que contiene *lobesia botrana*. Esta operación convolucional tiene el tipo de activación "sigmoid" para ajustar los valores de sus filtros. Se ha escogido este tipo de activación debido a sus características a la hora de modelar problemas de probabilidad, concretmente binarios.

En el siguiente listado de código se puede observar la implementación de la arquitectura de la red realizada en Python.

```
1  import tensorflow as tf
2  from tensorflow.keras.models import Sequential
3  from tensorflow.keras.layers import Dense, Activation, Flatten, Conv2D,
    MaxPool2D, Input, Add, Dropout, MaxPooling2D, UpSampling2D,
   Conv2DTranspose, BatchNormalization
4  from tensorflow.keras import Model, layers
5  from keras.preprocessing.image import ImageDataGenerator
6
7  #ENCODER
8  i = Input(shape=image_shape)
9  e1 = Conv2D(16, (7, 7), activation='relu', padding='same')(i)
10 mp1 = MaxPooling2D((2, 2))(e1)
11 e2 = Conv2D(32, (5, 5), activation='relu', padding='same')(mp1)
12 mp2 = MaxPooling2D((2, 2))(e2)
13 e3 = Conv2D(64, (3, 3), activation='relu', padding='same')(mp2)
14 mp3 = MaxPooling2D((2, 2))(e3)
15 e4 = Conv2D(128, (3, 3), activation='relu', padding='same')(mp3)
16 mp4 = MaxPooling2D((2, 2))(e4)
17 e5 = Conv2D(256, (3, 3), activation='relu', padding='same')(mp4)
18
19 #DECODER
20 us4 = UpSampling2D((2, 2))(e5)
```

```
21     d4 = Conv2DTranspose(128, (3, 3), activation='relu', padding='same')(
       us4)
22     f4 = Add()([e4, d4])
23     us3 = UpSampling2D((2, 2))(f4)
24     d3 = Conv2DTranspose(64, (3, 3), activation='relu', padding='same')(us3
       )
25     f3 = Add()([e3, d3])
26     us2 = UpSampling2D((2, 2))(f3)
27     d2 = Conv2DTranspose(32, (5, 5), activation='relu', padding='same')(us2
       )
28     f2 = Add()([e2, d2])
29     us1 = UpSampling2D((2, 2))(f2)
30     d1 = Conv2DTranspose(16, (7, 7), activation='relu', padding='same')(us1
       )
31     f1 = Add()([e1, d1])
32
33     #OUTPUT
34     o = Conv2DTranspose(1, activation='sigmoid', kernel_size=(7, 7),
       padding='same')(f1)
35
36     model = Model(i, o)
```

Listing 4.1: Implementación del modelo en Python

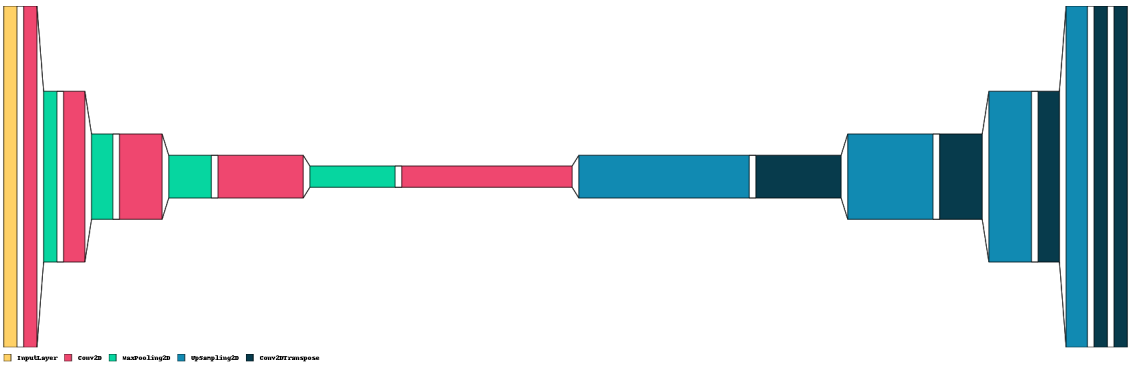


Figura 4.7: Arquitectura empleada en la red neuronal convolucional

4.3.1. Evaluación de la red

El entrenamiento de la red neuronal se ha realizado en base a un *dataset* de 1200 imágenes, que se dividen en una proporción de 80 %/20 % entre conjunto de entrenamiento y de test respectivamente. Las pruebas sobre el rendimiento de la arquitectura se estructuran

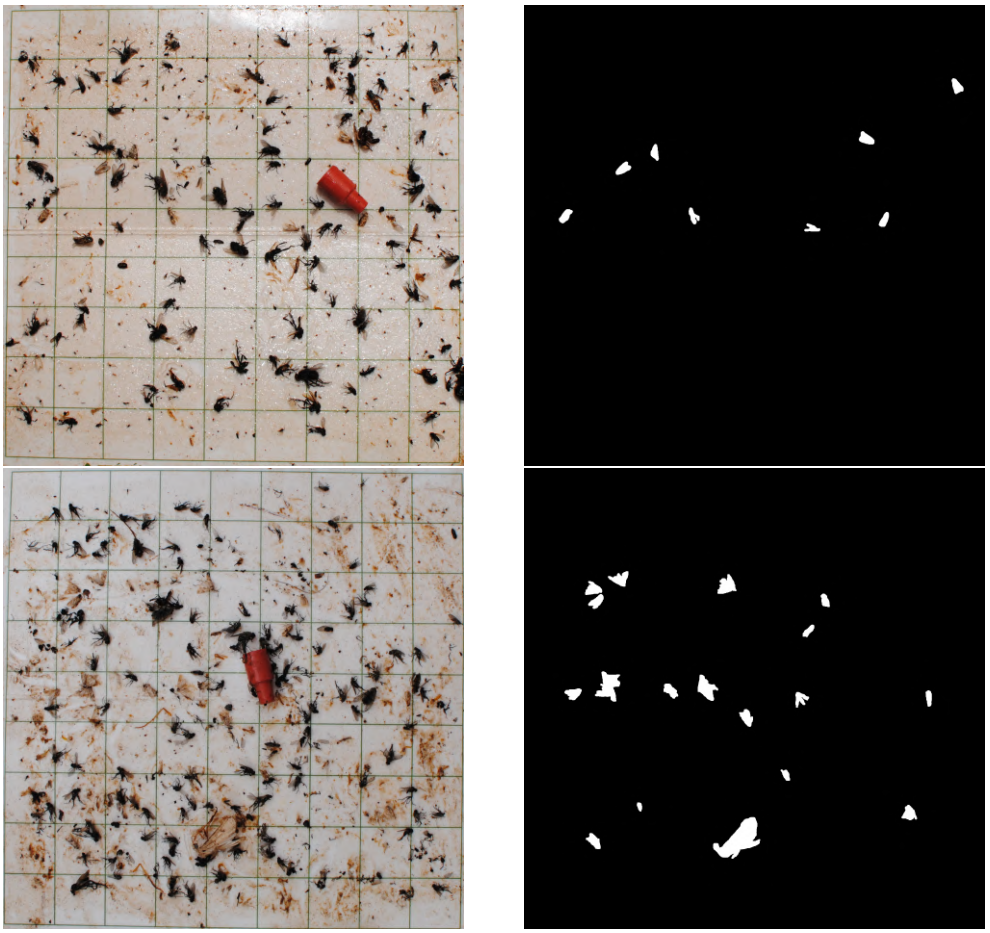
en dos grupos, con imágenes de una única dimensión en escala de grises y con imágenes de 3 dimensiones RGB. El objetivo es comprobar las diferencias entre ambas en cuanto a la calidad de la segmentación obtenida. La utilización de imágenes en escala de grises puede suponer un ahorro de memoria y de tiempo de ejecución, con respecto a imágenes RGB. Además, en determinadas tareas relacionadas con la adquisición y el preprocesamiento de dichas imágenes, antes de entrar al proceso de predicción en la red neuronal, el esfuerzo computacional al utilizar escalas de grises se reduce considerablemente.

Otro de los factores a tener en cuenta es el propio tamaño de las imágenes, este tamaño influye de manera directa en los recursos necesarios para realizar la inferencia con la red neuronal, tanto en tiempo necesario para realizar las operaciones de convolución como en la memoria de almacenamiento empleada para las capas de imágenes que se van generando durante la misma. Por ello, se han realizado pruebas para determinar la calidad de la predicción de la red utilizando imágenes de entrada de 3 tamaños diferentes: 1024x1024, 512x512 y 256x256. Estos tamaños se han escogido debido a que los diferentes tamaños producidos en base a las reducciones de tamaño internas de la red siguen conservando las características de ser potencias de 2. Los resultados han sido obtenidos con una configuración de entrenamiento de 8 *batchs* y 100 *epochs*, esta configuración ha sido establecida tratando de mantener una relación entre la calidad del entrenamiento y los recursos disponibles.

Para validar el modelo entrenado, aparte del conjunto de test, se dejaron fuera dos imágenes más Figura 4.8 que no participaron en el proceso de entrenamiento. Así, se pretende tener una idea más cercana a la realidad del sistema en producción de la calidad de la predicción.

Para evaluar la calidad de la predicción del modelo se han empleado métricas clásicas en problemas de clasificación como son : *accuracy*, *precision*, *recall* y *F1*. Como el proyecto afronta un problema de conteo de insectos se ha definido otra métrica basada en la diferencia media entre el número de insectos observados en el ground truth y en la predicción utilizando el algoritmo *conected components* para la identificación de regiones independientes en imágenes binarias. Esta métrica la podemos expresar mediante la siguiente formula:

$$Error\ medio = \frac{\sum_1^n\ |N^\circ\ insectos\ Ground\ Truth - N^\circ\ insectos\ predicción|}{n}$$



(a) Imágenes adquiridas **(b)** *Ground truth*

Figura 4.8: Imágenes empleadas en la validación de resultados

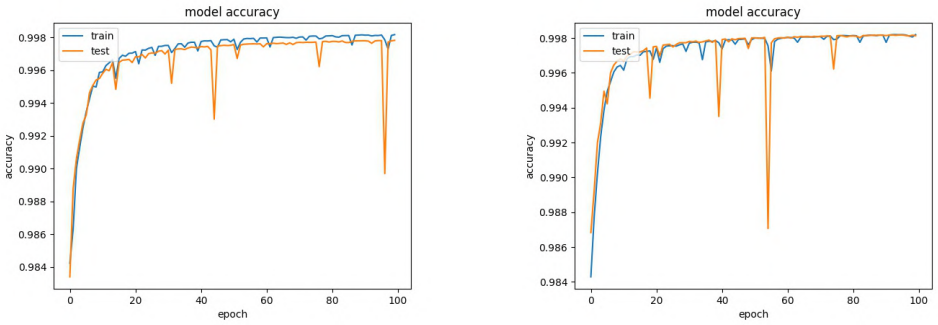
Imágenes de 1024x1024

A continuación se pueden observar las gráficas de entrenamiento obtenidas para la *accuracy* Figura 4.9 y el *loss* Figura 4.10 de imágenes de 1024x1024 píxeles en formato escala de grises y RGB.

En las siguientes tablas se muestran las métricas obtenidas para el conjunto de test Tabla 4.1 y para las dos imágenes de evaluación Tabla 4.2.

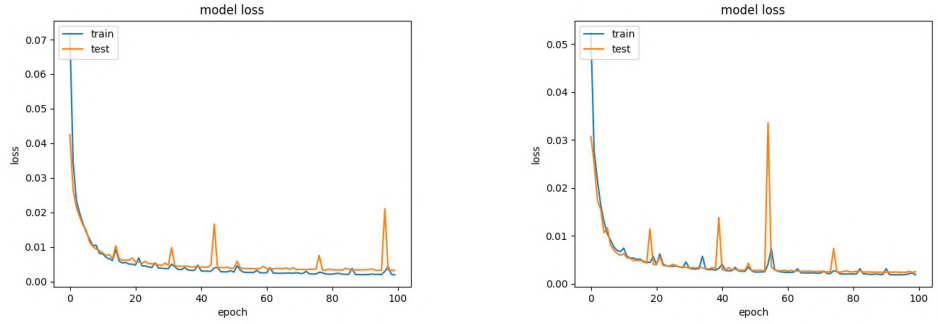
Imagen	Accuracy	Precision	Recall	F1	Error medio
Escala de grises	99,8 %	92,8 %	97,9 %	95,3 %	0,71
RGB	99,8 %	91,1 %	99,1 %	94,9 %	0,65

Tabla 4.1: Métricas obtenidas en el conjunto de test con imágenes de 1024x1024



(a) Escala de grises **(b)** RGB

Figura 4.9: Evolución del parámetro de *accuracy* durante el proceso de entrenamiento en imágenes de 1024x1024



(a) Escala de grises **(b)** RGB

Figura 4.10: Evolución del parámetro de *loss* durante el proceso de entrenamiento en imágenes de 1024x1024

Imágenes de 512x512

A continuación se pueden observar las gráficas de entrenamiento obtenidas para la *accuracy* Figura 4.11 y el *loss* Figura 4.12 de imágenes de 512x512 píxeles en formato escala de grises y RGB.

En las siguientes tablas se muestran las métricas obtenidas para el conjunto de test Tabla 4.3 y para las dos imágenes de evaluación Tabla 4.4.

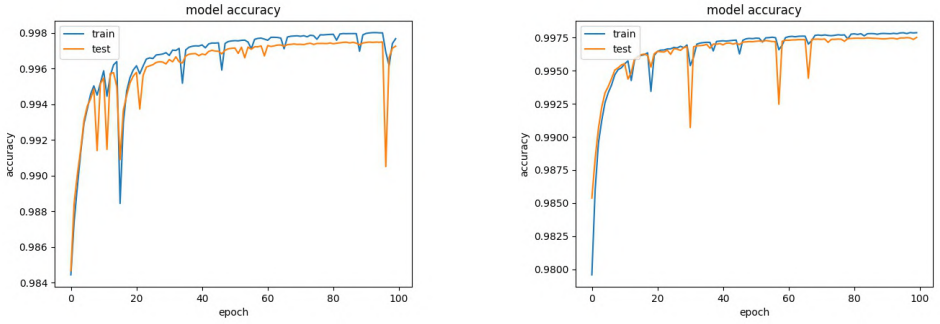
Imágenes de 256x256

A continuación se pueden observar las gráficas de entrenamiento obtenidas para la *accuracy* Figura 4.13 y el *loss* Figura 4.14 de imágenes de 256x256 píxeles en formato escala de grises y RGB.

En las siguientes tablas se muestran las métricas obtenidas para el conjunto de test

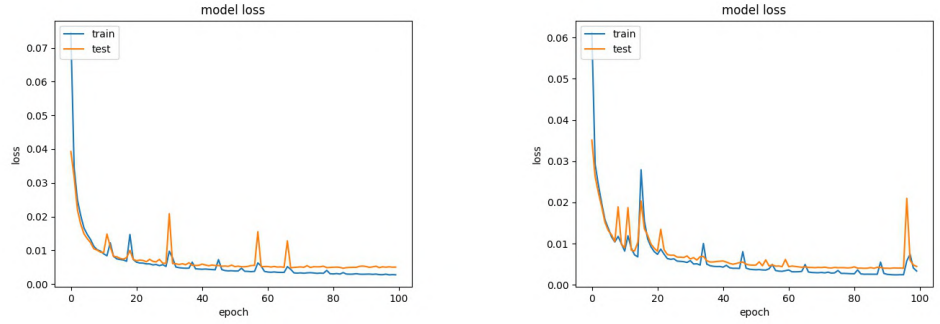
Imagen	Accuracy	Precision	Recall	F1	Error medio
Escala de grises	99 %	47,6 %	49,7 %	48,6	11,5
RGB	99,3 %	63,7 %	59,9 %	61,7 %	4

Tabla 4.2: Métricas obtenidas en el conjunto de evaluación con imágenes de 1024x1024



(a) Escala de grises **(b)** RGB

Figura 4.11: Evolución del parámetro de *accuracy* durante el proceso de entrenamiento en imágenes de 512x512



(a) Escala de grises **(b)** RGB

Figura 4.12: Evolución del parámetro de *loss* durante el proceso de entrenamiento en imágenes de 512x512

Tabla 4.5 y para las dos imágenes de evaluación Tabla 4.6.

Evaluación de resultados

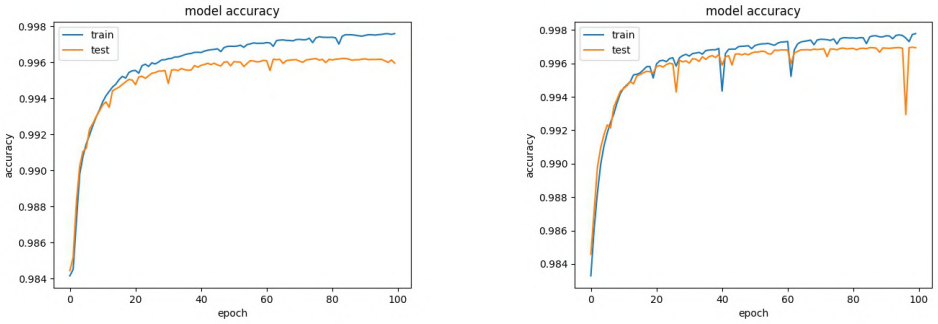
Los resultados obtenidos muestran que existe una ligera mejora si se utilizan imágenes en formato RGB con respecto a las imágenes en escala de grises, sobre todo con las dos imágenes del conjunto de evaluación. Esto nos puede indicar que utilizar imágenes en formato escala de grises hace que la red sea más propensa a caer en el *overfitting*. Otra

Imagen	Accuracy	Precision	Recall	F1	Error medio
Escala de grises	99,8 %	91,9 %	96,2 %	94 %	0,95
RGB	99,8 %	90,4 %	97,1 %	93,6 %	1,18

Tabla 4.3: Métricas obtenidas en el conjunto de test con imágenes de 512x512

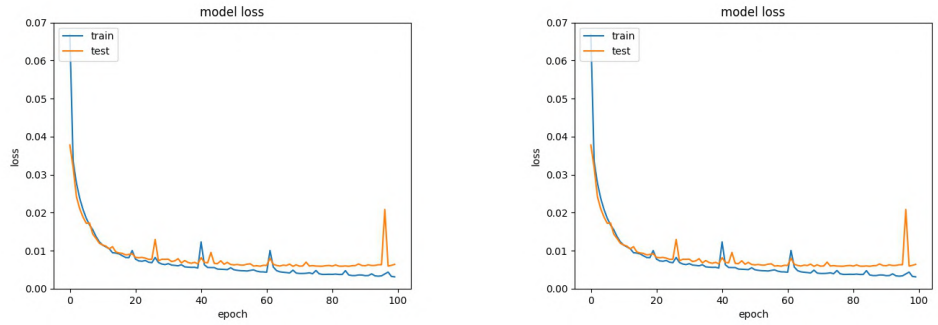
Imagen	Accuracy	Precision	Recall	F1	Error medio
Escala de grises	99 %	50 %	44,5 %	47,2 %	6,5
RGB	99,2 %	60,4 %	47,7 %	53,3 %	7

Tabla 4.4: Métricas obtenidas en el conjunto de evaluación con imágenes de 512x512



(a) Escala de grises **(b)** RGB

Figura 4.13: Evolución del parámetro de *accuracy* durante el proceso de entrenamiento en imágenes de 256x256



(a) Escala de grises **(b)** RGB

Figura 4.14: Evolución del parámetro de *loss* durante el proceso de entrenamiento en imágenes de 256x256

de las conclusiones que podríamos obtener es que el espacio de color juega un papel clave en la diferenciación de las regiones que se pretenden segmentar respecto a otros elementos

Imagen	Accuracy	Precision	Recall	F1	Error medio
Escala de grises	99,6 %	86,6 %	91 %	88,8 %	3,34
RGB	99,7 %	90,6 %	94 %	92,3 %	0,96

Tabla 4.5: Métricas obtenidas en el conjunto de test con imágenes de 256x256

Imagen	Accuracy	Precision	Recall	F1	Error medio
Escala de grises	99 %	50,2 %	22,5 %	31,1 %	3,5
RGB	99,1 %	62 %	21,1 %	31,5 %	4

Tabla 4.6: Métricas obtenidas en el conjunto de evaluación con imágenes de 256x256

como polvo u otro tipo de insectos.

Por otra parte también se puede observar una diferencia significativa entre los resultados obtenidos con el conjunto de test y el conjunto de evaluación. Esto se puede entender en base a las diferencias existentes entre las imágenes del conjunto de test y las imágenes del conjunto de evaluación, para las cuales la red no es capaz de ceñirse con la misma precisión.

Si tenemos en cuenta los resultados en función del tamaño de las imágenes, existe una tendencia directamente relacionada entre la resolución y la predicción, donde a mayor resolución se obtienen mejores resultados en la predicción.

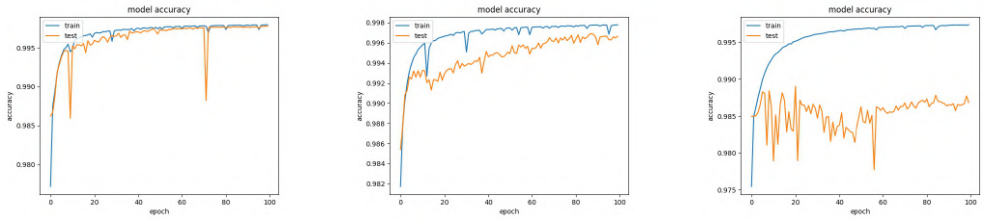
4.3.2. Regularización de la red neuronal aplicando dropout

Con el objetivo de mejorar la capacidad de generalización de la red en las predicciones, se ha optado por aplicar técnicas de regularización. La técnica escogida ha sido la de dropout, en la cual se desactiva un número de capas aleatorias para forzar que la red no sea capaz de ajustarse a los datos de entrenamiento y por tanto reducir el *overfitting*.

Dado que las imágenes en escala de grises han arrojado peores resultados en la etapa anterior, llegados a este punto se descarta su uso. Esta técnica de regularización se ha aplicado a todas las capas convolucionales de la red. Generalmente se utilizan valores de *dropout* dentro del rango 0,2-0,5 por ello se ha decidido escoger los valores límite (0,2 y 0,5) además de un valor intermedio (0,3). A continuación se van a evaluar estos 3 ratios aplicados en modelos con imágenes de entrada de 1024x1024, 512x512 y 256x256 píxeles.

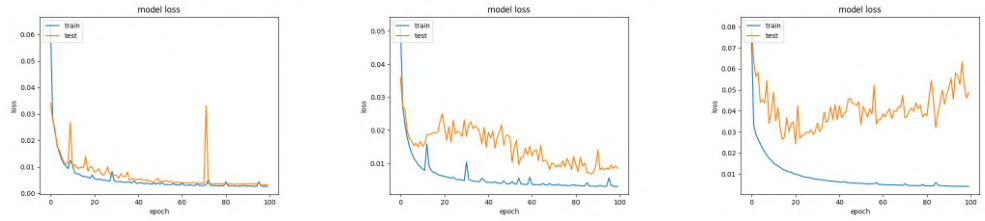
Imágenes de 1024x1024

A continuación se pueden observar las gráficas de entrenamiento obtenidas para la *accuracy* Figura 4.15 y el *loss* Figura 4.16 de imágenes de 1024x1024 píxeles en formato RGB.



(a) dropout = 0,2 **(b)** dropout = 0,3 **(c)** dropout = 0,5

Figura 4.15: Evolución del parámetro de *accuracy* durante el proceso de entrenamiento con diferentes valores de dropout en imágenes de 1024x1024



(a) dropout = 0,2 **(b)** dropout = 0,3 **(c)** dropout = 0,5

Figura 4.16: Evolución del parámetro de *loss* durante el proceso de entrenamiento con diferentes valores de dropout en imágenes de 1024x1024

En las siguientes tablas se muestran las métricas obtenidas para el conjunto de test Tabla 4.7 y para las dos imágenes de evaluación Tabla 4.8.

Dropout	Accuracy	Precision	Recall	F1	Error medio
0,2	99,8 %	93,6 %	97,2 %	95,4 %	0,5
0,3	99,7 %	93,7 %	89,8 %	91,7 %	2
0,5	98,7 %	62,4 %	46,1 %	53 %	13,3

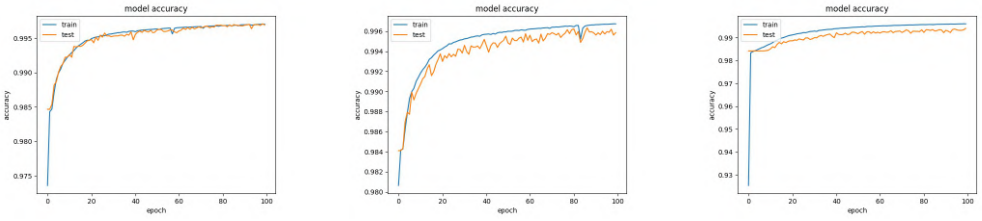
Tabla 4.7: Métricas obtenidas en el conjunto de test aplicando *dropout* con imágenes de 1024x1024

Dropout	Accuracy	Precision	Recall	F1	Error medio
0,2	99,3 %	68,3 %	53,9 %	60,2 %	4
0,3	99 %	52,7 %	36,7 %	43,2 %	4
0,5	98,1 %	20,3 %	32,2 %	24,9 %	10,5

Tabla 4.8: Métricas obtenidas en el conjunto de evaluación aplicando *dropout* con imágenes de 1024x1024

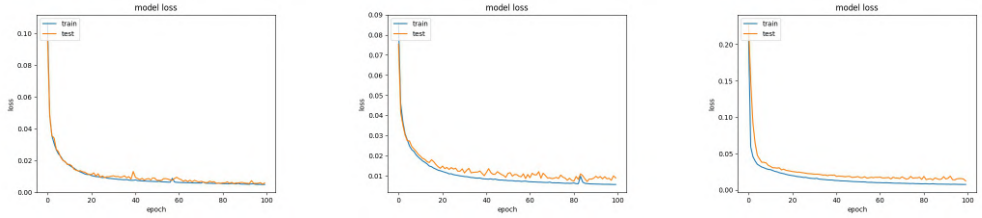
Imágenes de 512x512

A continuación se pueden observar las gráficas de entrenamiento obtenidas para la *accuracy* Figura 4.17 y el *loss* Figura 4.18 de imágenes de 512x512 píxeles en formato RGB.



(a) dropout = 0,2 **(b)** dropout = 0,3 **(c)** dropout = 0,5

Figura 4.17: Evolución del parámetro de *accuracy* durante el proceso de entrenamiento con diferentes valores de dropout en imágenes de 512x512



(a) dropout = 0,2 **(b)** dropout = 0,3 **(c)** dropout = 0,5

Figura 4.18: Evolución del parámetro de *loss* durante el proceso de entrenamiento con diferentes valores de dropout en imágenes de 512x512

En las siguientes tablas se muestran las métricas obtenidas para el conjunto de test Tabla 4.9 y para las dos imágenes de evaluación Tabla 4.10.

Dropout	Accuracy	Precision	Recall	F1	Error medio
0,2	99,7 %	90,7 %	92,8 %	91,7 %	1,1
0,3	99,7 %	89,8 %	88,3 %	89 %	1,59
0,5	99,5 %	87 %	78 %	82 %	2,5

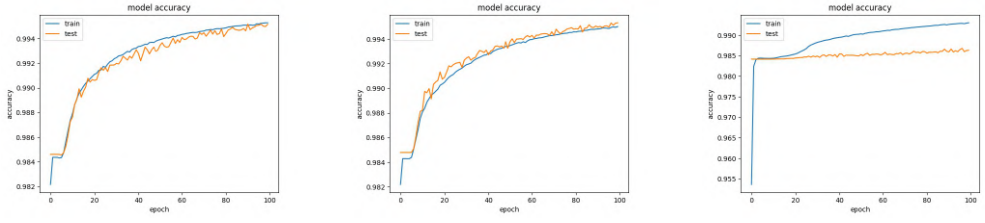
Tabla 4.9: Métricas obtenidas en el conjunto de test aplicando *dropout* con imágenes de 512x512

Dropout	Accuracy	Precision	Recall	F1	Error medio
0,2	99,3 %	67,3 %	56,8 %	61,6 %	0,5
0,3	99,1 %	62,3 %	23,7 %	34,5 %	10
0,5	99,1 %	62,2 %	30,3 %	40,7 %	8

Tabla 4.10: Métricas obtenidas en el conjunto de evaluación aplicando *dropout* con imágenes de 512x512

Imágenes de 256

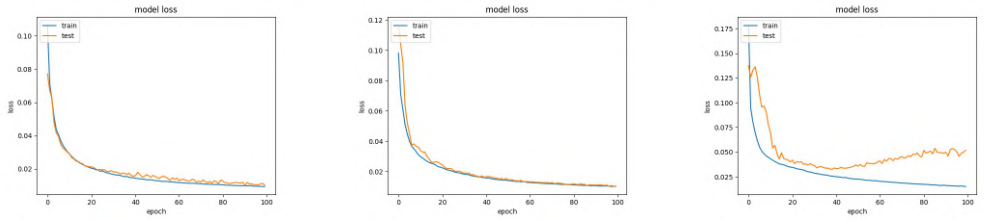
A continuación se pueden observar las gráficas de entrenamiento obtenidas para la *accuracy* Figura 4.19 y el *loss* Figura 4.20 de imágenes de 256x256 píxeles en formato RGB.



(a) dropout = 0,2 **(b)** dropout = 0,3 **(c)** dropout = 0,5

Figura 4.19: Evolución del parámetro de *accuracy* durante el proceso de entrenamiento con diferentes valores de dropout en imágenes de 256x256

En las siguientes tablas se muestran las métricas obtenidas para el conjunto de test Tabla 4.11 y para las dos imágenes de evaluación Tabla 4.12.



(a) dropout = 0,2 **(b)** dropout = 0,3 **(c)** dropout = 0,5

Figura 4.20: Evolución del parámetro de *loss* durante el proceso de entrenamiento con diferentes valores de dropout en imágenes de 256x256

Dropout	Accuracy	Precision	Recall	F1	Error medio
0,2	99,6 %	88,3 %	84,3 %	86,2 %	1,4
0,3	99,5 %	86,1 %	84,6 %	85,3 %	1,5
0,5	98,7 %	94,1 %	14 %	24,3 %	12,5

Tabla 4.11: Métricas obtenidas en el conjunto de test aplicando *dropout* con imágenes de 256x256

Evaluación de resultados

Los resultados obtenidos en las pruebas muestran una diferencia muy significativa entre los resultados obtenidos con cada uno de estos ratios diferentes, observamos una relación directa en la que a mayor valor de *dropout*, peor es el resultado obtenido. Con respecto a los resultados originales, estos solo son mejorados con el ratio de 0,2. Como se ha podido observar en las gráficas anteriores, el entrenamiento con todos los tamaños de imagen presenta un comportamiento más errático a mayor valor de ratio de *dropout*.

Tras esta evaluación de resultados nos encontramos que la mejor predicción se ha dado con imágenes de tamaño 512x512, formato de color RGB y *dropout*=0,2. En la Figura 4.21 podemos observar la predicción obtenida para una de las imágenes del conjunto de evaluación.

4.4. Conversión a Tensorflow Lite y optimizaciones

Dado que el objetivo de este proyecto es la creación de un sistema empotrado el cual sea capaz de realizar este procesamiento, uno de los pasos necesarios es la adaptación del modelo entrenado a este tipo de dispositivos. Es por ello que existe una versión optimi-

Dropout	Accuracy	Precision	Recall	F1	Error medio
0,2	99,1 %	59,4 %	24,4 %	34,6 %	2,5
0,3	99,1 %	58,5 %	13,4 %	21,8 %	7,5
0,5	99 %	0 %	0 %	0 %	12,5

Tabla 4.12: Métricas obtenidas en el conjunto de evaluación aplicando *dropout* con imágenes de 256x256

(a) Imagen original **(b)** Ground thruth **(c)** Predicción

Figura 4.21: Resultado predicción

zada para su uso en dispositivos móviles, TensorFlow Lite. Algunas de sus características principales son las siguientes:

1. No hay transferencia de información de ida y vuelta con un servidor y no es necesaria la conexión a Internet.
2. Ningún dato sale del dispositivo.
3. Tamaño reducido del modelo y de los objetos binarios.
4. Inferencia de alto rendimiento con aceleradores hardware y opciones de optimización de modelos.
5. Compatibilidad con diversas plataformas y lenguajes.

Habitualmente, los modelos de *deep learning* utilizan tipos de datos numéricos en punto flotante de 16 o 32 bits con los que realizar las operaciones internas de inferencia. Una de las posibilidades que nos ofrece TensorFlow Lite es la de producir una cuantización de estos tipos de datos a enteros con 8 bits de precisión. Esta cuantización supone un proceso de discretización de las variables involucradas en las operaciones internas del modelo. Con

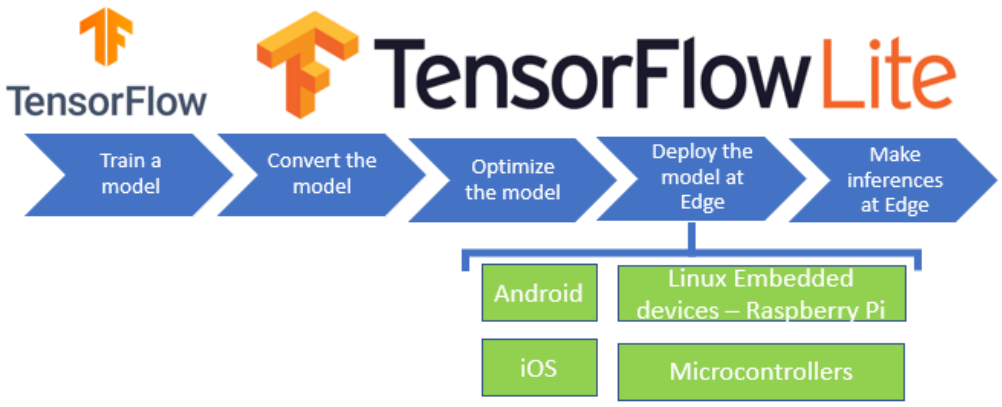


Figura 4.22: Diagrama de flujo TensorFlow Lite [14]

esto, se definen rangos en los que las variables contenidas dentro de los mismos pasarán a tener un mismo valor identificando el rango al que pertenecen. Esta operación se realiza debido a que las CPU de sistemas empotrados son mucho más eficientes trabajando con números enteros, incluso algunos microcontroladores sólo tienen la capacidad de operar con este tipo de datos.

A continuación se van a evaluar las diferencias ente el modelo convertido a TensorFlow Lite sin ninguna mejora y el modelo convertido aplicándole la cuantización. Para esto se van a medir las calidades de predicciones y la memoria utilizada con cada uno.

En la Tabla 4.13 se muestra la diferencia en uso de memoria para cada uno de los dos modelos en función del tamaño de la imagen. Esta estimación de la utilización de memoria del modelo se ha realizado a través de una de las funcionalidades proporcionadas por la propia librería TensorFlow. Como podemos observar apenas existe una reducción de entre el 1 % y el 6 % en la memoria utilizada durante la inferencia.

Modelo	**256**	**512**	**1024**
Modelo sin cuantización	380 MB	1305 MB	4992 MB
Modelo con cuantización a enteros de 8 bits	358 MB	1279 MB	4964 MB

Tabla 4.13: Memoria consumida por los modelos antes y después de aplicar cuantización.

En las siguiente tablas Tabla 4.14, Tabla 4.15 y Tabla 4.16 se pueden observar de manera respectiva la diferencia en los resultados obtenidos para el modelo con y sin cuan-

tización en función de diferentes tamaños de imágenes de entrada. En estas métricas se aprecia una ligera perdida de calidad en las predicciones en los modelos a los que se les ha aplicado la cuantización. También se puede apreciar que cuanto menor es la resolución de las imágenes de los modelos, menor es la diferencia existente entre la aplicación o no de este proceso. En la Figura 4.23 se puede observar la diferencia en la predicción entre estos dos modelos.

Modelo	**Accuracy**	**Precision**	**Recall**	**F1**	**Error medio**
Modelo sin cuantización	99,3 %	68,3 %	53,9 %	60,2 %	4
Modelo con cuantización a enteros de 8 bits	99 %	60,5 %	11 %	18,6 %	19

Tabla 4.14: Diferencia entre las métricas de los modelos en TensorFlow Lite con imágenes de 1024x1024

Modelo	**Accuracy**	**Precision**	**Recall**	**F1**	**Error medio**
Modelo sin cuantización	99,3 %	67,3 %	56,8 %	61,6 %	0,5
Modelo con cuantización a enteros de 8 bits	99,3 %	69 %	51 %	58,7 %	1

Tabla 4.15: Diferencia entre las métricas de los modelos en TensorFlow Lite con imágenes de 512x512

Modelo	**Accuracy**	**Precision**	**Recall**	**F1**	**Error medio**
Modelo sin cuantización	99,1 %	59,4 %	24,4 %	34,6 %	2,5
Modelo con cuantización a enteros de 8 bits	99,1 %	62,9 %	23,8 %	34,5 %	1,5

Tabla 4.16: Diferencia entre las métricas de los modelos en TensorFlow Lite con imágenes de 256x256

(a) Ground truth **(b)** Predicción sin cuantización **(c)** Predicción con cuantización

Figura 4.23: Diferencias predicciones TensorFlow Lite

4.5. Implementación sistema *on the edge*

Se ha optado por escoger el lenguaje de programación C++ para realizar la implementación del sistema de inferencia empotrado, debido al mayor rendimiento y eficiencia en el uso de recursos que hace este con respecto a otros lenguajes. Sumado a esto, también se ha hecho uso de las librerías OpenCV y TensorFlow Lite.

En el Apéndice A se puede observar la implementación del sistema de inferencia final que va a ser desplegado en los dispositivos, este sera el encargado de la adquisición de las imágenes, así como de adaptarlas a las características espaciales del modelo que se está haciendo uso y de obtener la predicción desde este.

Esta implementación va a ser desplegada en 3 modelos de dispositivos Raspberry Pi con diferentes características hardware con tal de evaluar la viabilidad de implementación del sistema final en cada uno de estos dispositivos. Para ello se van a medir las diferencias en tiempo de ejecución y consumo energético obtenidas en la ejecución de los modelos.

Modelo	CPU	RAM
Raspberry Pi 4	Cortex-A72 64 bit @ 1,5 GHz	4 GB LPDDR4
Raspberry Pi 3 B	Cortex-A53 32 bit @ 1,2 GHz	1 GB LPDDR2
Raspberry Pi Zero 2 W	Cortex-A53 32 bit @ 1 GHz	512 MB LPDDR2

Tabla 4.17: Características técnicas sistemas Raspberry Pi

Tal y como se observaba en la Tabla 4.13 los tamaños de imagen con los que se han entrenado los modelos tienen un impacto directo en la memoria que necesita cada uno de estos para ejecutarse, lo que hace que existan restricciones en el número de dispositivos que se puede desplegar en cada uno.

4.5.1. Comparación tiempos de ejecución

Los tiempos de ejecución que se van a mostrar a continuación corresponden a una media del tiempo necesario para procesar una imagen de un determinado tamaño en cada uno de estos dispositivos.

Modelo	**256**	**512**	**1024**
Modelo sin cuantización	1,85 s	5,89 s	108 s
Modelo con cuantización a enteros de 8 bits	1,35 s	3,82 s	90,7 s

Tabla 4.18: Tiempo de ejecución en Raspberry Pi 4

Modelo	**256**	**512**	**1024**
Modelo sin cuantización	5,62 s	757 s	NA
Modelo con cuantización a enteros de 8 bits	4,04 s	457 s	NA

Tabla 4.19: Tiempo de ejecución en Raspberry Pi 3 B

Modelo	**256**	**512**	**1024**
Modelo sin cuantización	62,8 s	NA	NA
Modelo con cuantización a enteros de 8 bits	30,2 s	NA	NA

Tabla 4.20: Tiempo de ejecución en Raspberry Pi Zero 2 W

Tal y como se puede observar en las tablas anteriores, la memoria RAM disponible en cada uno de estos dispositivos es un factor limitante a la hora de desplegar estos modelos. En los casos en los que la memoria necesaria sobrepasa o se encuentra cerca del límite de la memoria disponible el dispositivo se ve obligado a hacer uso de la memoria virtual. Esto supone que el sistema operativo se ve obligado a almacenar parte de su memoria en el almacenamiento primario, lo que hace que el acceso a los datos sea mucho más lento. En algunos casos el despliegue de un determinado tamaño de imagen se hace completamente inviable.

4.5.2. Comparación consumo energético

El consumo energético es uno de los factores clave en el despliegue de este tipo de sistemas debido a las características de los mismos que muchas veces se van a encontrar desconectados de un suministro eléctrico estable. Por esta razón es necesario buscar el mayor ahorro energético posible. Los consumos que se van a mostrar a continuación se corresponden con una media del consumo necesario para procesar una imagen de un determinado tamaño en cada uno de estos dispositivos medido en miliamperios hora.

Modelo	**256**	**512**	**1024**
Modelo sin cuantización	0,32 MAH	1,14 MAH	18 MAH
Modelo con cuantización a enteros de 8 bits	0,28 MAH	0,75 MAH	15 MAH

Tabla 4.21: Consumo energético en Raspberry Pi 4

Modelo	**256**	**512**	**1024**
Modelo sin cuantización	0,75 MAH	74 MAH	NA
Modelo con cuantización a enteros de 8 bits	0,45 MAH	44 MAH	NA

Tabla 4.22: Consumo energético en Raspberry Pi 3 B

Modelo	**256**	**512**	**1024**
Modelo sin cuantización	2,57 MAH	NA	NA
Modelo con cuantización a enteros de 8 bits	1,28 MAH	NA	NA

Tabla 4.23: Consumo energético en Raspberry Pi Zero 2 W

Los resultados obtenidos nos indican que existe una relación directa entre el tiempo de ejecución necesario para procesar las imágenes con la cantidad de energía empleada independientemente del dispositivo empleado.

Capítulo 5

Conclusiones y Trabajo Futuro

En este capítulo se van a exponer las principales conclusiones derivadas de la realización de este TFM así como una revisión de los objetivos cumplidos y el posible trabajo futuro.

5.1. Objetivos cumplidos

El objetivo de este proyecto trataba sobre el desarrollo de un sistema *edge computing* para la detección de *Lobesia Botrana*, en el cual se implementará un sistema que tuviera la capacidad de estimar el número de insectos utilizando técnicas de visión por computador e inteligencia artificial. Para evaluar la consecución global del proyecto vamos a analizar los objetivos parciales propuestos.

Objetivos específicos cumplidos:

- **Análisis de las tecnologías potenciales a utilizar**: Se ha realizado un análisis de las tecnologías necesarias que han sido empleadas para la consecución del proyecto. En cuanto al hardware se ha determinado la necesidad de tener un dispositivo el cual fuese capaz de realizar la adquisición y procesamiento de imágenes con un reducido consumo energético, del que se ha determinado que los sistemas Raspberry Pi son candidatos ideales para realizar estas funciones. En la parte de diseño software del sistema se optó por escoger OpenCV para el procesamiento de imagen y TensorFlow para la inteligencia del sistema debido a que ambos cumplían con las capacidades necesarias para solventar los requisitos asociados al proyecto.

- **Estudio de las técnicas de análisis de imagen mediante IA**: Se han estudiado diferentes alternativas que podrían servir para afrontar la solución del problema.

De las cuales se ha optado por el empleo de una arquitectura de redes neuronales convoluciones de tipo Encoder-Decoder. Se ha elegido esta arquitectura por ser usada en otros problemas de segmentación semántica con buenos resultados. Además se han hecho varias pruebas con los parámetros de la arquitectura para determinar que combinaciones de estos aportan mejores resultados. En el contexto del proyecto, la utilización de imágenes en formato RGB sumadas a una regularización de los parámetros mediante *dropout* ha demostrado obtener los mejores resultados.

- **Adquisición de imágenes y creación del conjunto de datos de entrenamiento**: Las imágenes con las que se ha trabajado en el problema se han obtenido a partir de trampas cromáticas en una explotación agrícola de vid en la provincia de Ciudad Real. El hecho de tener un conjunto reducido de imágenes ha hecho necesaria la utilización de técnicas de *data augmentation* con las que incrementar este.
- **Adaptación del algoritmo a un sistema empotrado**: Se ha realizado una implementación en C++ del sistema para realizar el análisis de nuevas imágenes adquiridas en un sistema Raspberry Pi.
- **Optimización del modelo de IA para su utilización en un sistema empotrado**: Se ha realizado la conversión de los modelos entrenados en TensorFlow a TensorFlow Lite para su utilización en sistemas móviles con una menor cantidad de recursos. Además también se han aplicado técnicas de cuantización para modificar los datos internos del modelo de punto flotante a entero y así ejecutarse de una forma más eficiente en microcontroladores. Tras las pruebas realizadas se ha determinado que las CPU de los dispositivos utilizados son lo suficientemente avanzadas como para realizar operaciones con datos vectoriales en punto flotante y por tanto no existe una diferencia significativa en la cuantización o no de los parámetros.
- **Analizar desempeño y valoración de requisitos no funcionales como consumo o recursos utilizados**: Se ha realizado un análisis exhaustivo de las diferentes configuraciones de los modelos para realizar la inferencia de imágenes. Se ha analizado la memoria utilizada por cada una de las configuraciones de los modelos para realizar la inferencia así como el consumo de los mismos en tiempo y energía eléctrica en 3 dispositivos Raspberry Pi diferentes. Las pruebas demostraron que el modelo Raspberry Pi 4 fue el único con la potencia y capacidad de memoria necesarias para poder afrontar la ejecución de todas las configuraciones posibles de los

modelos. Además este dispositivo también ha sido el mejor en términos de eficiencia energética, descartando para afrontar el problema a los modelos Raspberry Pi 3 B y Raspberry Pi Zero 2 W.

Por todas estas razones se considera que se han cumplido los requisitos establecidos para la consecución del proyecto.

5.2. Trabajo futuro

- **Aumento del conjunto de datos**: Incrementar el número de imágenes utilizadas en el entrenamiento de los modelos es clave para poder tener una mejor robustez y precisión en estos.
- **Módulo de comunicaciones**: La implementación de un modulo de comunicaciones proporcionaría la capacidad de poder obtener resultados del procesamiento realizado de manera remota, adquirir nuevas imágenes para incrementar el conjunto de datos o poder realizar mejoras de manera remota en el sistema. Estas actualizaciones principalmente podrían tener el cometido de actualizar el modelo utilizado para la inferencia con nuevas versiones mejoradas.
- **Exploración con nuevos parámetros y arquitecturas**: Explorar la utilización de parámetros diferentes y nuevas arquitecturas y su impacto en los resultados del problema.
- **Construcción del prototipo**: Construir el prototipo con el sistema funcional y evaluar el rendimiento del sistema en un entorno de producción.

Bibliografía

[1] (2022). "¿Qué es Python?", dirección: https://web.archive.org/web/20200224120525/https://luca-d3.com/es/data-speaks/diccionario-tecnologico/python-lenguaje.

[2] Y. Ai y col., "Research on Recognition Model of Crop Diseases and Insect Pests Based on Deep Learning in Harsh Environments," *IEEE Access*, vol. 8, págs. 171 686-171 693, ene. de 2020. DOI: 10.1109/ACCESS.2020.3025325.

[3] V. Badrinarayanan, A. Kendall y R. Cipolla, "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, n.º 12, págs. 2481-2495, 2017. DOI: 10.1109/TPAMI.2016.2644615.

[4] O. Comunity. (2020). "Introduction", [Online]. Available: https://docs.opencv.org/4.3.0/d1/dfb/intro.html.

[5] D. E. (2019). "Understand tensorflow by mimicking it api from scratch", [Online]. Available: https://d3lm.medium.com/understand-tensorflow-by-mimicking-its-api-from-scratch-faa55787170d.

[6] C. Europea. (2020). "Apoyo y protección de los viticultores, productores de vino, vendedores y consumidores de la UE a través de políticas, legislación, etiquetado, medidas comerciales y seguimiento de los mercados.", dirección: https://ec.europa.eu/info/food-farming-fisheries/plants-and-plant-products/plant-products/wine_es.

[7] H. Fakhruddin. (2017). "Smart agriculture: 13 trends to watch out for", [Online]. Available: https://teks.co.in/site/blog/smart-agriculture-13-trends-to-watch-out-for/.

[8] P. S. Foundation. (2022). "Python documentation", [Online]. Available: https://docs.python.org/3/.

[9] R. P. Foundation. (2022). "Raspberry pi", [Online]. Available: https://www.raspberrypi.com/.

[10] A. Garcia-Garcia y col., "A review on deep learning techniques applied to semantic segmentation," *arXiv preprint arXiv:1704.06857*, 2017.

[11] G. B. Garcıa y col., *Learning image processing with OpenCV*. Packt Publishing Birmingham, 2015.

[12] R. Grosse-Kunstleve y col., "Automatic Fortran to C++ conversion with FABLE," *Source code for biology and medicine*, vol. 7, pág. 5, mayo de 2012. DOI: 10.1186/1751-0473-7-5.

[13] H. He y col., "A Comparative Deep Learning Algorithms for Agricultural Insect Recognition," en *2021 International Conference on Machine Learning and Intelligent Systems Engineering (MLISE)*, IEEE, 2021, págs. 321-325.

[14] R. Khandelwal. (2020). "A basic introduction to tensorflow lite", [Online]. Available: https://towardsdatascience.com/a-basic-introduction-to-tensorflow-lite-59e480c57292.

[15] C. Nwankpa y col., "Activation functions: Comparison of trends in practice and research for deep learning," *arXiv preprint arXiv:1811.03378*, 2018.

[16] K. O'Shea y R. Nash, "An introduction to convolutional neural networks," *arXiv preprint arXiv:1511.08458*, 2015.

[17] J. Pastor. (2018). "Edge Computing: que es y por que hay gente que piensa que es el futuro", dirección: https://www.xataka.com/internet-of-things/edge-computing-que-es-y-por-que-hay-gente-que-piensa-que-es-el-futuro (visitado 30-06-2020).

[18] B. Ramalingam y col., "Remote insects trap monitoring system using deep learning framework and IoT," *Sensors*, vol. 20, n.º 18, pág. 5280, 2020.

[19] F. Rosenblatt, "Perceptron Simulation Experiments," *Proceedings of the IRE*, vol. 48, n.º 3, págs. 301-309, 1960. DOI: 10.1109/JRPROC.1960.287598.

[20] J. L. M. Serrano. (2021). "Algoritmo de segmentación eficiente para la detección de insectos en trampas cromáticas", dirección: https://arcoresearch.com/2021/09/14/algoritmo-de-segmentacion-eficiente-para-la-deteccion-de-insectos-en-trampas-cromaticas/.

[21] Y. Shen y col., "Detection of stored-grain insects using deep learning," *Computers and Electronics in Agriculture*, vol. 145, págs. 319-325, 2018.

[22] I. Sommerville, *Ingenieria del Software*, 7.ª ed. Addison-Wesley, 2007.

[23] N. Srivastava y col., "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, n.º 1, págs. 1929-1958, 2014.

[24] B. Stroustrup, "A History of C++: 1979–1991," en *History of Programming Languages—II*. New York, NY, USA: Association for Computing Machinery, 1996, págs. 699-769, ISBN: 0201895021. dirección: https://doi.org/10.1145/234286.1057836.

[25] G. B. Team. (2020). "Introduction", [Online]. Available: https://www.tensorflow.org/.

[26] H. Vivas, H. J. Martınez y R. Pérez, "Structured secant method for the multilayer perceptron training," *Revista de Ciencias*, vol. 18, n.º 2, págs. 131-150, 2014.

[27] B. Wicht, "Deep Learning feature Extraction for Image Processing," Tesis doct., ene. de 2018. DOI: 10.13140/RG.2.2.11700.35207.

Apéndice A

Código fuente sistema de inferencia *on the edge*

```
int main(int argc, char *argv[])
{
    auto start = std::chrono::steady_clock::now();

    std::string SourceImgPathString, ConfigFilePathString,
    DestinationImgPathString;

    parseArguments(argc, argv, SourceImgPathString, ConfigFilePathString,
    DestinationImgPathString);

    fs::path SourceImgPath = fs::path(SourceImgPathString);
    fs::path ConfigFilePath = fs::path(ConfigFilePathString);
    fs::path DestinationImgPath = fs::path(DestinationImgPathString);

    auto now = std::chrono::system_clock::now();
    auto in_time_t = std::chrono::system_clock::to_time_t(now);
    DestinationImgPath /= std::ctime(&in_time_t);
    DestinationImgPath += fs::path::preferred_separator;

    fs::create_directories(DestinationImgPath);

    std::ofstream OutputFile = std::ofstream(DestinationImgPath/"Output.txt
    ");

    std::ifstream ConfigFile;
    ConfigFile.open(ConfigFilePath, std::ios::out);

    if(!ConfigFile)
    {
        std::cerr << "Couldn open config file\n";
        return 1;
    }

    std::string ModeString, ImgSizeString, ModelPathString, ModelNameString
    ;

    std::getline(ConfigFile, ModeString);
    std::getline(ConfigFile, ImgSizeString);
    std::getline(ConfigFile, ModelPathString);
    std::getline(ConfigFile, ModelNameString);

```

```
38    OutputFile << ModeString << std::endl;
39    OutputFile << ImgSizeString << std::endl;
40    OutputFile << ModelNameString << std::endl << std::endl;
41
42
43    cv::ImreadModes Mode;
44    int Channels = 0;
45
46    if(ModeString == "RGB")
47    {
48        Mode = cv::IMREAD_COLOR;
49        Channels = 3;
50    }
51    else
52    {
53        Mode = cv::IMREAD_GRAYSCALE;
54        Channels = 1;
55    }
56
57    int ImgSize = stoi(ImgSizeString);
58
59    fs::path ModelPath = fs::path(ModelPathString)+=fs::path(ModeString)/=
      fs::path(ImgSizeString)/=fs::path(ModelNameString);
60
61    std::unique_ptr<tflite::FlatBufferModel> model = tflite::
      FlatBufferModel::BuildFromFile(ModelPath.string().c_str());
62    TFLITE_CHECK(model != nullptr);
63
64    tflite::ops::builtin::BuiltinOpResolver resolver;
65    tflite::InterpreterBuilder builder(*model, resolver);
66    std::unique_ptr<tflite::Interpreter> interpreter;
67    builder(&interpreter);
68
69    TFLITE_CHECK(interpreter != nullptr);
70
71  const auto StartMemoryUsage = tflite::profiling::memory::GetMemoryUsage()
      ;
72
73    // Allocate tensor buffers.
74    TFLITE_CHECK(interpreter->AllocateTensors() == kTfLiteOk);
75    std::cout << "=== Pre-invoke Interpreter State ===" << std::endl;
```

```
76      tflite::PrintInterpreterState(interpreter.get());
77
78      const auto InitMemoryUsage = tflite::profiling::memory::GetMemoryUsage
        () - StartMemoryUsage;
79
80      std::cout << "Pre-invoke memory usage: " << InitMemoryUsage.max_rss_kb
        / 1024.0 << " MB" << std::endl << std::endl;
81      OutputFile << "Pre-invoke memory usage: " << InitMemoryUsage.max_rss_kb
         / 1024.0 << " MB" << std::endl << std::endl;
82
83      std::string ImgName;
84
85      for(auto const& dir_entry: fs::directory_iterator(SourceImgPath))
86      {
87          auto Inicio = std::chrono::steady_clock::now();
88
89          fs::path ImagePath = fs::path(dir_entry);
90          cv::Mat Img = cv::imread(ImagePath, Mode);
91
92          int OriginalImgSize = Img.rows;
93
94          cv::Mat ImgConverted;
95          Img.convertTo(ImgConverted, CV_32FC3);
96
97          cv::Mat ImgResized;
98          cv::resize(ImgConverted, ImgResized, cv::Size(ImgSize, ImgSize), 0,
         0, cv::INTER_AREA);
99
100         ImgResized /= 255.0;
101
102         float* inputImg_ptr = ImgResized.ptr<float>(0);
103
104         float* inputLayer = interpreter->typed_input_tensor<float>(0);
105
106         memcpy(inputLayer, inputImg_ptr, ImgSize * ImgSize * Channels *
        sizeof(kTfLiteFloat32));
107
108         TFLITE_CHECK(interpreter->Invoke() == kTfLiteOk);
109         const auto MemoryUsage = tflite::profiling::memory::GetMemoryUsage
        ()- StartMemoryUsage;
110
```

```
111         float* outputLayer = interpreter->typed_output_tensor<float>(0);
112 
113         cv::Mat Prediction = cv::Mat::zeros(ImgSize, ImgSize, CV_32FC1);
114         float* outputImg_ptr = Prediction.ptr<float>(0);
115 
116         memcpy(outputImg_ptr, outputLayer, ImgSize * ImgSize * sizeof(
    kTfLiteFloat32));
117 
118         Prediction *= 255;
119 
120         cv::Mat PredictionConverted;
121         Prediction.convertTo(PredictionConverted, CV_8UC1);
122 
123         cv::Mat ProcesedPrediction = cv::Mat();
124         cv::threshold(PredictionConverted, ProcesedPrediction, 127, 255, cv
    ::ThresholdTypes::THRESH_BINARY);
125 
126         cv::Mat ImgFinalResized;
127         cv::resize(ProcesedPrediction, ImgFinalResized, cv::Size(
    OriginalImgSize, OriginalImgSize), 0, 0, cv::INTER_NEAREST);
128 
129         ImgName = ImagePath.filename();
130 
131         fs::path SaveImgPath = DestinationImgPath / fs::path(ImgName);
132 
133         cv::imwrite(SaveImgPath, ImgFinalResized);
134 
135         auto Fin = std::chrono::steady_clock::now();
136         std::cout << " " << ImgName << std::endl;
137         std::cout << " Prediction time:  "<< std::chrono::duration <double,
     std::milli> (Fin-Inicio).count() << " ms " << std::endl;
138         std::cout << " Post-invoke memory usage: " << MemoryUsage.
    max_rss_kb / 1024.0 << " MB" << std::endl << std::endl;
139 
140         OutputFile << " " << ImgName << std::endl;
141         OutputFile << " Prediction time:  "<< std::chrono::duration <double
    , std::milli> (Fin-Inicio).count() << " ms " << std::endl;
142         OutputFile << " Post-invoke memory usage: " << MemoryUsage.
    max_rss_kb / 1024.0 << " MB" << std::endl << std::endl;
143     }
144 
```

```
145     auto finish = std::chrono::steady_clock::now();
146     auto diff = finish - start;
147 
148     std::cout << "Execution time: " << std::chrono::duration <double, std::
        milli> (diff).count() << " ms" << std::endl;
149     OutputFile << "Execution time: " << std::chrono::duration <double, std
        ::milli> (diff).count() << " ms" << std::endl;
150 
151     OutputFile.close();
152 
153     return 0;
154 }
```

Listing A.1: Implementación del modelo C++

Apéndice B

Artículo de investigación

Sistema *edge computing* para la detección de *Lobesia Botrana* mediante imágenes

José Luis Mira Serrano

Universidad Internacional de la Rioja, Logroño (España)

20-Febrero-2022

RESUMEN

En la actualidad, la recogida de datos, que permiten estimar la población de insectos, se realiza por observación directa y recuento manual de los mismos. Los técnicos de prevención de plagas cruzan esta información obtenida con otras variables de tipo ambiental (temperatura, humedad, etc.), alimentando de esta forma un modelo de predicción el cual permite adelantarse a épocas de riesgo para poder aplicar las medidas preventivas correspondientes. Otro beneficio a tener en cuenta de la aplicación de estos modelos es la capacidad de ajustar los tratamientos fitosanitarios con el consiguiente ahorro para los agricultores e impacto positivo en el medio ambiente.

La tarea de recuento requiere de un gran esfuerzo y cantidad de mano de obra, lo que también lleva en ocasiones a limitar su aplicabilidad debido a los altos costes del proceso.

Este TFM pretende desarrollar un algoritmo de inteligencia artificial y visión por computador, que sera desplegado en un sistema Raspberry Pi. El resultado supone un primer paso para automatizar este tipo de tareas.

PALABRAS CLAVE

Inteligencia Artificial, Control de Plagas, Sistemas *on the edge*, Visión por Computador

I. INTRODUCCIÓN

Vivimos en plena era de la información. A día de hoy la humanidad está sufriendo uno de los mayores cambios en su historia, equiparable a los cambios sociales que supuso la aparición de la agricultura o la revolución industrial. Las tecnologías de la información han revolucionado por completo nuestra forma de comportarnos e interactuar con nuestro entorno, tanto individual como colectivamente.

Los diferentes sectores económicos avanzan y evolucionan rápidamente hacia un mundo dominado por la tecnología. Este proceso de innovación hace que todos deban adaptarse para evitar quedarse obsoletos y mejorar la calidad de vida de las personas que se dedican a estos. Sin embargo, a veces hay lugares donde la utilización de estas tecnologías no está demasiado arraigada, ya sea por la falta de iniciativa para implantarlas o por el desconocimiento de las mismas. En esta situación se encuentra, aunque cada vez menos, la mayor parte del mundo rural.

Entre los sectores más olvidados tenemos el agrícola, ganadero, pesca, silvicultura… y entre ellos también encontramos al vitivinícola, sector entorno al que girará este proyecto. Todos los mencionados anteriormente son pertenecientes al sector primario, piedra fundamental de nuestra economía, es por ello que se debería tener más en cuenta a la hora de invertir en avances tecnológicos ya que dicha inversión puede garantizar una reducción en los gastos de producción, aumentando así su rentabilidad y potenciando la economía, además de mantener

activa una parte tan amplia cómo es el mundo rural que se mantiene gracias a empleos relacionados con el sector primario.

La Unión Europea es el principal productor de vino del mundo, centrándonos sobre todo en Francia, Italia y España, representa el 45 % de la superficie vitícola mundial, el 65 % de la producción y además el 70 % de las exportaciones, dentro de la Unión Europea los países más importantes son Francia, Italia y España [1].

II. ESTADO DEL ARTE

Las técnicas actuales relacionadas con la detección y prevención de una plaga de insectos conllevan un despliegue en el cual se deben instalar una serie de trampas denominadas cromáticas, con el fin de analizar el volumen de insectos presentes en el medio. Posteriormente esas mismas trampas se recogen y analizan en un laboratorio por un técnico contando y clasificando los insectos de manera totalmente manual. El número de trampas crece en relación a la superficie a cubrir, y a la cantidad de densidad de las mismas por metro cuadrado.

Este proceso manual, en los casos en los que quiere analizar una gran cantidad de terreno o zonas con una alta densidad de trampas por metro cuadrado, puede llegar a convertirse en una tarea muy tediosa para los técnicos agrícolas. Además, el tiempo que transcurre entre el despliegue de los resultados obtenidos puede ser demasiado elevado para poder actuar con rapidez contra una plaga, esto repercute con la aparición de enormes pérdidas del producto.

La Visión por Computador es una de las ramas dentro de la inteligencia artificial que mayor crecimiento ha tenido en los últimos años. La definimos como la disciplina que estudia cómo procesar, analizar e interpretar imágenes de forma automática. Estas técnicas tienen aplicaciones en muchos ámbitos, como la seguridad, la medicina, la inspección automática, o la navegación automática. La visión por computador es un campo de estudio, el cual desarrolla técnicas que ayuden a los ordenadores a "ver" por sí solos. El campo de desarrollo de la misma abarca las funciones de captura, interpretación y procesamiento de los objetos que son perceptibles visualmente. Existen varios retos dentro de la visión por computador, pero el principal es el reconocimiento de objetos [2].

Las redes neuronales artificiales son sistemas de computación inspirados por las neuronas biológicas que constituyen los cerebros en el reino animal. Una red neuronal artificial es por tanto una colección de neuronas artificiales las cuales se interconectan entre ellas organizándose en capas. Estas se suelen dar de forma que existe una de entrada, una de salida y varias capas ocultas. La capa de entrada recibe y procesa las señales de entrada y envía la señal de salida a la siguiente capa de neuronas de la red. Cada una de estas neuronas se pueden conectar a las neuronas de una de las capas adyacentes.

Dentro de las redes neuronales artificiales existes diversas variantes enfocadas a resolver problemas de distinto tipo. Las redes neuronales convolucionales son igual que las redes neuronales artificiales en cuanto a que están compuestas por neuronas que se van auto calibrando durante el entrenamiento, estas redes está inspiradas en la organización de los campos receptivos de las neuronas de la corteza visual primaria (v1) de un cerebro biológico humano. Es por esto que son utilizadas principalmente en aplicaciones de visión artificial. En los últimos años este tipo de redes ha tenido un gran crecimiento en campos relacionados con el reconocimiento de patrones, del procesamiento de imágenes al reconocimiento de voz.

Las redes neuronales convolucionales han sido utilizadas de manera generalizada para la clasificación de diferentes tipos de imágenes. Entre estos tipos también encontramos problemas relacionados con el mundo de la agricultura como la identificación automática de diversas especies de insectos [3] [4] [5] [6] o de enfermedades relacionadas con los cultivos [4]. Estos trabajos pretenden resolver el problema en el que mediante la captura de imágenes se pueda identificar la presencia de determinados elementos de interés como insectos nocivos o enfermedades botánicas. Otro de los campos de estudio de las redes neuronales convolucionales es la segmentación semántica de imágenes. La

segmentación semántica es un proceso que busca asignar una etiqueta categórica a cada uno de los píxeles contenidos en una imagen [7].

El IoT es un concepto que hace referencia a la interconexión, haciendo uso de Internet, de distintos objetos cotidianos, los cuales contienen un pequeño procesador con que mejorar sus capacidades. Las aplicaciones introducidas en estos dispositivos se centran principalmente en recoger datos del ambiente relativos a la funcionalidad de las mismas, para procesarlos y realizar una estimación óptima de una serie de recursos o bien poder realizar un control remoto de estos dispositivos. Uno de los dispositivos IoT más famosos y utilizados son las placas basadas en un SoC ARM fabricadas por la compañía Raspberry Pi Foundation [8].

III. OBJETIVOS Y METODOLOGÍA

El objetivo general de este proyecto consiste en el desarrollo de un sistema el cual implemente un algoritmo de procesamiento de imágenes que, mediante técnicas de segmentación e IA, realice una estimación del número de insectos atrapados en una serie de trampas cromáticas. Siendo concretos el proyecto se engloba dentro de la construcción de un sistema mas grande el cual va destinado a realizar un control sobre la plaga de la polilla de la vid.

Esta implementación se desplegaría en un sistema empotrado con capacidad de comunicación limitada y alimentado por batería. El procesamiento realizado reduciría la cantidad de información a transmitir, solo zonas de interés de la imagen o datos numéricos sobre las mismas.

Este objetivo se subdivide en objetivos específicos, a saber:

A. Análisis de las tecnologías potenciales a utilizar

Teniendo en cuenta cuáles son los requisitos funcionales y no funcionales que tiene que cumplir nuestro sistema, se tiene que analizar qué tecnologías serían las más adecuadas para su implementación en el proyecto. Es necesario evaluar el rendimiento proporcionado por estas tecnologías en relación con el rendimiento ideal que vamos a necesitar.

B. Estudio de las técnicas de análisis de imagen mediante IA

Estudio de las viabilidad de aplicación de diferentes tecnologías basadas en IA para poder realizar la segmentación y análisis de las imágenes obtenidas de las trampas cromáticas.

A partir de estas imágenes se deberá realizar un procesamiento que proporcione como resultado una estimación de la presencia o no de *Lobesia Botrana* en las imágenes.

Python es un lenguaje de programación ampliamente conocido tanto por su versatilidad, como por su fácil curva de aprendizaje. Estas características lo convierten en un candidato ideal para poder llevar a cabo de forma rápida un prototipo del propio sistema, pudiendo variar los parámetros del mismo e ir haciendo comprobaciones de los resultados obtenidos mediante el método de ensayo y error.

C. Adquisición de imágenes y creación del conjunto de datos de entrenamiento

Los algoritmos de IA que sirven para la clasificación y segmentación de imágenes suelen ser implementados a partir de un entrenamiento supervisado con un conjunto de ejemplos a partir de los cuales puedan aprender una serie de patrones dados.

Por esto es clave obtener una serie de muestras representativas y generar un conjunto de datos para entrenar estos algoritmos. Las técnicas de *data augmentation* suelen ser claves en la consecución de este paso.

D. Adaptación del algoritmo a un sistema empotrado

Python es una buena opción por su rapidez a la hora de poder prototipar, pero no destaca por su velocidad ni eficiencia energética. C++ es un lenguaje ampliamente utilizado en siste-

mas empotrados por su eficiencia y capacidad de adaptación.

En nuestro caso es un requisito crítico que el sistema consuma poca energía. Por lo tendremos que adaptar la funcionalidad obtenida en el prototipo para que funcione con la máxima eficiencia posible.

E. Optimización del modelo de IA para su utilización en un sistema empotrado

Los sistemas basados en inteligencia artificial suelen ejecutarse con el apoyo de aceleración hardware como GPU debido a la alta cantidad de parámetros que contienen estos. Para realizar la inferencia de un modelo a partir de una entrada que se le proporciona se deben realizar una alta cantidad de operaciones en punto flotante (Float16 y Float32) y con forma matricial. Las CPU utilizadas habitualmente en la implementación de sistemas empotrados, no están preparadas para realizar operaciones con este tipo de datos de manera eficiente.

Es clave la adaptación de este modelo en base a técnicas de cuantificación que transformen los tipos de datos que se utilizan internamente de punto flotante a entero.

F. Analizar desempeño y valoración de requisitos no funcionales como consumo o recursos utilizados

Por ultimo, hemos de tener en cuenta los recursos utilizados en nuestro modelo, así como los tiempos de ejecución. Y en caso de que alguno de esos parámetros no sean satisfactorios volveríamos a la etapa de optimización del algoritmo.

IV. CONTRIBUCIÓN

La contribución de este proyecto se divide entre el análisis de una arquitectura neuronal que tenga la capacidad de resolver el problema de segmentación semántica contemplado y la adaptación de este sistema a un dispositivo *on the edge*.

A. Análisis arquitecturas neuronales

La necesidad principal de este proyecto es la segmentación de imágenes, por tanto se ha tomado la decisión de escoger la utilización de una arquitectura del tipo Encoder-Decoder. Además el diseño de la arquitectura de la red esta inspirado por otras que ya han sido probadas en el campo de la clasificación y segmentación de imágenes con buenos resultados, como pueden ser AlexNet y SegNet. En la Figura 1 se puede ver la representación gráfica de la arquitectura de la misma, dentro de la cual se pueden ver dos partes claramente diferenciadas. El Encoder en colores magenta y verde en el que los datos van reduciendo sus dimensiones espaciales y aumentando su profundidad y el Decoder en el que se produce el efecto contrario hasta obtener una salida con las mismas características espaciales que la entrada.

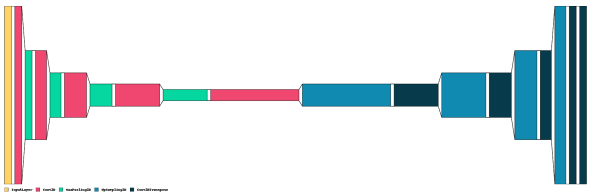


Figura 1: Arquitectura empleada en la red neuronal convolucional

En esta arquitectura se han evaluado diferentes cambios en los parámetros de la misma y en el formato de las imágenes de entrada para determinar que combinación de características proporciona los mejores resultados. Para la evaluación de la calidad de las predicciones se han empleado métricas habitualmente utilizadas en los problemas de clasificacion como: *accuracy*, *precision*, *recall* y *F1*. A estas métricas también se ha añadido una basada en la diferencia existente entre el numero de insectos detectados en la prediccion y el numero de insectos real.

$$Error\ medio$$

$$\frac{\sum_1^n |N^\circ\ insectos\ Ground\ Truth - N^\circ\ insectos\ prediccin|}{n}$$

B. Implementación *on the edge*

Dado que el objetivo de este proyecto es la creación de un sistema empotrado el cual sea capaz de realizar este procesamiento, uno de los pasos necesarios es la adaptación del modelo entrenado a este tipo de dispositivos.

Habitualmente los modelos de *deep learning* utilizan tipos de datos numéricos en punto flotante de 16 o 32 bits con los que realizar las operaciones internas de inferencia. Realizar una cuantización de estos tipos de datos a enteros con 8 bits de precisión es uno de los pasos para optimizar este tipo de . Esta cuantización supone un proceso de discretización de las variables involucradas en las operaciones internas del modelo, con esto se definen rangos en los que las variables contenidas dentro de los mismos pasarán a tener un mismo valor identificando el rango al que pertenecen. Esta operación se realiza debido a que las CPU de sistemas empotrados, son mucho más eficientes trabajando con números enteros, incluso algunos microcontroladores solo tienen la capacidad de operar con este tipo de datos.

Se ha optado por escoger el lenguaje de programación C++ para realizar la implementación del sistema de inferencia empotrado, debido al mayor rendimiento y eficiencia en el uso de recursos que hace este con respecto a otros lenguajes. Sumado a esto, también se ha hecho uso de las librerías OpenCV y TensorFlow Lite.

V. RESULTADOS O EVALUACIÓN

El analisis de resultados se va a dividir entre las pruebas realizadas con la arquitectura neuronal y las conclusiones obtenidas de la implementacion del sistema *on the edge*.

A. Arquitectura neuronal

El entrenamiento de la red neuronal se ha realizado en base a un *dataset* de 1200 imágenes obtenidas mediante el uso de tecnicas de *data augmentation*. A pesar de ello las metricas se han obtenido a partir de la prediccion sobre dos imagenes independientes que no forman parte del *dataset* de entrenamiento. Las pruebas sobre el rendimiento de la arquitectura se estructuran en dos grupos, con imágenes de una única dimensión en escala de grises y con imágenes de 3 dimensiones RGB. El objetivo es comprobar las diferencias entre ambas en cuanto a la calidad de la segmentación obtenida. La utilización de imágenes en escala de grises puede suponer un ahorro de memoria y de tiempo de ejecución, con respecto a imágenes RGB, en determinadas tareas relacionadas con la adquisición y el preprocesamiento de las mismas antes de entrar al proceso de predicción en la red neuronal.

Otro de los factores a tener en cuenta es el propio tamaño de las imágenes, este tamaño influye de manera directa en los recursos necesarios para realizar la inferencia con la red neuronal, tanto en tiempo necesario para realizar las operaciones de convolución como en la memoria de almacenamiento empleada para las capas de imágenes que se van generando durante la misma. Por ello se han realizado pruebas para determinar la calidad de la predicción de la red utilizando imágenes de entrada de 3 tamaños diferentes: 1024x1024, 512x512 y 256x256. Estos tamaños se han escogido debido a que los diferentes tamaños producidos en base a las reducciones de tamaño internas de la red siguen conservando las características de ser potencias de 2. Los resultados han sido obtenidos con una configuración de entrenamiento de 8 *batchs* y 100 *epochs*.

Las pruebas han mostrado que de manera generalizada existe una desmejora entre los resultados obtenidos con las imágenes en escala de grises y RGB de aproximadamente 10 puntos en la métrica F1. Las imágenes en escala de grises se sitúan en torno al 45 % en la métrica *F1* mientras que en formato RGB se ubican en el 55 %.

Con tal de mejorar los resultados obtenidos se han realizado pruebas en el entrenamiento de la red neuronal aplicando técnicas de *dropout* con tal de evitar posibles casos de *overfitting* en la misma. Estas pruebas se han realizado con 3 valores diferentes: 0.2, 0.3 y 0.5. Los resultados obtenidos muestran que los valores

0.3 y 0.5 reducen la calidad del modelo existente, siendo de esta forma 0.2 el único valor capaz de conseguir mejorar los resultados obtenidos previamente. A mayor valor de *dropout* peores resultados se obtienen, siendo esta diferencia de hasta el 20,9 puntos en la métrica *F1* entre el modelo entrenado con un valor de 0,5 y otro con un valor de 0,2. Este ultimo modelo supera los resultados obtenidos previamente en 8,3 puntos en la métrica *F1*.

B. Implementación *on the edge*

La implementación del sistema *on the edge* se ha llevado a cabo utilizando el lenguaje de programación C++ con las librerias OpenCV y TensorFLow Lite. Habitualmente los modelos de *deep learning* utilizan tipos de datos numéricos en punto flotante de 16 o 32 bits con los que realizar las operaciones internas de inferencia. Una de las posibilidades que nos ofrece TensorFlow Lite es la de producir una cuantización de estos tipos de datos a enteros con 8 bits de precisión. Esta cuantización supone un proceso de discretización de las variables involucradas en las operaciones internas del modelo, con esto se definen rangos en los que las variables contenidas dentro de los mismos pasarán a tener un mismo valor identificando el rango al que pertenecen. Esta operación se realiza debido a que las CPU de sistemas empotrados, son mucho más eficientes trabajando con números enteros, incluso algunos microcontroladores solo tienen la capacidad de operar con este tipo de datos.

Evaluando las diferencias entre el uso de memoria utilizado por un modelo sin cuantizar y otro cuantizado observamos que solo existe una reducción de entre el 1 % y el 6 %. En función del tamaño de las imágenes de entrada, en la Tabla 1 se muestra la memoria utilizada por cada uno de estos modelos. En cuanto a diferencias en la calidad de las predicciones se estima una perdida media con la cuantizacion de entorno al 15 % en la métrica *F1*, siendo mas acusada esta perdida cuanto mayor es el tamaño de la imagen.

La ejecución de estos modelos se ha evaluado en 3 dispositivos Raspberry Pi: 4, 3 B, Zero 2 W. En cada uno de ellos se ha medido el tiempo empleado para ejecutar un modelo y el consumo energético del mismo. Tras realizar las pruebas se ha comprobado que la Raspberry Pi 4 es el único dispositivo capaz de ejecutar los modelos de los 3 tamaños distintos debido a las capacidades de su hardware.

256	**512**	**1024**
380 MB	1305 MB	4992 MB
358 MB	1279 MB	4964 MB

Cuadro 1: Memoria consumida por los modelos

256	**512**	**1024**
1,85 s	5,89 s	108 s

Cuadro 2: Tiempo de ejecución en Raspberry Pi 4

De la misma manera también obtiene los mejores resultados en cuanto a consumo energético, dado la relación existente entre el tiempo de ejecución y este.

256	**512**	**1024**
0,32 MAH	1,14 MAH	18 MAH

Cuadro 3: Consumo energético en Raspberry Pi 4

VI. DISCUSIÓN O ANÁLISIS DE RESULTADOS

Tras la realizacion del trabajo se ha comprobado la viabilidad en la implantacion de este sistema en un dispositivo Raspberry Pi 4 tanto en el ambito del consumo energetico como en la calidad de los resultados obtenidos.

En cuanto al diseño del modelo nos encontramos que la mejor predicción se ha dado con imágenes de tamaño 512x512, formato de color RGB y *dropout*=0,2. En la Figura 5 podemos observar la predicción obtenida para una de las imágenes del conjunto de evaluación.

VII. CONCLUSIONES

El objetivo de este proyecto trataba sobre el desarrollo de un sistema *edge computing* para la detección de *Lobesia Botrana*, en el cual se implementará un sistema que tuviera la capacidad de estimar el número de insectos utilizando técnicas de visión por computador e inteligencia artificial. Para evaluar la consecución global del proyecto vamos a analizar los objetivos parciales propuestos.

Objetivos específicos cumplidos:

- **Análisis de las tecnologías potenciales a utilizar**: Se ha realizado un análisis de las tecnologías necesarias que han sido empleadas para la consecución del proyecto. En cuanto al hardware se ha determinado la necesidad de tener un dispositivo el cual fuese capaz de realizar la adquisición y procesamiento de imágenes con un reducido consumo energético, del que se ha determinado que los sistemas Raspberry Pi son candidatos ideales para realizar estas funciones. En la parte de diseño software del sistema se optó por escoger OpenCV para el procesamiento de imagen y TensorFlow para la inteligencia del sistema debido a que ambos cumplían con las capacidades necesarias para solventar los requisitos asociados al proyecto.

- **Estudio de las técnicas de análisis de imagen mediante IA**: Se han estudiado diferentes alternativas que podrían servir para afrontar la solución del problema. De las cuales se ha optado por el empleo de una arquitectura de redes neuronales convoluciones de tipo Encoder-Decoder. Se ha elegido esta arquitectura por ser usada en otros problemas de segmentación semántica con buenos resultados. Además se han hecho varias pruebas con los parámetros de la arquitectura para determinar que combinaciones de estos aportan mejores resultados. En el contexto del proyecto, la utilización de imágenes en formato RGB sumadas a una regularización de los parámetros mediante *dropout* ha demostrado obtener los mejores resultados.

- **Adquisición de imágenes y creación del conjunto de datos de entrenamiento**: Las imágenes con las que se ha trabajado en el problema se han obtenido a partir de trampas cromáticas en una explotación agrícola de vid en la provincia de Ciudad Real. El hecho de tener un conjunto reducido de imágenes ha hecho necesaria la utilización de técnicas de *data augmentation* con las que incrementar este.

- **Adaptación del algoritmo a un sistema empotrado**: Se ha realizado una implementación en C++ del sistema para realizar el análisis de nuevas imágenes adquiridas en un sistema Raspberry Pi.

- **Optimización del modelo de IA para su utilización en un sistema empotrado**: Se ha realizado la conversión de los modelos entrenados en TensorFlow a TensorFlow Lite para su utilización en sistemas móviles con una menor cantidad de recursos. Además también se han aplicado técnicas de cuantización para modificar los datos internos del modelo de punto flotante a entero y así ejecutarse de una forma más eficiente en microcontroladores. Tras las pruebas realizadas se ha determinado que las CPU de los dispositivos utilizados son lo suficientemente avanzadas como para realizar operaciones con datos vectoriales en punto flotante y por tanto no existe una diferencia significativa en la cuantización o no de los parámetros.

- **Analizar desempeño y valoración de requisitos no funcionales como consumo o recursos utilizados**: Se ha realizado un análisis exhaustivo de las diferentes configuraciones de los modelos para realizar la inferencia de imágenes. Se ha analizado la memoria utilizada por cada una de las configuraciones de los modelos para realizar la inferencia así como el consumo de los mismos en tiempo y energía eléctrica en 3 dispositivos Raspberry Pi diferentes. Las pruebas demostraron que el modelo Raspberry Pi 4 fue el único con

la potencia y capacidad de memoria necesarias para poder afrontar la ejecución de todas las configuraciones posibles de los modelos. Además este dispositivo también ha sido el mejor en términos de eficiencia energética, descartando para afrontar el problema a los modelos Raspberry Pi 3 B y Raspberry Pi Zero 2 W.

Por todas estas razones se considera que se han cumplido los requisitos establecidos para la consecución del proyecto. Para futuros trabajos de investigación, las líneas de mejora que se pueden llevar a cabo en este proyecto son las siguiente:

- **Aumento del conjunto de datos**: Incrementar el número de imágenes utilizadas en el entrenamiento de los modelos es clave para poder tener una mejor robustez y precisión en estos.
- **Módulo de comunicaciones**: La implementación de un modulo de comunicaciones proporcionaría la capacidad de poder obtener resultados del procesamiento realizado de manera remota, adquirir nuevas imágenes para incrementar el conjunto de datos o poder realizar mejoras de manera remota en el sistema. Estas actualizaciones principalmente podrían tener el cometido de actualizar el modelo utilizado para la inferencia con nuevas versiones mejoradas.

Referencias

[1] Comision Europea. Apoyo y protección de los viticultores, productores de vino, vendedores y consumidores de la ue a través de políticas, legislación, etiquetado, medidas comerciales y seguimiento de los mercados., 2020.

[2] Profesor/a UOC. La vision por computador: Una disciplina en auge, 2012.

[3] Yufeng Shen, Huiling Zhou, Jiangtao Li, Fuji Jian, and Digvir S Jayas. Detection of stored-grain insects using deep learning. *Computers and Electronics in Agriculture*, 145:319–325, 2018.

[4] Yong Ai, Chong Sun, Jun Tie, and Xiantao Cai. Research on recognition model of crop diseases and insect pests based on deep learning in harsh environments. *IEEE Access*, 8:171686–171693, 01 2020.

[5] Haiyang He, Jing Wang, Shiguo Huang, and Xiaolin Li. A comparative deep learning algorithms for agricultural insect recognition. In *2021 International Conference on Machine Learning and Intelligent Systems Engineering (MLISE)*, pages 321–325. IEEE, 2021.

[6] Balakrishnan Ramalingam, Rajesh Elara Mohan, Sathian Pookkuttath, Braulio Félix Gómez, Charan Satya Chandra Sairam Borusu, Tey Wee Teng, and Yokhesh Krishnasamy Tamilselvam. Remote insects trap monitoring system using deep learning framework and iot. *Sensors*, 20(18):5280, 2020.

[7] Alberto Garcia-Garcia, Sergio Orts-Escolano, Sergiu Oprea, Victor Villena-Martinez, and Jose Garcia-Rodriguez. A review on deep learning techniques applied to semantic segmentation. *arXiv preprint arXiv:1704.06857*, 2017.

[8] Raspberry Pi Foundation. Raspberry pi, 2022.

[b]0.3

Figura 2: Imagen original

[b]0.3

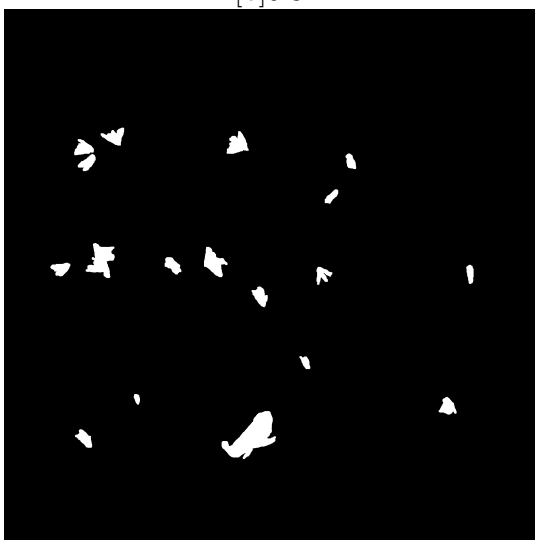


Figura 3: Ground thruth

[b]0.3

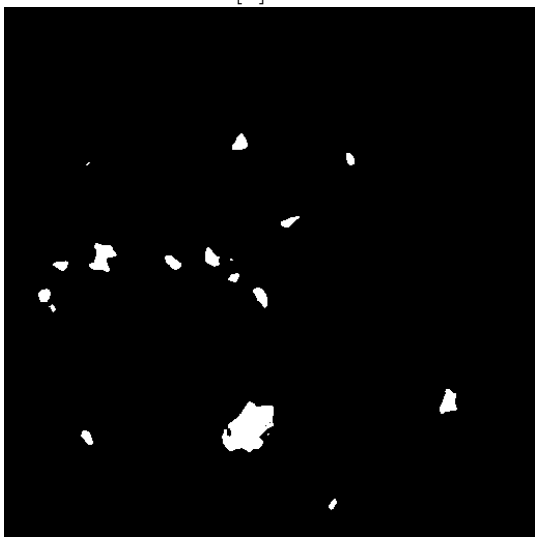


Figura 4: Predicción

Figura 5: Resultado predicción