



**viu**

**Universidad  
Internacional  
de Valencia**

# **Aplicación de técnicas de *Big Data* y Aprendizaje Automático para el moni- toreo de cultivos**

Titulación:  
Máster en *Big Data* y  
Ciencia de Datos

Curso Académico  
10-2022

Alumno/a: Moratalla  
Muñoz, José Ramón  
DNI: 06286194H

Director/a del TFT:  
Cardinale Coromoto, Yudith

Convocatoria:

Primera

# Índice general

<b>Índice de figuras</b>	<b>3</b>
<b>Índice de tablas</b>	<b>4</b>
<b>Lista de Pseudocódigos</b>	<b>5</b>
<b>Resumen</b>	<b>6</b>
<b>1 Introducción</b>	<b>9</b>
1.1 Objetivos . . . . .	10
1.1.1 General . . . . .	10
1.1.2 Específicos . . . . .	11
1.2 Estructura del Trabajo . . . . .	12
<b>2 Una aproximación a la agricultura inteligente</b>	<b>14</b>
2.1 El paisaje tecnológico de la Industria 4.0 . . . . .	15
2.1.1 La era de los datos . . . . .	15
2.1.2 Un recorrido desde la Inteligencia Artificial hasta las Redes Neuronales Convolucionales . . . . .	16
2.2 Agricultura 4.0 o <i>Smart Agriculture</i> . . . . .	20
2.3 Trabajos relacionados con la agricultura inteligente . . . . .	22
<b>3 Metodología</b>	<b>24</b>
<b>4 Sistema de monitoreo de cultivos</b>	<b>26</b>
4.1 Fase 1: Procesamiento de datos en <i>streaming</i> . . . . .	27
4.1.1 Iteración 1: Elección del ingestor de datos . . . . .	27
4.1.2 Iteración 2: Elección del procesador de datos . . . . .	30
4.1.3 Iteración 3: Implementación de la ingesta de datos . . . . .	34
4.1.4 Iteración 4: Integración entre sensores, ingestor y procesador . . . . .	41
4.2 Fase 2: Detección de enfermedades de cultivos a partir de imágenes . . . . .	44
4.2.1 Iteración 1: Búsqueda del conjunto de datos . . . . .	44
4.2.2 Iteración 2: Carga e inspección del conjunto de datos . . . . .	45
4.2.3 Iteración 3: División de los datos de trabajo y desarrollo de funciones auxiliares . . . . .	47
4.2.4 Iteración 4: Entrenamiento de los modelos . . . . .	49
<b>5 Evaluación</b>	<b>58</b>
5.1 Procesamiento <i>streaming</i> de datos de calidad de suelo . . . . .	59
5.2 Detección de enfermedades de cultivos a partir de imágenes . . . . .	60
5.2.1 Evaluación del modelo 1 . . . . .	61
5.2.2 Evaluación del modelo 2 . . . . .	61
5.2.3 Evaluación del modelo 3 . . . . .	62
5.2.4 Evaluación del modelo 4 . . . . .	63
5.2.5 Resumen de los entrenamientos de modelos . . . . .	64

5.2.6	Cumplimiento de requisitos . . . . .	65
<b>6</b>	<b>Conclusiones y trabajos futuros</b>	<b>66</b>
6.1	Trabajos futuros . . . . .	66
6.1.1	Subida del sistema al <i>Cloud</i> . . . . .	67
6.1.2	Uso de sensores IoT reales para la recogida de datos . . . . .	67
6.1.3	Aumentar la cantidad de datos de entrada para el sistema de procesamiento . . . . .	67
6.1.4	Aumentar los cultivos a los que se pueda detectar enfermedades	68
6.1.5	Integración con otros TFM	68
<b>A</b>	<b>Apéndice: Repositorio del trabajo</b>	<b>73</b>

# Índice de figuras

2.1	Estimación de población global de la ONU para 2050 . . . . .	14
2.2	Ejemplo funcionamiento MapReduce . . . . .	16
2.3	Ejemplo arquitectura CNN . . . . .	19
2.4	Esquema visual de términos definidos . . . . .	19
2.5	Esquema visual de <i>smart farming</i> . . . . .	21
3.1	Esquema de la metodología adhoc abordada . . . . .	25
4.1	Boceto de solución deseada . . . . .	26
4.2	Figura 4 de Lopez et al. (2016) . . . . .	32
4.3	Figura 5A (izquierda) y 5B (derecha) de Lopez et al. (2016) . . . . .	33
4.4	Figura 6A (izquierda) y 6B (derecha) de Lopez et al. (2016) . . . . .	33
4.5	Información estación XMS-CAT Clarella . . . . .	35
4.6	Ejemplo de datos con formato <i>headers+values</i> de la estación Clarella . . . . .	35
4.7	Explicación y diferenciación de los campos del nombre de los ficheros . . . . .	36
4.8	Explicación y diferenciación de los campos de las líneas de los ficheros . . . . .	36
4.9	Esquema del funcionamiento de los <i>topics</i> de Kafka y sus particiones . . . . .	38
4.10	Organización de <i>topics</i> y particiones adoptada para el trabajo . . . . .	39
4.11	Esquema funcionamiento Spark Structured Streaming . . . . .	42
4.12	Topología del modelo 1 . . . . .	52
4.13	Resultados del entrenamiento del modelo 1 . . . . .	52
4.14	Topología del modelo 2 . . . . .	53
4.15	Resultados del entrenamiento del modelo 2 . . . . .	53
4.16	Topología del modelo 3 . . . . .	55
4.17	Resultados del entrenamiento del modelo 3 . . . . .	55
4.18	Topología del modelo 4 . . . . .	56
4.19	Resultados del entrenamiento del modelo 4 . . . . .	56
5.1	Boceto de soluciones alcanzadas . . . . .	58
5.2	Datos enviados por el productor . . . . .	60
5.3	Salida escrita por el consumidor . . . . .	60
5.4	Precisión del modelo 1 con los datos de test . . . . .	61
5.5	10 predicciones aleatorias hechas con el modelo 1 . . . . .	61
5.6	Precisión del modelo 2 con los datos de test . . . . .	62
5.7	10 predicciones aleatorias hechas con el modelo 1 . . . . .	62
5.8	Precisión del modelo 3 con los datos de test . . . . .	62
5.9	10 predicciones aleatorias hechas con el modelo 3 . . . . .	63
5.10	Precisión del modelo 4 con los datos de test . . . . .	63
5.11	10 predicciones aleatorias hechas con el modelo 4 . . . . .	64

# Índice de tablas

4.1	Resumen de los ingestores de datos . . . . .	30
4.2	Resumen de los motores de procesamiento . . . . .	33
4.3	Número de imágenes en cada categoría de la uva . . . . .	46
4.4	Hiperparámetros de entrenamiento del modelo 1 . . . . .	51
4.5	Hiperparámetros de entrenamiento del modelo 2 . . . . .	51
4.6	Hiperparámetros de entrenamiento del modelo 3 . . . . .	54
4.7	Hiperparámetros de entrenamiento del modelo 4 . . . . .	54
4.8	Resumen de hiperparámetros y entrenamiento de todos los modelos . . . . .	57
5.1	<u>Resumen de parámetros entrenables y precisión de los modelos . . . . .</u>	64

# Listado de Pseudocódigos

1	read_and_send_data()	40
2	Consumer Main Method	43

## Resumen

La agricultura inteligente se antoja como una de las posibles soluciones a la falta de recursos, más concretamente, de productos agrícolas a la que se estima que el ser humano se enfrentará en los próximos años debido a la superpoblación. En consecuencia, el número de granjas o explotaciones que están dando pasos en esa dirección ha aumentado a lo largo de los últimos años.

La aplicación de tecnologías como el Internet de las Cosas y el *Big Data* al sector agrícola, podría permitir llevar a dicho sector al siguiente nivel en términos de eficiencia y productividad. A todo esto, habría que sumar también tecnologías como el Aprendizaje Profundo y, más concretamente, las Redes Neuronales Convolucionales cuyas aplicaciones, especialmente la visión por computador, tienen también bastante que aportar al sector.

En este trabajo se propone una primera aproximación a la agricultura inteligente enfocada en dos aspectos principales: procesamiento de datos en tiempo real y detección de enfermedades de cultivos a partir de imágenes. Para llevar a cabo el procesamiento en *streaming*, se ha utilizado Kafka como ingestor de datos y Spark Structured Streaming para aplicar las transformaciones necesarias a los datos. Por otro lado, la detección de enfermedades de cultivos plantea un problema de clasificación de imágenes multiclase, el cual se ha resuelto utilizando distintos modelos de redes neuronales convolucionales desarrollados en Python 3 haciendo uso de la librería TensorFlow.

Puesto que cada solución da respuesta a diferentes problemas, se han evaluado de distinta manera. El modelo convolucional se ha evaluado por la precisión obtenida al trabajar con los datos de prueba. Por otro lado, en el caso del procesamiento de datos en tiempo real, la solución ha sido evaluada en función del cumplimiento de los objetivos planteados al inicio del trabajo. Los resultados obtenidos han sido muy positivos en ambos casos, cumpliendo con los objetivos planteados en el caso del procesamiento de datos en *streaming*, y habiendo obtenido modelos convolucionales con más de un 95 % de precisión para la tarea de detección de enfermedades de cultivos a partir de imágenes. Sin embargo, esto no supone más que una mera aproximación a la agricultura inteligente y numerosos trabajos futuros surgen a raíz de las soluciones planteadas.

**Palabras clave:** agricultura inteligente, internet de las cosas, procesamiento de datos en tiempo real, aprendizaje profundo, visión por computador, redes neuronales convolucionales, detección de enfermedades en cultivos a partir de imágenes.

## Abstract

Smart agriculture appears as one of the potential solutions to the shortage of resources, more specifically, agricultural products that it is estimated humanity will face in the coming years due to overpopulation. Consequently, the number of farms or enterprises taking steps in that direction has increased over the past years.

The application of technologies such as the Internet of Things (IoT) and Big Data to the agricultural sector could potentially elevate this sector to the next level in terms of efficiency and productivity. On top of that, we should also consider technologies like Deep Learning, and more specifically, Convolutional Neural Networks, whose applications, especially computer vision, also have much to contribute to the sector, should be added.

In this work, a preliminary approach to smart agriculture is proposed, focusing on two main aspects: streaming soil quality data processing and crop disease detection from images. To carry out streaming data processing of soil quality data, Kafka has been used as the data ingestor, and Spark Structured Streaming has been employed to apply the necessary data transformations. On the other hand, crop disease detection poses a multiclass image classification problem, which has been addressed using various convolutional neural network models developed in Python 3 by using the TensorFlow library.

Since each solution addresses different issues, they have been assessed differently. The convolutional model has been evaluated based on the accuracy achieved when working with test data. On the other hand, in the case of streaming data processing, the solution has been evaluated concerning the achievement of the objectives set at the beginning of the work. The results obtained have been very positive in both cases, reaching the objectives in the case of streaming data processing and achieving convolutional models with over 95% accuracy for the task of crop disease detection from images. However, this represents nothing more than a mere approach to smart farming, and numerous future works arise as a result of the proposed solutions.

**Palabras clave:** smart farming/agriculture, internet of things, streaming data processing, deep learning, computer/deep vision, convolutional neural networks, detection of crop diseases from images.

## Agradecimientos

En primer lugar, agradecer a la Universidad Internacional de Valencia por brindarme la oportunidad de aumentar mi formación mediante la realización de este máster.

Seguidamente, me gustaría hacer una mención especial para mi tutora Yudith por su dedicación y orientación durante el desarrollo de este trabajo. Siempre estuvo dispuesta a responder mis preguntas y proporcionar la experiencia necesaria para llevar este proyecto a buen puerto.

Por último, pero no menos importante, agradecer a mis familiares y amigos por apoyarme durante todo este proceso y ayudarme a desconectar cuando lo he necesitado.

# 1. Introducción

En los últimos años, con motivo de dar respuesta a la creciente demanda de productos agrícolas como resultado del continuo crecimiento de la población global, el sector agrícola se ha visto en la urgente necesidad de buscar soluciones en el desarrollo de la ciencia y la tecnología orientadas a mejorar la productividad y la eficiencia del sector (Araújo et al., 2021).

Esta búsqueda ha llevado a la industria agrícola a la adopción de tecnologías como el Internet de las Cosas (*IoT* por sus siglas en inglés) (Quy et al., 2022), el *Big Data* (Chergui and Kechadi, 2022) y el Aprendizaje Automático (*ML* por sus siglas en inglés) (Cravero et al., 2022). Como resultado de dicha adopción y, con la promesa de aplicar una nueva evolución en el sector revolucionando la gestión y producción de alimentos, aparece el concepto de agricultura inteligente o *smart agriculture/farming* (Moysiadis et al., 2021).

En este trabajo se aborda, por tanto, una primera aproximación a la agricultura inteligente y, debido a que las potenciales aplicaciones de las tecnologías anteriormente mencionadas en el sector agrícola son muy numerosas, para el desarrollo de este trabajo se han focalizado los esfuerzos en dos aplicaciones: procesamiento de datos en tiempo real y detección de enfermedades de cultivos a partir de imágenes.

Para el desarrollo de la solución de procesamiento de datos en tiempo real se utilizaron datos de la Red Internacional de Calidad de Suelo (*ISMN*<sup>1</sup> por sus siglas en inglés). La solución se basa, principalmente, en el uso de dos herramientas: Apache Kafka<sup>2</sup> como ingestor de datos y Apache Spark Structured Streaming<sup>3</sup> para el procesamiento de los datos ingestados por Kafka. Todo ello se ha integrado en un entorno Linux como es la distribución AlmaLinux<sup>4</sup>. Esta primera solución se evalúa en función de los objetivos definidos al inicio del proyecto para esta parte del trabajo.

En otro orden, la detección de enfermedades de cultivos a partir de imágenes plantea un claro problema de clasificación de imágenes multiclas. En este caso, puesto que hay múltiples cultivos y, dentro de cada uno de ellos, múltiples casos, se decide centrar el caso de uso en la detección de enfermedades de la vid. Para el desarrollo de esta solución, se ha utilizado Python 3 en Google Colab<sup>5</sup> y la librería de TensorFlow<sup>6</sup> para construir los modelos basados en redes neuronales convolucionales que dan respuesta al problema de clasificación de imágenes multiclas. En

<sup>1</sup><https://ismn.earth/en/>

<sup>2</sup><https://kafka.apache.org/>

<sup>3</sup><https://spark.incubator.apache.org/streaming/>

<sup>4</sup><https://almalinux.org/>

<sup>5</sup><https://colab.research.google.com/>

<sup>6</sup><https://www.tensorflow.org/?hl=es-419>

este caso, para evaluar los resultados obtenidos con dichos modelos, se ha tenido en cuenta la precisión obtenida con los mismos al trabajar con los datos de prueba.

Por último, es necesario remarcar que este trabajo nace dentro del proyecto de investigación UbiCDatos: Ciencia de Datos para ambientes de Computación UbiCua de la Universidad Internacional de Valencia<sup>7</sup>. Lo que dicho proyecto persigue es la aplicación de *frameworks* como Apache Hadoop<sup>8</sup>, Apache Spark<sup>9</sup> o Apache Flink<sup>10</sup>, según sea el caso, en sectores como la agricultura, la salud, la educación y el turismo. En este contexto, se establecieron los objetivos que se describen a continuación.

## 1.1. Objetivos

Tal y como se comentó anteriormente, este trabajo forma parte del proyecto de investigación UbiCDatos, por lo que los objetivos vienen, en parte, definidos por el mismo. Sin embargo, puesto que el alcance del proyecto de investigación abarca varios sectores y eso sería demasiado para un solo Trabajo Fin de Máster (TFM), ha sido necesario acotar dicho alcance, en el caso de este TFM concreto, al sector agrícola.

Llegados a este punto y habiendo acotado debidamente el alcance, a nivel de sector, del trabajo, se procede a definir los objetivos del mismo. Dicha definición consta de un objetivo general y varios objetivos específicos, siendo estos últimos un conjunto de hitos a superar para conseguir el objetivo general.

### General

Teniendo en cuenta todo lo anterior, se define como objetivo general implementar un prototipo de sistema que sirva como prueba de concepto para demostrar cómo el Internet de las Cosas, *Big Data*, Aprendizaje Profundo y Redes Neuronales Convencionales se integran para lograr una aproximación a la agricultura inteligente.

Este tipo de soluciones permiten mejorar la eficiencia y productividad la industria agrícola mediante la aplicación de las tecnologías anteriormente mencionadas, con la intención de preparar al sector agrícola ante problemas que la humanidad podría enfrentar en un futuro no muy lejano como la seguridad alimentaria y la superpoblación (Moysiadis et al., 2021).

<sup>7</sup><https://portaldeinvestigacionviu.com/en/main/projects?name=UbiCDatos>

<sup>8</sup><https://hadoop.apache.org/>

<sup>9</sup><https://spark.apache.org/>

<sup>10</sup><https://flink.apache.org/>

## Específicos

En este contexto y debido a las múltiples aplicaciones que las tecnologías mencionadas en el apartado anterior tienen en la industria agrícola, se decide centrar este TFM en dos aplicaciones concretas: procesamiento de datos en tiempo real y detección de enfermedades de cultivos a partir de imágenes. Dada esta bifurcación del trabajo en dos diferentes soluciones, se han definido los objetivos específicos teniendo en cuenta esta división.

### Procesamiento de datos en *streaming*

Para alcanzar el procesamiento de datos en tiempo real se han definido como objetivos específicos los siguientes:

- Evaluar varias tecnologías de ingesta de los datos y decidir cómo hacerla. La disposición o no de dispositivos que generen información, como, por ejemplo, sensores *IoT*, marca no sólo la manera de ingestar los datos, sino también los datos con los que trabajar.
- Elegir el ingestor de datos. De las herramientas y estrategias más comunes se selecciona una en base a sus funcionalidades y cómo las mismas se ajustan a los requisitos del sistema.
- Seleccionar un gestor de procesamiento de datos en *streaming*. De igual manera que en el caso de los ingestores, son múltiples las opciones que hay por lo que fue necesario evaluarlas y seleccionar una apropiada.
- Integrar el ingestor de datos y el gestor de procesamiento en *streaming*. De esta manera, se ha logrado obtener un prototipo de procesamiento de datos en tiempo real el cual, a su vez, puede alimentar cualquier sistema de visualización de datos.

### Detección de enfermedades de cultivos a partir de imágenes

Para llevar a cabo la detección de enfermedades de cultivos a partir de imágenes, los objetivos específicos que se proponen son:

- Elegir un *dataset* o conjunto de datos de imágenes de cultivos. Para dicha elección, se ha tenido en cuenta que el conjunto de datos se ajustase al problema que se quería resolver. En este caso concreto, se plantea un claro problema de clasificación de imágenes multiclas.
- Escoger el entorno de trabajo para la implementación de la solución. Las posibilidades son amplias (distintos lenguajes de programación, *on Cloud* u *on Premise*, etc.) pero, tras evaluarlas se ha seleccionado una que permite llevar a cabo la implementación de la solución de manera cómoda y efectiva.

- Cargar e inspeccionar los datos de trabajo. En el entorno elegido se ha llevado a cabo una carga e inspección del conjunto de datos seleccionado con el objetivo de ver si, efectivamente, son válidos para la tarea a desarrollar.
- Construir y entrenar varios modelos. El *Deep Learning* es una tecnología empírica, por lo que fue necesario crear y entrenar varios modelos antes de obtener modelos prometedores.
- Evaluar los modelos. Los modelos obtenidos se evaluaron en función de la precisión obtenida al trabajar con los datos de prueba. La precisión con datos de prueba mínima fijada para considerar a los modelos construidos válidos ha sido del 90 %.

## 1.2. Estructura del Trabajo

La estructura del trabajo está dividida en seis capítulos y un apéndice. A continuación, se pasa a describir el contenido de cada capítulo y del apéndice:

1. **Introducción:** En este capítulo se presenta el contexto y objeto del trabajo, así como las herramientas utilizadas y los resultados obtenidos. También se definirán los objetivos a cumplir con el desarrollo del trabajo y la estructura del mismo.
2. **Una aproximación a la agricultura inteligente:** En este capítulo se presenta el término de *smart farming* y se definen y desglosan las tecnologías implicadas, empezando por la recopilación de datos hasta el uso de técnicas de Inteligencia Artificial. Finalmente, se revisan trabajos previos relacionados con *smart agriculture*.
3. **Metodología:** En este capítulo se detalla la metodología utilizada para llevar a cabo el desarrollo del trabajo, definiendo roles y conceptos y relacionándolos con sus homólogos de otras metodologías conocidas.
4. **Sistema de monitoreo de cultivos:** En este capítulo se aborda la definición de requisitos e implementación de las soluciones que componen el total del trabajo, describiendo las investigaciones llevadas a cabo, las dificultades encontradas y las decisiones tomadas.
5. **Evaluación:** En este capítulo se evalúan las soluciones construidas durante el capítulo 4, contrastando los resultados obtenidos frente a los requisitos definidos.
6. **Conclusiones y trabajos futuros:** En este último capítulo, se resume el trabajo realizado y se identifican áreas de mejora y posibles trabajos futuros para las soluciones implementadas.

7. **Apéndice A: Repositorio del trabajo:** En este apéndice se proporciona información adicional acerca del repositorio GitHub en el cual está publicado el código fuente de las soluciones implementadas durante este trabajo.

## 2. Una aproximación a la agricultura inteligente

A lo largo de la historia, la raza humana se ha encontrado dificultades las cuales ha tenido que superar. La pasada pandemia del COVID-19 es un ejemplo de una de estas dificultades. Sin embargo, muchas de esas dificultades siguen presentando desafíos a día de hoy y los presentarán también en un futuro. Es el caso del calentamiento global el cual, ya desde hace unos años, se está intentando reducir mediante una acción global coordinada.

La Figura 2.1<sup>1</sup> muestra como la ONU estima que la población mundial alcanzará los 9700 millones para el año 2050 (United Nations Department of Economic and Social Affairs, Population Division, 2022). Estas cifras de población supondrán una demanda de recursos básicos como, agua dulce y comida, sin precedentes. En concreto, se estima que la demanda de comida o productos agrícolas subirá un 50% para el año 2050 (Food and Agriculture Organization of the United Nations, 2017). Tomando en cuenta estos datos, la superpoblación y la seguridad alimentaria se alzan como unas de esas grandes dificultades globales a las que la humanidad tendrá que hacer frente en un futuro no muy lejano.

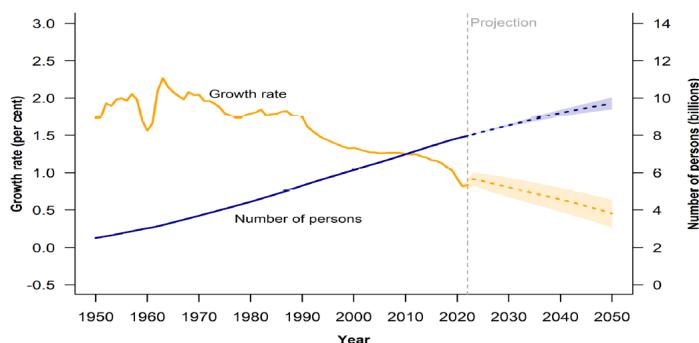


Figura 2.1: Estimación de población global de la ONU para 2050

Con el motivo de mitigar estas amenazas, el sector agrícola se ha visto obligado a iniciar una búsqueda de soluciones en el desarrollo de la ciencia y la tecnología, que propicie una nueva evolución del sector que permita incrementar la productividad y eficiencia actuales (Araújo et al., 2021).

<sup>1</sup>Gráfico sacado de (United Nations Department of Economic and Social Affairs, Population Division, 2022)

## 2.1. El paisaje tecnológico de la Industria 4.0

### La era de los datos

La Unión Internacional de Telecomunicaciones (UIT<sup>2</sup>) define *IoT* como “una infraestructura mundial para la sociedad de la información, que propicia la prestación de servicios avanzados mediante la interconexión de objetos (físicos y virtuales) gracias a la interoperatividad de tecnologías de la información y la comunicación presentes y futuras” (Sector de normalización de las telecomunicaciones de la UIT, 2012).

Básicamente, cuando se habla de *IoT*, se hace referencia a un red de objetos físicos con capacidad de conexión a internet. Dichos objetos no son únicamente ordenadores, tabletas o *smartphones* sino que, bien podrían ser vehículos, electrodomésticos o *wearables*<sup>3</sup>, entre otros. Todos estos dispositivos, coleccionan datos y, además, se comunican y comparten información entre si. El *IoT* afecta prácticamente a todas las áreas de la vida cotidiana y permite la creación de entornos inteligentes en múltiples dominios como transporte, salud o agricultura (Patel et al., 2016).

El término *Big Data* se utiliza para describir conjuntos de datos extremadamente grandes con una estructura muy diversa y compleja, lo que hace que sea complicado almacenarlos, analizarlos y visualizarlos para llevar a cabo procesos adicionales o obtener resultados. Dichos procesamiento o búsqueda de resultados en forma de patrones o conexiones que se mantienen ocultas a simple vista se denomina análisis de datos masivos o *Big Data Analytics* (Sagiroglu and Sinanc, 2013).

En este contexto, aparece *MapReduce*. Diseñado por Google y, con el mismo enfoque que la popular técnica de resolución de problemas *Divide and Conquer* (Farzan and Nicolet, 2021), *MapReduce* se utiliza en el ámbito de *Big Data Analytics* para descomponer análisis de grandes conjuntos de datos en tareas más pequeñas y procesarlas de manera paralela (Sagiroglu and Sinanc, 2013).

En un algoritmo *MapReduce*, el proceso se divide en rondas, y cada ronda tiene tres etapas: *map*, *shuffle* y *reduce*. Aunque sea un trabajo en paralelo, a continuación se describe el flujo de *MapReduce* desde el punto de vista de una máquina específica llamada *M* (Tao et al., 2013). Adicionalmente, en la Figura 2.2, se muestra de manera visual el funcionamiento de *MapReduce*.

- **Map:** *M* toma sus datos locales y crea una lista de pares clave-valor (*k*, *v*) mediante la aplicación de una función *f*. Estos pares se preparan para ser enviados a otra máquina en la fase de reorganización, y la elección de la máquina de destino se basa en la clave *k* (Tao et al., 2013).

<sup>2</sup><https://www.itu.int/es/about/Pages/default.aspx>

<sup>3</sup><https://www.universidadviu.com/es/actualidad/nuestros-expertos/que-es-wearable-y-que-tipos-de-dispositivos-existen>

- **Combine:** Fase opcional que consiste en un pre-agrupamiento de los pares clave-valor locales de todas las máquinas para minimizar el coste de transferencia entre máquinas de la siguiente fase (Lee et al., 2012).
- **Shuffle:** Habiendo, cada máquina, creado su propia lista clave-valor, se distribuyen todas estas listas teniendo en cuenta una regla importante: los pares con la misma clave deben ser entregados a la misma máquina.
- **Reduce:**  $M$  toma los pares clave-valor que ha recibido en la fase de reorganización y aplica una función  $p$ . El resultado es la salida final local de  $M$  y puede utilizarse a su vez como entrada para otro *MapReduce* (Tao et al., 2013).
- **Salida final:**  $M$  tiene su salida local en un fichero al igual que las otras máquinas. Sin embargo, la salida final sería el conjunto de las salidas locales de cada máquina (Dean and Ghemawat, 2008).

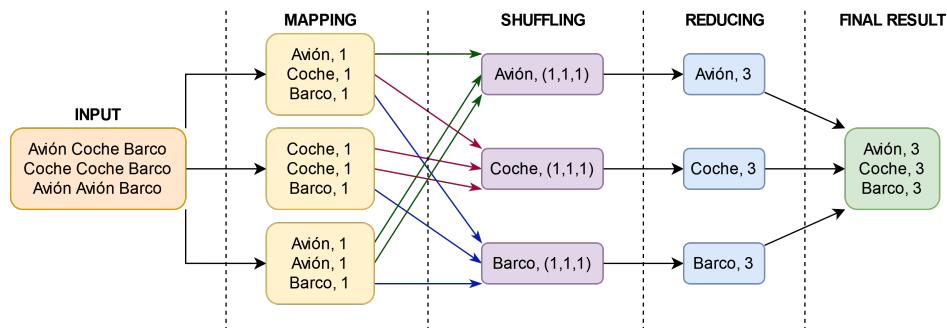


Figura 2.2: Ejemplo funcionamiento MapReduce

Llegados a este punto y teniendo en mente una sociedad en la que se generan datos a diario como la actual, la mayoría de las empresas, sin importar su sector, están recopilando, almacenando y aplicando *Big Data Analytics* con el fin de obtener valor, generar predicciones más precisas y, en última instancia, tomar decisiones más informadas (Vassakis et al., 2018).

## Un recorrido desde la Inteligencia Artificial hasta las Redes Neuronales Convolucionales

La Inteligencia Artificial (IA) está transformando múltiples ámbitos de la sociedad. La IA podría definirse como la capacidad de las máquinas para realizar tareas que normalmente requerirían inteligencia humana como percibir el entorno, tomar decisiones o aprender de la experiencia. Reconocimiento del habla, competir en juegos estratégicos, como el ajedrez, vehículos autónomos o análisis de datos complejos son algunas de las aplicaciones más conocidas de la IA (Ongsulee, 2017).

El término *Machine Learning (ML)* o Aprendizaje Automático fue acuñado en 1959 por Arthur Samuel cuando trataba de enseñar a una máquina a jugar a las damas. Se refiere a programas informáticos que pueden aprender a realizar acciones para las cuales no se les ha programado de forma explícita. Dicho aprendizaje viene dado por tres factores: datos consumidos por el programa, una métrica de error entre el comportamiento actual y el comportamiento ideal y un mecanismo de retroalimentación que utiliza el error cuantificado para mejorar el comportamiento en eventos futuros (Joshi, 2020).

El *ML* es concebido como un rama específica de la IA y se pueden distinguir tres principales tipos o paradigmas dentro de ella (Aggarwal et al., 2022):

- **Aprendizaje Supervisado:** Se trabaja con datos etiquetados donde cada observación del conjunto de datos tiene asignado un resultado específico. El objetivo es realizar una predicción. Si las predicciones son valores discretos, como una decisión de compra, se identifica un problema de clasificación. Si los valores son continuos, como el precio de un producto, se trata de un problema de regresión (Aggarwal et al., 2022).
- **Aprendizaje No Supervisado:** Se buscan patrones ocultos en los datos sin la necesidad de etiquetas previas por lo que el conjunto de datos no necesita estar etiquetado. El objetivo es clasificar o agrupar datos en función de similitudes y estructuras no evidentes a simple vista sin necesidad de conocer de antemano las categorías específicas (Aggarwal et al., 2022).
- **Aprendizaje por Refuerzo:** El objetivo es desarrollar un sistema cuyo aprendizaje está basado en las interacciones que el mismo hace con su entorno. Lo normal es que en este tipo de sistemas deban de tomar decisiones en un entorno específico para maximizar una recompensa acumulativa (Mahesh, 2019).

Entre los campos de aplicación del *ML* se encuentran la visión por computador, el análisis semántico, predicción, procesamiento de lenguaje natural y recuperación de información (Shinde and Shah, 2018).

Las redes neuronales artificiales, *ANN* por sus siglas en inglés, son, básicamente, una tecnología basada en la estructura de las redes de neuronas que están presentes en el cerebro humano. Al igual que en el cerebro, una *ANN* consta de nodos interconectados entre sí y organizados en capas que hacen las veces de neuronas. Estas neuronas artificiales toman entradas de otros elementos o nodos y, después de aplicar ponderaciones y sumar las entradas, el resultado se procesa mediante una función de transformación para obtener la salida (Sharma et al., 2012).

El *Deep Learning (DL)* o Aprendizaje Profundo, es una rama del Aprendizaje Automático que permite que las máquinas puedan extraer, analizar y comprender, de manera automática, información valiosa a partir de datos sin procesar. Para ello, el

*DL* se vale de modelos no lineales cuya arquitectura está basada en redes neuronales profundas y que permiten al sistema aprender las relaciones complejas entre los datos de entrada y los resultados. La ventaja principal del Aprendizaje Profundo sobre el Aprendizaje Automático convencional es que no requiere la extracción manual o predefinida de características de los datos, ya que automáticamente identifica y utiliza las características más relevantes (Chauhan and Singh, 2018).

Puesto que el *DL* es una rama del *ML* que hace uso de modelos de redes neuronales profundas, los ámbitos de aplicación del *DL* coinciden con los del *ML*. Así pues, la visión por computador, el análisis semántico, predicción, procesamiento de lenguaje natural y recuperación de información son también áreas en de aplicación del Aprendizaje Profundo (Shinde and Shah, 2018).

A lo largo del desarrollo y popularización del las *ANN* y el *DL*, han aparecido diferentes arquitecturas neuronales que, aunque pueden utilizarse para cualquier tarea algunas son más adecuadas para ciertos tipos de datos. Las diferencias entre estas estructuras se encuentran, principalmente, en su composición, es decir, en los tipos de capas, nodos neuronales y conexiones que emplean. Algunos ejemplos de arquitecturas neuronales populares son: Redes Neuronales Convolucionales (*CNN*), Redes Generativas Adversariales (*GANs*), Redes Neuronales Recurrentes (*RNNs*) o Codificadores Automáticos (*Autoencoders*) (Leijnen and van Veen, 2020).

Las *CNN* son un tipo de arquitectura neuronal que permite incorporar características específicas de las imágenes a la estructura de la red lo que la hace más adecuada para tareas en la que los datos de entrada son imágenes. Esta arquitectura suele formarse combinando tres tipos de capas neuronales: Capas Convolucionales, Capas de Agrupación y Capas Densas (O'Shea and Nash, 2015).

Normalmente, se apilan dos Capas Convolucionales seguidas de una Capa de Agrupación lo que constituye un Bloque Convolutional. Un modelo *CNN* suele estar compuesto por uno o más Bloques Convolucionales. Finalmente y, tras aplicar un *Flatten* o aplanado, se aplican una o varias Capas Densas las cuales utilizan la información proveniente de las Capas Convolucionales para tomar decisiones como, por ejemplo, determinar si la imagen es de un gato o un perro (O'Shea and Nash, 2015).

En la Figura 2.3<sup>4</sup> se muestra un ejemplo de arquitectura de una *CNN* de dos Bloques Convolucionales constituidos, cada uno de ellos, por dos Capas Convolucionales y una Capa de Agrupación. Seguidamente a los Bloques Convolucionales y, previo aplanado, se añaden Capas Densas. El problema que se supone que se resuelve con esta *CNN* de ejemplo, es un problema de clasificación de imágenes, siendo capaz de discernir si la imagen entrante se trata de un hombre o una mujer.

<sup>4</sup>La imagen de entrada de la Figura 2.3 ha sido generada en <https://this-person-does-not-exist.com/es>

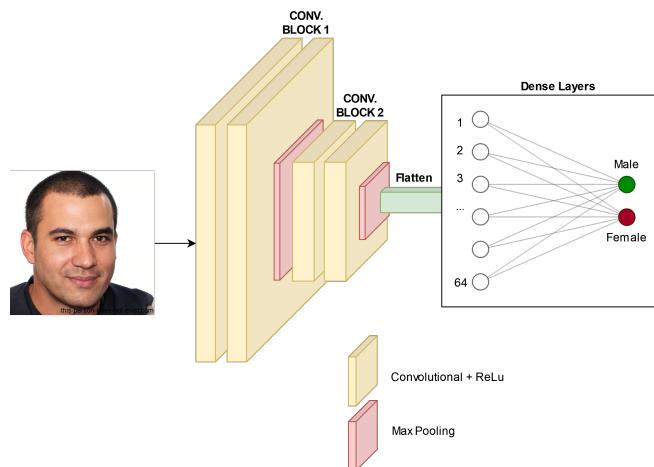


Figura 2.3: Ejemplo arquitectura CNN

Las aplicaciones de las *CNN* están, obviamente, fuertemente ligadas a las imágenes. Clasificación de imágenes, detección de objetos, segmentación de imágenes y reconocimiento facial son algunas de las principales áreas de aplicación de las *CNN*.

Habiendo definido todos los conceptos, en la Figura 2.4 se muestra un muy breve resumen del camino que se ha seguido y hasta qué punto se ha llegado.



Figura 2.4: Esquema visual de términos definidos

## 2.2. Agricultura 4.0 o *Smart Agriculture*

Tal y como se comentó al inicio de la Sección 2, ante el continuo crecimiento de la demanda de alimentos, el sector agrícola se ha visto en la obligación de buscar en las nuevas tecnologías las soluciones que le permitan aumentar la productividad y eficiencia para poder absorber la creciente demanda (Araújo et al., 2021).

La adopción de las tecnologías definidas y detalladas en la Sección 2.1 por parte de la industria agrícola da como resultado la aparición de términos como agricultura inteligente, *smart farming*, *smart agriculture* o agricultura 4.0. Al fin y al cabo, todos estos términos hacen referencia a lo mismo: la adopción del sector agrícola de tecnologías como el *IoT*, el *Big Data* o *ML*, entre otras, con el objetivo de revolucionar la producción de productos agrícolas en términos de eficiencia, productividad y sostenibilidad (Moysiadis et al., 2021).

En el contexto agrícola, el *IoT* se refiere al uso de sensores y otros dispositivos para recopilar datos sobre todos los elementos y acciones involucrados en la actividad agrícola. De hecho, el uso del *IoT* permite el uso de otros avances tecnológicos como el *Big Data* debido a que genera y recopila gran cantidad de información valiosa que puede ser analizada (Quy et al., 2022).

Como aplicaciones que el *IoT* aporta directamente al sector agrícola se pueden mencionar la monitorización o el rastreo y seguimiento de alimentos. La agricultura inteligente se sirve del *IoT* para monitorizar factores clave que afectan al proceso de cultivo y producción de alimentos. En la agricultura de cultivos, la monitorización de características como humedades de suelo y aire, radiación solar o nutrientes del suelo, puede proporcionar a los agricultores información en tiempo real del estado de sus cultivos (Quy et al., 2022). Adicionalmente, Wang et al. (2021) proponen un sistema de trazabilidad de alimentos mediante la integración de *IoT*, *RFID*<sup>5</sup> y *block-chain*.

Como consecuencia del uso de dispositivos *IoT* para la monitorización de factores clave, la agricultura inteligente genera enormes cantidades de datos. Es aquí donde entra en juego el *Big Data* y, más concretamente, el *Big Data Analytics*. El análisis de la gran cantidad información que es recopilada por los sensores *IoT*, puede revelar conocimiento que hasta ahora habría permanecido oculto y facilitaría la toma de decisiones a un nivel sin precedentes ya que dichas decisiones estarían influenciadas únicamente por datos (Wolfert et al., 2017). Mediante el procesamiento y el análisis de datos masivos, el sector agrícola puede obtener datos cruciales para la toma de decisiones de los agricultores como datos del clima, del suelo o del comportamiento animal (Cravero et al., 2022).

<sup>5</sup>Identificación por radiofrecuencia: tecnología cuyo objetivo es transmitir la identidad de un objeto mediante ondas de radio. Muy usada en *IoT*

De igual manera que el *Big Data Analytics* se beneficia del uso del *IoT* en *smart farming*, el *ML* también se ve beneficiado. De nuevo gracias a la enorme cantidad de datos recopilada por los dispositivos *IoT*, los algoritmos de *ML* tienen más y mejores datos de los que poder aprender. Algunas de los casos de uso derivados del uso del *ML* en el sector agrícola son: diagnóstico de enfermedades en los cultivos, identificación de tipos de cultivos, gestión eficiente del riego, evaluación de la calidad del suelo, predicción del clima o análisis del mercado (Sinha and Dhanalakshmi, 2022).

La industria agrícola también se está viendo increíblemente beneficiada por la adopción del *DL*. Un claro ejemplo es la detección de enfermedades de cultivos a través de imágenes. El uso de drones en el sector agrícola es cada vez más frecuente. Los drones van recopilando imágenes de los cultivos de las explotaciones que se usan para, mediante modelos de *DL* basados en *CNN*, detectar si el cultivo padece alguna enfermedad. Otro caso de uso útil del *DL* en *smart agriculture* es la clasificación del terreno. A través de imágenes satelitales, se usa *DL* y, de nuevo, *CNN* para clasificar el terreno en base a diferentes características, por ejemplo, productividad de la tierra (Ünal, 2020).

Por último, el *Cloud Computing* también juega un papel crucial en la agricultura inteligente. Normalmente las explotaciones agrícolas suelen estar en zonas rurales donde es difícil montar la infraestructura necesaria para ejecutar los sistemas descritos previamente. El *Cloud Computing* proporciona los servicios de almacenamiento, sistemas de cómputo y seguridad informática que los casos de uso anteriores requieren a un coste asequible para los agricultores (Araújo et al., 2021).

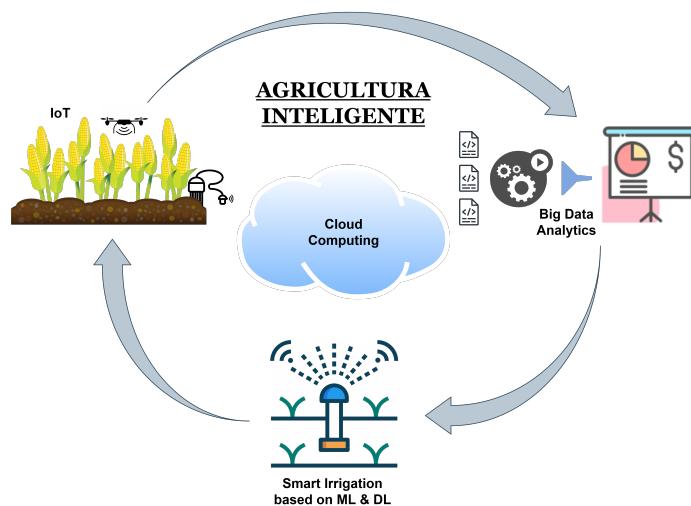


Figura 2.5: Esquema visual de smart farming

## 2.3. Trabajos relacionados con la agricultura inteligente

Existen numerosos trabajos que proponen la utilización de tecnologías como el *IoT*, *Big Data*, *Machine Learning* y *Deep Learning* en el sector agrícola. Adicionalmente, también hay algunos proyectos que ya han integrado este tipo de tecnologías en soluciones para el sector agrícola. A continuación, se describen algunos artículos en los que se analiza el uso de las tecnologías mencionadas anteriormente en la industria agrícola, no solo no es descabellado sino que, además, es beneficioso.

Almalki et al. (2021) integran *IoT* con Vehículos Aéreos No Tripulados (*UAVs* por sus siglas en inglés), para presentar una plataforma de bajo costo para el monitoreo integral de parámetros ambientales utilizando *IoT* volador. Estos parámetros ambientales se recopilan y se envían a la nube para analizarlos. La plataforma se ha probado en un escenario real en una granja en Medenine, Túnez desde marzo de 2020 a marzo de 2021. Los resultados prácticos obtenidos indican que se han aplicado o sugerido acciones automatizadas y humanas que llevan al aumento significativo de la productividad de los cultivos y contribuye al ahorro de recursos naturales.

Channe et al. (2015) proponen una plataforma que, mediante el uso de *IoT*, *Cloud Computing* y *Big Data Analysis*, recopila y analiza cantidades masivas de datos de agricultores, calidad de suelo, condiciones climáticas, vendedores de productos agrícolas, agromarketing e incluso programas del gobierno para el sector agrícola. A través de *Big Data Analysis* sobre la información anterior, AgroCloud es capaz de sacar conclusiones como fertilizantes recomendados, secuencias de cultivos recomendados o requisitos de cumplimiento recomendado para el mercado actual. Los autores concluyen apuntando que la plataforma es beneficiosa no solo para aumentar la producción agrícola sino que también lo es para controlar los costos de los productos agrícolas.

Alfred et al. (2021) hacen una revisión de cómo tecnologías como el *IoT*, *Big Data* y *ML* se podrían aplicar a la industria de la producción de arroz analizando aplicaciones como irrigación inteligente para el arroz, la estimación de rendimiento del arroz, el monitoreo del crecimiento del arroz, el monitoreo de enfermedades en el arroz, la evaluación de la calidad del arroz y la clasificación de muestras de arroz. La conclusión a la que llegan los autores es que una integración eficiente y efectiva de estas tres tecnologías es fundamental para transformar las prácticas tradicionales del cultivo del arroz en términos de productividad y eficiencia.

Las hojas de los cultivos son un elemento clave a la hora de valorar la salud, el entorno y la madurez de los cultivos. Kumar et al. (2019) presentan una solución llamada gCrop que, mediante *IoT*, procesamiento de imágenes y *ML*, monitoriza en tiempo real el crecimiento y desarrollo de los cultivos con hojas. gCrop identifica las hojas de los cultivos mediante una cámara inteligente, calcula sus dimensiones y las

correlaciona con las condiciones ideales para la edad y madurez de la especie. El modelo muestra una precisión cercana al 98 % al predecir el crecimiento de las hojas lo que aporta muchísima seguridad a la hora de comparar las condiciones reales con las condiciones ideales. Se espera que este sistema contribuya de manera efectiva a fortalecer las prácticas agrícolas actuales garantizando la calidad de los cultivos y mejorando el rendimiento de la producción

Para Mohanty et al. (2016), de nuevo, las hojas de los cultivos juegan un papel crucial. Los autores han sido capaces de construir un modelo basado en *CNN* capaz de identificar 14 especies de cultivos y 26 enfermedades. Utilizando un conjunto de datos público de 54306 imágenes de hojas de plantas enfermas y sanas y aplicando *DL* han conseguido entrenar un modelo que logra una precisión de más del 99 % con un conjunto de datos de prueba independiente. Este enfoque demuestra que el uso de *DL* y, más concretamente, *CNN* es un claro camino hacia el diagnóstico de enfermedades de cultivos asistido o, incluso, automático.

Como se puede comprobar gracias a los trabajos anteriormente mencionados, la agricultura inteligente es un enfoque más actual que futurista, ya que actualmente existen muchos proyectos que abrazan dicho enfoque. A ello han ayudado enormemente el desarrollo de las tecnologías mencionadas a lo largo de este capítulo (*IoT*, *Big Data*, *ML*, *ANNs*, *DL* y *CNN*), a su vez, propiciado por el aumento de capacidad computacional alcanzado gracias a los avances en *Hardware*.

Tanto es así que está empezando a aparecer el término de Industria 5.0 (Agricultura 5.0 si se aplica al sector agrícola) dejando el término Agricultura 4.0 para los sistemas digitales basados en datos que aumentan el conocimiento de los productores sobre sus campos. Cuando estas explotaciones agrícolas basadas en datos incorporan la robótica con algoritmos de inteligencia artificial a sus sistemas se le denominaría Agricultura 5.0 (Saiz-Rubio and Rovira-Más, 2020).

### 3. Metodología

Para llevar a cabo la implementación de las soluciones de este trabajo, se ha utilizado una metodología *adhoc*, es decir, una metodología a medida. Tal y como se ha detallado en el Apartado 1.1.2, el objetivo de este TFM es implementar dos soluciones distintas. El enfoque del que se ha partido y, por tanto, se ha tomado como base para la metodología *adhoc*, ha sido un enfoque iterativo e incremental como es Scrum<sup>1</sup>.

En primer lugar, la metodología distingue dos grandes fases bien diferenciadas (una por cada solución a desarrollar). La primera fase se corresponde con el desarrollo del procesamiento de datos en tiempo real, mientras que en la segunda fase se aborda la implementación de la solución de detección de enfermedades de cultivos a partir de imágenes. Cada una de estas fases tiene sus propios requisitos o funcionalidades a implementar, es decir, tiene su equivalente al *Product Backlog*<sup>2</sup> que existe en Scrum.

Dentro de cada fase, se distinguen iteraciones o, si se habla en argot de Scrum, *sprints*<sup>3</sup>. Cada iteración cuenta con su propio subconjunto de funcionalidades o requisitos del *Product Backlog* a desarrollar, es decir, lo que en Scrum se llama *Sprint Backlog*<sup>4</sup>.

Durante el desarrollo del trabajo, cada semana tuvo lugar una reunión con la tutora, Judith Cardinale en el caso de este TFM en concreto. En dichas reuniones se discutieron los avances de la semana, dudas, obstáculos y tareas pendientes. En este contexto, Judith tuvo el rol de Scrum conocido como *Product Owner*<sup>5</sup> mientras que, José Ramón Moratalla, el estudiante, tuvo el rol de equipo de desarrollo<sup>6</sup>.

Como se puede comprobar, la metodología personalizada que se utilizó es, básicamente, una adaptación de Scrum a las circunstancias de este TFM. Si bien se han cambiando algunas cosas, la mayoría de terminología y funcionamiento de Scrum se mantienen.

---

<sup>1</sup><https://www.scrum.org/learning-series/what-is-scrum>

<sup>2</sup><https://www.scrum.org/learning-series/what-is-scrum/the-scrum-artifacts/what-is-a-product-backlog>

<sup>3</sup><https://www.scrum.org/learning-series/what-is-scrum/the-scrum-events/what-is-a-sprint>

<sup>4</sup><https://www.scrum.org/learning-series/what-is-scrum/the-scrum-artifacts/what-is-a-sprint-backlog>

<sup>5</sup><https://www.scrum.org/learning-series/what-is-scrum/the-scrum-team/what-is-a-product-owner>

<sup>6</sup><https://www.scrum.org/learning-series/what-is-scrum/the-scrum-team/what-is-a-developer>

A continuación, en la Figura 3.1, se muestra un esquema que detalla los componentes y el funcionamiento de la metodología *adhoc* abordada.

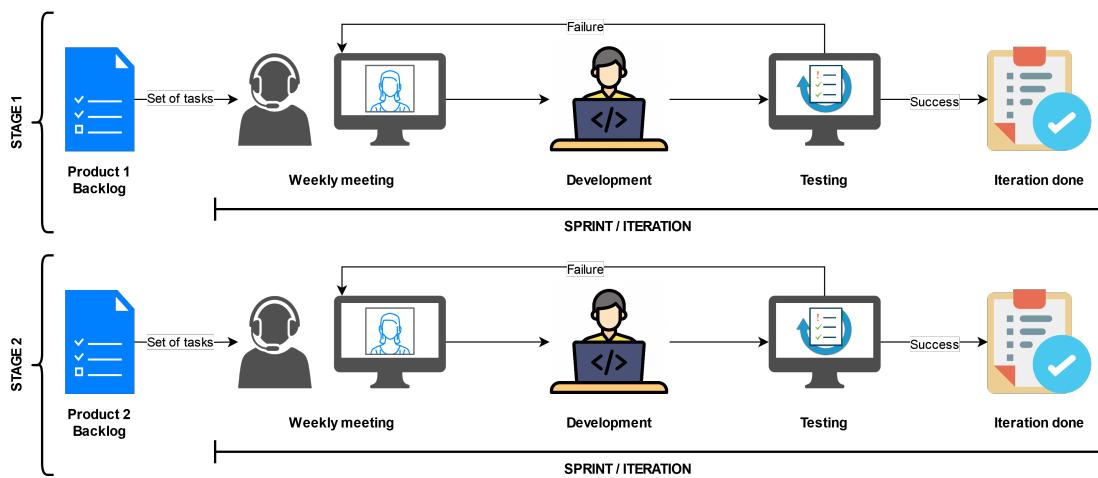


Figura 3.1: Esquema de la metodología adhoc abordada

## 4. Sistema de monitoreo de cultivos

Recordando lo comentado en el Apartado 1.1.1, el objetivo general del trabajo es implementar un prototipo de sistema que suponga una aproximación a la agricultura inteligente mediante el uso e integración de tecnologías como el Internet de las Cosas, *Big Data*, Aprendizaje Profundo y Redes Neuronales Convolucionales.

Adicionalmente, en la Sección 1.1.2 se definen dos áreas del *smart farming* concretas sobre las que se centraron los esfuerzos. Dichas áreas son: procesamiento de datos en tiempo real y detección de enfermedades de cultivos a partir de imágenes.

La Figura 4.1 muestra de manera esquemática lo se quiere conseguir.

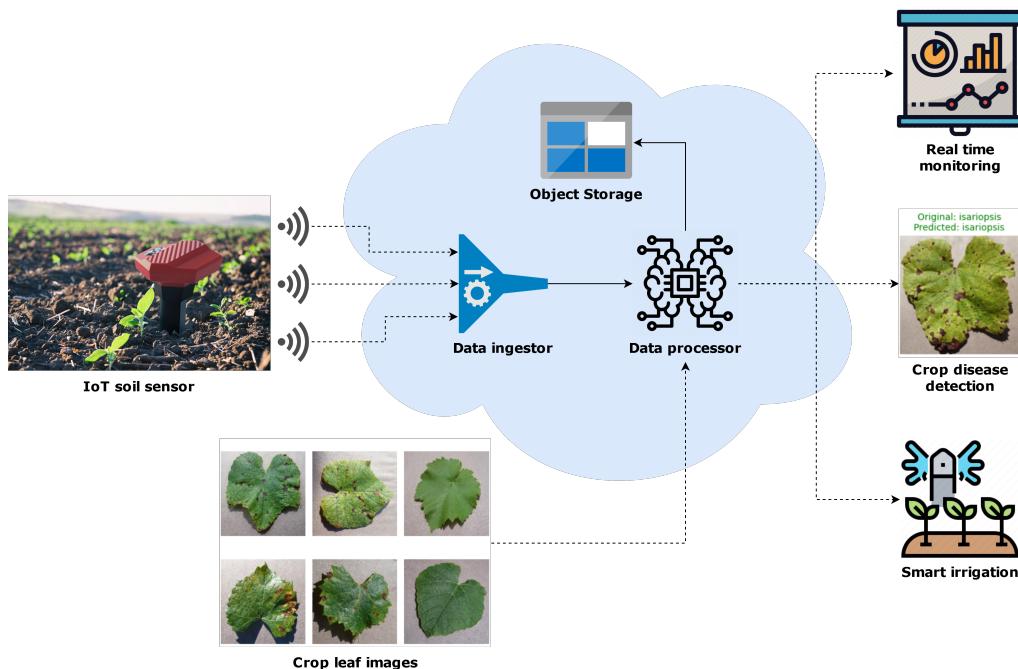


Figura 4.1: Boceto de solución deseada

A partir de los sensores que instrumentan una plataforma IoT para el monitoreo de cultivos, se obtienen los datos que pueden ser procesados en tiempo real con la intervención de un ingestor de datos y un motor de procesamiento en *streaming*; el resultado del procesamiento puede ser usado para distintas aplicaciones, tanto en lotes (por lo que requieren ser almacenados para posterior análisis) o en tiempo real, como visualización de datos en tiempo real, control de riego inteligente o detección de enfermedades de los cultivos.

Como se mencionó en el Capítulo 3, la construcción de las soluciones se aborda en dos fases diferenciadas, una para cada solución implementada. Cada fase, a su vez, está dividida en iteraciones o *sprints*. En las siguientes secciones se describen en detalle ambas fases.

## 4.1. Fase 1: Procesamiento de datos en *streaming*

A continuación se listan los requisitos o funcionalidades que la solución ha de cumplir, es decir, se define el *Product Backlog*:

- **REQ1:** Los datos que se han de utilizar deben de tener que ver con calidad de suelo. Este requisito viene dado por el proyecto de investigación en el que se ve envuelto este TFM.
- **REQ2:** La salida tras el procesamiento ha de ser en formato CSV.
- **REQ3:** Obviamente, el procesamiento debe de ser en *streaming* o tiempo real.

### Iteración 1: Elección del ingestor de datos

Antes siquiera de empezar a valorar las herramientas disponibles para llevar a cabo la ingesta de datos, es necesario definir cómo se va a hacer la ingesta de los datos. Lo ideal, para ser fieles a los casos de uso que se dan en la agricultura inteligente, es consumir los datos de sensores de suelo IoT. Sin embargo, no se dispone de este tipo de dispositivos durante el desarrollo del TFM. Como solución alternativa, se模拟aron estos sensores IoT mediante *scripts Python* que van generando datos sintéticos a partir de información disponible en repositorios públicos captada por sensores reales.

Otro de los aspectos a definir es dónde se va a alojar el sistema de procesamiento. ¿Va a ser un sistema alojado *on Cloud* o será un sistema *on Premise*<sup>1</sup>? De nuevo, para ir en consonancia con los casos de uso que se dan en *smart farming*, lo correcto sería que fuese el sistema estuviese alojado en la nube. No obstante, puesto que el objetivo es obtener un prototipo que suponga una aproximación a la agricultura inteligente, se empieza a trabajar en local y, por restricciones de tiempo, se plantea como trabajo futuro subir la solución a cualquier proveedor *Cloud*.

Habiendo establecido el punto de partida del sistema de procesamiento en *streaming*, se procede, ahora sí, a analizar las herramientas de ingesta de datos disponibles en el mercado. Tras la lectura de varios artículos, entre los que destacaron Sharma et al. (2021); Mătăcuță and Popa (2018); y Alwidian et al. (2020), las opciones para una ingesta de datos para un procesamiento en *streaming on Premise*

---

<sup>1</sup>Alojado *in situ*, es decir, en las instalaciones de la empresa.

quedan reducidas a: Apache Flume<sup>2</sup>, Apache NiFi<sup>3</sup> y Apache Kafka<sup>4</sup>.

## Apache Flume

En la web oficial de Apache Flume se encuentra la siguiente definición: “Flume es un servicio distribuido, confiable y disponible diseñado para recopilar, agregar y trasladar de manera eficiente grandes cantidades de datos de registro. Cuenta con una arquitectura sencilla y flexible basada en flujos de datos en tiempo real. Es robusto y tolerante a fallos, con mecanismos de confiabilidad ajustables y numerosos mecanismos de comutación por error y recuperación.”

Flume tiene como objetivo principal la recopilación eficiente de grandes cantidades de datos desde diversas fuentes para su posterior almacenamiento en el Sistema de Archivos Distribuido Hadoop (HDFS<sup>5</sup>) o similares (Mătăcuță and Popa, 2018). En este contexto, podría pensarse que al guardarlos en HDFS o similares, Flume está más enfocado al procesamiento en *batch*, sin embargo, tal y como se expone más adelante, es posible que herramientas de procesamiento en *streaming* tomen HDFS como fuente.

Un caso de uso específico de Flume es la industria manufacturera, donde es muy común generar y almacenar un registro en cada fabricación de un producto y se almacena en un archivo. En un solo día, se fabrican miles de productos por lo que los archivos de registro suponen una gran cantidad de datos. Flume se utiliza para transmitir estos datos a una herramienta de procesamiento en tiempo real alimentada por el almacenamiento de HDFS (Mătăcuță and Popa, 2018).

## Apache NiFi

La descripción de Apache NiFi que se encuentra en su web oficial es: “Un sistema fácil de usar, potente y confiable para procesar y distribuir datos”. Adicionalmente, se destacan varias características como: interfaz gráfica de control y monitorización, capacidad de seguimiento de los datos, comunicación segura gracias a protocolos como TLS o HTTPS, tolerancia a fallos y alto rendimiento.

Apache NiFi automatiza el movimiento de datos entre diversas fuentes de datos y sistemas de almacenamiento de datos, lo que facilita la ingestión de datos de manera rápida, sencilla y segura. Admite tanto el procesamiento por lotes como el procesamiento de datos en tiempo real. NiFi es capaz traspasar datos entre archivos en disco, sistema de archivos HDFS, colas de mensajes Kafka y bases de datos SQL y NoSQL (Sharma et al., 2021).

---

<sup>2</sup><https://flume.apache.org/>

<sup>3</sup><https://nifi.apache.org/>

<sup>4</sup><https://kafka.apache.org/>

<sup>5</sup>[https://hadoop.apache.org/docs/r1.2.1/hdfs\\_design.html](https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html)

Algunos sistemas generan los datos y otros sistemas los consumen. Apache NiFi está desarrollado para automatizar este flujo. Mătăcuță and Popa (2018) consideran a NiFi una buena alternativa a Flume y destacan su capacidad de personalización, su interfaz de usuario fácil de usar y su facilidad de configuración.

### Apache Kafka

En la página web oficial de Apache Kafka se encuentra la siguiente descripción: “Apache Kafka es una plataforma de transmisión de eventos distribuida de código abierto utilizada por miles de empresas para la creación de canalizaciones de datos de alto rendimiento, análisis en tiempo real, integración de datos y aplicaciones críticas”.

Kafka es un sistema de mensajería distribuida de alto rendimiento basado en un enfoque de publicación-suscripción. Al ser distribuido proporciona alta disponibilidad y replica los mensajes en todo el clúster antes de almacenarlos en disco (Mătăcuță and Popa, 2018). Kafka mantiene todos los mensajes, independientemente de si son consumidos por algún consumidor o no, durante un período de retención configurable a partir del cual se eliminan automáticamente (Sharma et al., 2021).

Kafka consta de cuatro componentes principales: el broker, los consumidores, los productores y los temas. Los productores envían los mensajes a los temas y los consumidores, que pueden suscribirse a esos temas, leen los mensajes de esos temas (Mătăcuță and Popa, 2018). Kafka se ha utilizado en diferentes investigaciones con éxito. Por ejemplo, se utilizó para estudiar la reacción de las personas ante un terremoto y tsunami en Japón. Kafka se utilizó para la ingestión de tweets como paso previo a un procesamiento de datos con Spark (Alwidian et al., 2020).

### Decisión del ingestor

Habiendo descrito cada una de las tres herramientas principales existentes en el mercado para la ingesta de datos en *streaming*, llega el momento de decidirse por una. El ingestor de datos que se ha elegido es Apache Kafka. Dicha decisión se toma en base a que esta herramienta se ajusta más a las funcionalidades que requiere la solución.

NiFi es bastante prometedora sobre todo porque parece que es bastante fácil de usar y configurar, sin embargo, está más enfocada al traspaso de información entre distintas fuentes de datos que en un sistema de procesamiento. NiFi encajaría perfectamente, por ejemplo, en un proceso ETL<sup>6</sup>, ya que permite realizar esa migración de información y a la vez aplicar transformaciones.

En lo pertinente a Flume, el problema identificado es que depende demasiado de HDFS como salida y, aunque posteriormente las herramientas de procesamiento

<sup>6</sup>Extract, Transform and Load: <https://www.ibm.com/topics/etl>

puedan alimentarse de HDFS, esto limitaría la diversidad de la salida del ingestor. Además, HDFS sería otro componente de la solución a configurar y gestionar.

Por el contrario, Kafka parece ajustarse a la perfección a la funcionalidad que se necesita. Según Mătăcuță and Popa (2018), Kafka y Flume comparten la capacidad de transmitir mensajes, pero tienen objetivos diferentes en su diseño. Flume se enfoca en enviar mensajes a destinos específicos, como HDFS o HBase, mientras que Kafka se concentra en permitir que múltiples aplicaciones consuman sus mensajes. Además, los autores recalcan su rapidez.

La Tabla 4.1 resume lo discutido en este apartado acerca de las distintas herramientas analizadas, mostrando cuál sería el caso de uso ideal para cada una de ellas en base a lo analizado.

Tabla 4.1: Resumen de los ingestores de datos

Ingestor de datos	Caso de uso ideal
Apache Flume	Recopilación de datos y envío a almacenamiento tipo HDFS
Apache NiFi	Traspaso de información entre distintas fuentes de datos
Apache Kafka	Permitir que múltiples aplicaciones consuman sus mensajes

En base a todo lo anterior, se decide seleccionar a Apache Kafka como el ingestor a usar en la solución desarrollada.

## Iteración 2: Elección del procesador de datos

Una vez elegido el ingestor de los datos, es necesario elegir un procesador de datos en tiempo real que se integre bien con el ingestor que se acaba de decidir usar. Al igual que se hizo en la iteración anterior con los ingestores de datos, se llevó a cabo una búsqueda de las opciones disponibles en el mercado. En este caso, las opciones eran más numerosas que en el caso de los ingestores. Aparecieron tecnologías como: Apache Spark<sup>7</sup>, Apache Storm, Apache Heron, Apache Samza y Apache Flink. Sin embargo, había tres herramientas que estaban presentes en todos los artículos revisados.

### Apache Spark

Apache Spark es un motor de procesamiento de código abierto ampliamente utilizado que soporta procesamiento por lotes así como procesamiento en *streaming*. Spark consigue acelerar los tiempos de procesamiento porque se ejecuta en memoria pero también puede realizar el procesamiento en disco cuando los conjuntos de datos son demasiado grandes para ser cargados en memoria (Nasiri et al., 2019).

<sup>7</sup><https://spark.apache.org/>

Spark *Streaming* es una extensión de la API principal de Spark que permite el procesamiento de transmisiones de datos en tiempo real escalable, de alto rendimiento y tolerante a fallos. Los datos pueden ser ingestados desde diversas fuentes como Kafka o Flume. Es necesario remarcar que Spark Streaming se basa en un modo de procesamiento por micro-lotes y no puramente *streaming* (Soumaya et al., 2017).

## Apache Storm

Apache Storm es un motor de procesamiento en tiempo real de código abierto creado por Twitter. Está diseñado para procesar grandes volúmenes de datos, es tolerante a fallos y puede escalar horizontalmente. Un clúster de Storm consta de dos tipos de nodos: el nodo maestro y el nodo trabajador. Dado que Apache Storm no puede gestionar el estado de su clúster, depende de Apache Zookeeper para este propósito (Nasiri et al., 2019).

En la web oficial de Apache Storm se afirma que Storm logra para el procesamiento en tiempo real lo que Hadoop hace para el procesamiento por lotes. También se mencionan algunas aplicaciones ideales para Storm: análisis en tiempo real, aprendizaje automático en línea, cálculo continuo, RPC distribuido o ETL. Además se menciona que una prueba de referencia registró más de un millón de tuplas procesadas por segundo por nodo.

## Apache Flink

Apache Flink es un motor de procesamiento de código abierto capaz de procesar datos en tiempo real y en lote. Puede llegar a una escala de procesamiento de millones de tuplas por segundo y posee mecanismos de tolerancia a fallos. Pese a ser un motor híbrido, el procesamiento en *streaming* es nativo, es decir, no procesa micro lotes. Su arquitectura está basada en un enfoque maestro-esclavo teniendo un Administrador de Trabajos y uno o varios Administradores de Tareas (Nasiri et al., 2019).

El Administrador de Trabajos organiza las tareas y las envía a los trabajadores. También mantiene el estado de todas las ejecuciones y el estado de cada trabajador. Los Administradores de Tareas ejecutan las tareas asignadas por el Administrador de Trabajos pudiendo intercambiar información cuando sea necesario.

## Decisión del motor de procesamiento

En primer lugar, es necesario remarcar la diferencia clara en la forma de procesar los datos entre Apache Spark Streaming y el resto de opciones. Apache Spark Streaming procesa los datos en micro-lotes mientras que Apache Storm y Apache Flink soportan procesamiento *streaming* de manera nativa. Gracias a alojar los datos en memoria, Apache Spark Streaming consigue una latencia tan baja que se

considera un motor de procesamiento en tiempo real.

De hecho, en la Figura 4.2 sacada del trabajo presentado por Lopez et al. (2016) puede verse claramente como Apache Spark Streaming, a pesar de ser capaz de procesar muchísimos mensajes por minuto, se ve, claramente, superado por Apache Storm y Apache Flink. Esta diferencia se debe, principalmente, al procesamiento en micro-lotes utilizado en Spark frente al procesamiento en tiempo real nativo que implementan Storm y Flink.

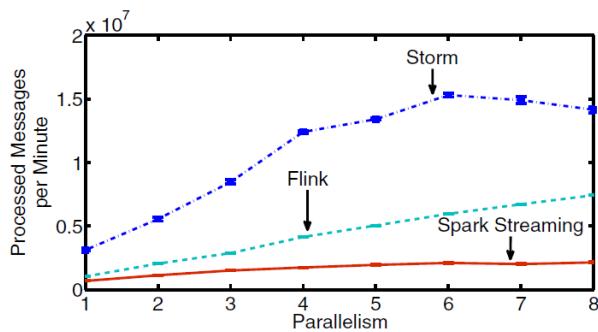


Figura 4.2: [Figura 4 de Lopez et al. \(2016\)](#)

Esta comparación es algo a tener muy en cuenta si la cantidad de mensajes por segundo que se requiere en la solución a desarrollar es muy alta. Sin embargo, en el caso de que la solución no necesite un ratio de envío de mensajes por segundo muy elevado, la importancia de la diferencia mostrada en la Figura 4.2 disminuye. En cualquier caso, el rendimiento en cuanto a mensajes procesados por segundo que ofrecen las herramientas que soportan el procesamiento en *streaming* de manera nativa es, objetivamente, mayor que las que trabajan con un enfoque basado en micro-lotes.

Otra característica importante a tener en cuenta es la capacidad de las herramientas a la hora de recuperarse de la caída de un nodo. La cantidad de mensajes perdidos ante el fallo de un nodo puede ser determinante a la hora de elegir una herramienta u otra. La Figura 4.3 se corresponde con la Figura 5A y 5B de Lopez et al. (2016). En ella puede verse el comportamiento de Storm y Flink ante la caída de un nodo. Como se puede comprobar, Storm tarda mucho menos en volver a su rendimiento normal que Flink.

La Figura 4.4 se corresponde con la Figura 6A y 6B de Lopez et al. (2016). El gráfico de la izquierda de la Figura 4.4 muestra el comportamiento de Spark frente al fallo de un nodo. Como se puede ver, Spark es capaz de mantener el rendimiento sin ningún tipo de degradación de servicio. Por último, el gráfico de la derecha de la Figura 4.4 muestra el porcentaje de mensajes perdidos de cada herramienta durante la caída del nodo, siendo Storm la herramienta que más pierde seguida de Flink. En

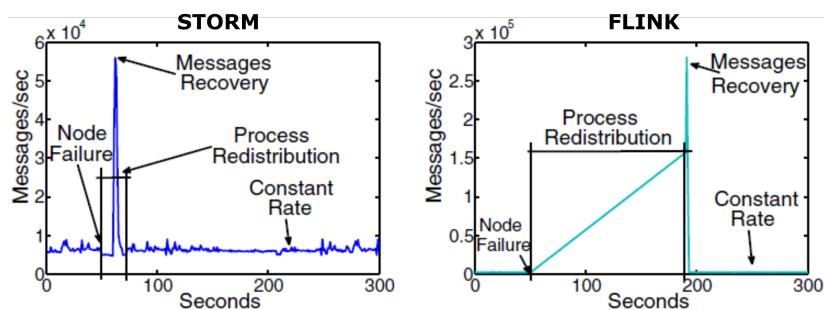


Figura 4.3: Figura 5A (izquierda) y 5B (derecha) de Lopez et al. (2016)

el caso de Spark, puede verse como no ha perdido ni un solo mensaje durante el fallo del nodo.

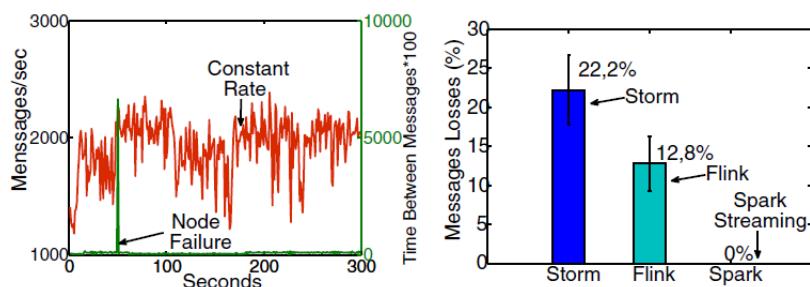


Figura 4.4: Figura 6A (izquierda) y 6B (derecha) de Lopez et al. (2016)

Teniendo en cuenta todo lo anterior, se puede concluir que, a pesar de que Apache Spark Streaming tiene la menor tasa de procesamiento de mensajes por segundo, es una herramienta muy fiable en términos de degradación de rendimiento y pérdida de mensajes en caso de la caída de un nodo.

Tabla 4.2: Resumen de los motores de procesamiento

Aspectos a valorar	Apache Spark	Apache Storm	Apache Flink
Streaming type	Micro-batches	Nativo	Nativo
MMPM <sup>8</sup>	$0,25 \cdot 10^7$	$1,5 \cdot 10^7$	$0,75 \cdot 10^7$
SRCN <sup>9</sup> (s)	0	25	150
MPDC <sup>10</sup> (%)	0	22.2	12.8

Puesto que, de por si, Apache Spark *Streaming* ya es capaz de procesar una enorme cantidad de mensajes por segundo; que para el caso de uso acotado a

<sup>8</sup>Máximo de Mensajes (aproximado) Procesados por Minuto en la Figura 4.2.

<sup>9</sup>Segundos (aproximados) en Recuperarse de la Caida del Nodo en las Figuras 4.3 y 4.4.

<sup>10</sup>Porcentaje de Mensajes Perdidos Durante la Caída en la Figura 4.4.

nivel TFM no se estima que se necesite un ratio de procesamiento por segundo muy exagerado y, habiendo visto la fiabilidad de Apache Spark *Streaming* ante un posible fallo nodo, se decide utilizar Apache Spark Streaming como motor de procesamiento de datos. La Tabla 4.2 resume la comparación de los tres motores de procesamiento evaluados.

## Iteración 3: Implementación de la ingesta de datos

Una vez decidido el ingestor de datos y el motor de procesamiento a utilizar es el momento de empezar a armar la solución. Llegados a este punto, es necesario recordar que, tal y como se comentó en la iteración 1 del Apartado 4.1, se utilizan *scripts* de Python para simular el envío de datos que, en un caso de uso real de *smart farming*, sería llevado a cabo por sensores de suelo IoT.

El primer paso es, por tanto, definir los datos con los que se va a trabajar. Al no contar con sensores de suelo IoT que proporcionen dichos datos, hay dos vías entre las que se pueden elegir: utilizar datos sintéticos, es decir, generados; o buscar algún conjunto de datos que proporcionen el tipo de datos que se necesita. Puesto que a priori, la segunda vía parece más sencilla y rápida, se inicia una búsqueda de datos de calidad de suelo.

### El conjunto de datos utilizado

Es aquí donde entra en juego la Red Internacional de Calidad de Suelo (*ISMN*<sup>11</sup> por sus siglas en inglés). El *ISMN* es una colaboración a nivel internacional que tiene como objetivo crear y mantener una base de datos global de medidas de humedad del suelo tomadas directamente en el terreno. Esta institución, dispone de varias redes de estaciones equipadas con varios tipos de sensores capaces de medir características del suelo.

Desde su página web, es posible visualizar los datos de manera gratuita, sin embargo, para descargar los datos es necesario registrarse. Como se puede comprobar desde su visor de datos<sup>12</sup>, *ISMN* cuenta con muchísimas estaciones ubicadas en distintos países de las cuales obtener datos.

Llegados a este punto, se puso el foco en una estación ubicada en España que tuviese datos recientes. XMS-CAT<sup>13</sup> es una estación ubicada en Cataluña equipada con sensores de humedad y temperatura de suelo a tres profundidades distintas; temperatura del aire y precipitación. Además, cuenta con datos relativos a este año.

Encontrada la red de estaciones sobre la que poner el foco y, previo registro en *ISMN*, se descargaron los datos desde el 1 de Enero de 2023 a 1 de Abril de 2023

<sup>11</sup><https://ismn.earth/en/>

<sup>12</sup><https://ismn.earth/en/dataviewer>

<sup>13</sup><https://ismn.earth/en/networks/?id=XMS-CAT>

de una estación de la red XMS-CAT llamada Clarella. La Figura 4.5 muestra información detallada sobre la estación Clarella, su equipamiento y el rango de datos disponible.

<b>Station:</b>	Clarella (CLA)
<b>Network:</b>	XMS-CAT
<b>Network URL:</b>	<a href="https://visors.icgc.cat/mesurasols/#9/42.1765/1.1132">https://visors.icgc.cat/mesurasols/#9/42.1765/1.1132</a>
<b>Data available</b>	<b>Variables measured</b>
from: 2021-12-01 00:00:00 to: 2023-05-03 17:00:00	soil temperature precipitation air temperature soil moisture
<b>Soil Moisture depths</b>	<b>Soil Moisture sensors</b>
0.05 - 0.05 m 0.20 - 0.20 m 0.50 - 0.50 m	Campbell Scientific, CS655,
<b>View Data</b>	

Figura 4.5: Información estación XMS-CAT Clarella

## El formato de entrada

El *ISMN* ofrece dos formatos a la hora de descargar los datos: *CEOP*<sup>14</sup> y *header+values*<sup>15</sup>. En el caso de este trabajo, se eligió el formato *header+values*. La Figura 4.6 muestra las cinco primeras líneas de uno de los archivos descargados correspondiente al sensor de humedad del suelo a 5 cm de profundidad.

XMS-CAT_XMS-CAT_Clarella_sm_0.050000_0.050000_CS655_20230101_20230401.stm					
1	XMS-CAT	XMS-CAT	Clarella	42.11946	
	2.28235		974.0	0.0500	0.0500 CS655
2	2023/01/01	00:00	0.11	G	M
3	2023/01/01	01:00	0.11	G	M
4	2023/01/01	02:00	0.109	G	M
5	2023/01/01	03:00	0.109	G	M

Figura 4.6: Ejemplo de datos con formato *headers+values* de la estación Clarella

Como se puede comprobar, en la primera línea del fichero hay nueve campos mientras que en el resto de líneas hay tan solo cinco campos. En el nombre del fichero también parece haber información valiosa. En cualquier caso, en el enlace del formato *header+values* adjuntado a pie de página se diferencian y explican los campos existentes en el nombre del fichero, la línea de cabeceras y el resto de líneas de

<sup>14</sup><https://ismn.earth/en/data/ceop-standard/>

<sup>15</sup><https://ismn.earth/en/data/header-value-files/>

valores.

En la Figura 4.7 se diferencian y explican los campos que hay en el nombre del fichero. Por último, en la Figura 4.8 se diferencian y explican los campos que hay en la línea de *header* y en las líneas de *values*.

```
Dataset Filename
CSE_Network_Station_Variablename_depthfrom_depthto_sensornname_startdate_enddate.ext
CSE          Continental Scale Experiment (CSE) acronym, if not applicable use Networkname
Network      Network abbreviation (e.g., OZNET)
Station      Station name (e.g., Widgiewa)
Variablename Name of the variable in the file (e.g., Soil-Moisture)
depthfrom    Depth in the ground in which the variable was observed (upper boundary)
depthto     Depth in the ground in which the variable was observed (lower boundary)
sensornname Name of the sensor used
startdate   Date of the first dataset in the file (format YYYYMMDD)
enddate     Date of the last dataset in the file (format YYYYMMDD)
ext         Extension .stm (Soil Temperature and Soil Moisture Data Set see CEOP standard)

Example: OZNET_OZNET_Widgiewa_Soil-
Temperature_0.150000_0.150000_20010103_20090812.stm
```

Figura 4.7: Explicación y diferenciación de los campos del nombre de los ficheros

```
File Contents
Example:
REMEDHUS REMEDHUS Zamarron 41.24100 -5.54300 855.00 0.05 0.05
2005/03/16 00:00 10.30 U M
2005/03/16 01:00 9.80 U M
...
Header
CSE IdentifierContinental Scale Experiment (CSE) acronym, if not applicable use Networkname
Network      Network abbreviation (e.g., OZNET)
Station      Station name (e.g., Widgiewa)
Latitude     Decimal degrees. South is negative.
Longitude    Decimal degrees. West is negative.
Elevation    Meters above sea level
Depth from  Depth in the ground in which the variable was observed (upper boundary)
Depth to    Depth in the ground in which the variable was observed (lower boundary)
Sensor       Name of the sensor used to make the measurements

Record
UTC actual date/time      yyyy/mm/dd HH:MM
Variable value
ISMN Quality Flag      Description
Data Provider Quality Flag See details in the section Original Flag Description in the Readme
file or on the page of the Network in the Networks section
```

Figura 4.8: Explicación y diferenciación de los campos de las líneas de los ficheros

La *quality flag*<sup>16</sup>, es un campo que el *ISMN* utiliza para valorar la calidad del dato mediante diferentes técnicas para detectar mediciones sospechosas. Para obtener una descripción detallada de la metodología detrás de las banderas de calidad es necesario revisar el artículo de Dorigo et al. (2013).

<sup>16</sup><https://ismn.earth/en/data/flag-overview/>

Por último, respecto al *quality flag* del proveedor de datos, tal y como se puede ver en la Figura 4.8, aparece la sección *Original Flag Description* del *Readme* del fichero comprimido descargado o en la página del *ISMN* donde se da información de la red, en este caso concreto, XMS-CAT<sup>17</sup>.

## Instalación de Kafka

Llegados a este punto y, antes de desarrollar los *scripts* productores que envíen información a un *topic* de Kafka, es necesario instalar Kafka y crear el entorno de trabajo.

En primer lugar, como entorno de trabajo se utiliza una máquina virtual (VM) Linux, en concreto, la distribución AlmaLinux<sup>18</sup>. En otro orden, para llevar a cabo la instalación de Kafka en dicha VM se ha utilizado la sección Quickstart<sup>19</sup> de la web oficial de Kafka. Es necesario remarcar que, tal y como se indica en el *Quickstart*, antes de instalar Kafka, es necesario tener instalado Java 8 o superior.

Tal y como se puede comprobar en dicha sección, Kafka puede ejecutarse con ZooKeeper o con KRaft. Para este trabajo se ha utilizado la ejecución con KRaft ya que es el enfoque que Apache está adoptando a partir de la versión 3.3. Adicionalmente, en el *Quickstart* también se pueden encontrar comandos importantes como los relativos a iniciar Kafka, crear un *topic* o leer mensajes de un *topic*.

Por último, otra sección bastante interesante de la web oficial de Apache Kafka, fue la sección de introducción<sup>20</sup>. En ella, se describen funcionalidades o términos que ya se han descrito aquí como *topic*, productor o consumidor; aportando esquemas y ejemplos que ayudan a su comprensión.

## El formato de envío

Una vez ya se tiene Kafka funcionando y se conocen los datos a enviar y su formato, es el momento de pasar a implementar los *scripts* de Python que simularán ser sensores de suelo IoT.

En primer lugar, es necesario tener claro cual va ser el formato en el que se van a enviar los datos. Muchos sensores IoT envían los datos en un formato propietario ideado por el propio fabricante. Sin embargo, existen numerosos formatos de serialización estandarizados que se utilizan en IoT como, por ejemplo, JSON, Apache Avro o XML (Luis et al., 2021).

<sup>17</sup><https://ismn.earth/en/networks/?id=XMS-CAT>

<sup>18</sup><https://almalinux.org/>

<sup>19</sup><https://kafka.apache.org/quickstart>

<sup>20</sup><https://kafka.apache.org/intro>

En el caso de este trabajo en concreto, se decide utilizar *JSON* como formato de envío ya que es uno de los más conocidos y con el que más familiarizado se está ya que se ha trabajado con el previamente. Antes de pasar al desarrollo del primer productor, es necesario decidir como organizar los *topics* de Kafka y sus correspondientes particiones.

### Organización de los *topics* de Kafka

Tal y como se describió al hablar de Kafka en la iteración 1 del Apartado 4.1, los productores envían los mensajes a los temas y los consumidores, que pueden suscribirse a esos temas, leen los mensajes de esos temas (Mătăcuță and Popa, 2018). Cada *topic* contiene a su vez particiones. Esto se explica bastante bien en la sección de introducción de la web de Apache Kafka que se menciona en la iteración 1 del Apartado 4.1.

En dicha sección se especifica: “Cuando se publica un nuevo evento en un tema, en realidad se agrega a una de las particiones del tema. Los eventos con la misma clave de evento (por ejemplo, un ID de cliente o vehículo) se escriben en la misma partición”. La Figura 4.9, presente en la sección de introducción de la web de Apache Kafka, muestra de manera esquemática el funcionamiento de los *topics* y sus particiones.

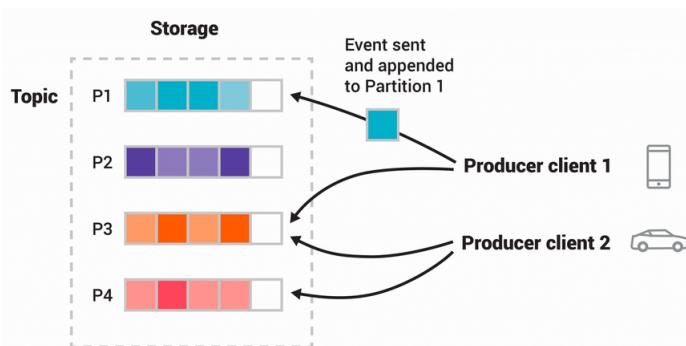


Figure: This example topic has four partitions P1–P4. Two different producer clients are publishing, independently from each other, new events to the topic by writing events over the network to the topic's partitions. Events with the same key (denoted by their color in the figure) are written to the same partition.

Note that both producers can write to the same partition if appropriate.

Figura 4.9: Esquema del funcionamiento de los *topics* de Kafka y sus particiones

Llegados a este punto, es necesario definir organización de los *topics* de Kafka que se van a crear. En un primer momento, se piensa generar un *topic* por estación y utilizar como clave de mensaje el identificador de sensor que enviaba. De esta manera y, aplicando la estructura descrita a la estación con la que se trabaja actualmente en este TFM, habría un *topic* llamado Clarella con tantas particiones como

sensores diferentes tenga equipados dicha estación.

El punto negativo de esta idea es que crece de manera exponencial hasta volverse inmanejable en cuanto que se empieza a trabajar con varias estaciones. Para evitar ese problema en el futuro se decide enfocar la estructura de otra manera. Puesto que los sensores de las estaciones del *ISMN* suelen coincidir en cuanto a tipo e incluso profundidad, se decide crear un *topic* por cada tipo de sensor que se quiera simular y utilizar el nombre de la estación como clave de mensaje.

De esta forma es más sencillo trabajar ya que, ante un *topic* con 5 particiones, si se están recibiendo datos de 3 estaciones distintas, habrá 3 particiones con datos y 2 esperando a que entren datos de nuevas estaciones. Esta organización se hace de manera automática y es posible gracias a la clave de los mensajes. Además, las particiones de los *topics* pueden aumentarse si se necesitase. La Figura 4.10 muestra, de manera esquemática, la estructura de *topics* y particiones que se ha decidido utilizar.

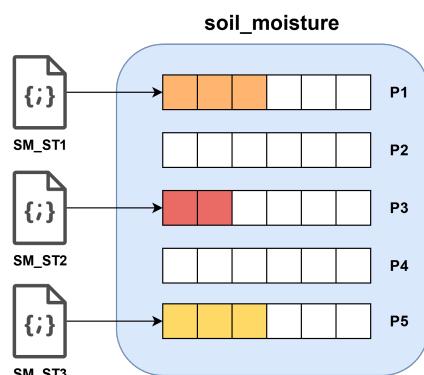


Figura 4.10: Organización de *topics* y particiones adoptada para el trabajo

## Implementación de los productores

Habiendo decidido el formato de envío y la organización de los *topics* y particiones de Kafka, se pasa a la implementación del primer productor. El primer productor desarrollado envía los datos del sensor de humedad de suelo de 5 cm de profundidad de la estación Clarella. A partir de aquí, con tan sólo cambiar el nombre del fichero del que se van a leer los datos, se podrán crear nuevos productores.

La lógica de los productores se basa en identificar y extraer de la información que hay en el nombre de los ficheros descargados, la línea de cabeceras y las líneas de valoras. Para ello, se cuenta con la explicación que el *ISMN* proporciona en su página web mostrada en la Figura 4.7 y Figura 4.8.

Debido a ello, se cuentan con tres métodos auxiliares los cuales van extrayendo e identificando la información para que, en el método principal, se vaya completando el mensaje *JSON* a enviar. A continuación, se describe de manera breve las funciones auxiliares y su funcionamiento:

- **define\_sensor\_type:** Determina el tipo de sensor en función del nombre del archivo.
- **process\_header:** Toma como argumento de entrada la línea de cabeceras del fichero. Posteriormente limpia, formatea e identifica los campos de dicha línea. Finalmente, crea y devuelve un diccionario con la información extraída de la línea de cabeceras.
- **complete\_message:** Toma como argumento de entrada el diccionario con la información extraída de la línea de cabeceras y una línea de valores. Extrae la información de la línea de valores y añade dicha información al diccionario con la información de las cabeceras completando así el mensaje que se va a enviar.

Obviamente, el *script* cuenta con una función principal que es la que va llamando a los métodos auxiliares que se acaban de describir. El Algoritmo 1 se corresponde al pseudocódigo de la función principal.

---

#### Algorithm 1 read\_and\_send\_data()

---

```
1: Input: None
2: Output: None
3: while true do
4:   sensor_type ← DEFINE_SENSOR_TYPE()
5:   producer ← PRODUCER({'bootstrap.servers': 'localhost:9092'})
6:   f ← OPEN(FILENAME, 'r')
7:   message ← PROCESS_HEADER(f.readline())
8:   for all line in f do
9:     COMPLETE_MESSAGE(message, line)
10:    topic ← LOWER(sensor_type).REPLACE(' ', '_')
11:    PRODUCER.PRODUCE(topic,           key=message['station'],
12:                      value=JSON.DUMPS(message).ENCODE('utf-8'))
13:    PRINT('A Message has been sent')
14:   end for
15:   PRODUCER.FLUSH()
16: end while
```

---

Dicha función, abre conexión con Kafka y, mediante el uso de las funciones auxiliares, va leyendo el fichero de datos, extrayendo su información y creando el mensaje en formato *JSON*. Esto puede verse de la línea 4 a la línea 9 del Algoritmo

### Algoritmo 1.

A continuación, cada 10 segundos, se envía el mensaje con la información extraída al *topic* de Kafka correspondiente según dicha información. Esto puede verse de la línea 10 a la 13 del Algoritmo 1.

De esta manera y, con solo cambiar la variable global *FILENAME* (línea 6 del Algoritmo 1), se pueden simular tantos sensores IoT como ficheros del *ISMN* se tengan descargados.

Por último es necesario mencionar que el código completo del productor está publicado y puede ser consultado en mi GitHub<sup>21</sup>. Más información acerca del código o el repositorio se puede encontrar en el Apéndice A.

## Iteración 4: Integración entre sensores, ingestor y procesador

Una vez desarrollada la ingesta de datos, es necesario implementar el procesamiento que se le va a aplicar a los datos ingeridos por Kafka. No obstante, antes de implementar cualquier procesamiento, es necesario instalar en el entorno de trabajo del motor de procesamiento elegido al final de la segunda iteración del Apartado 4.1.

Al igual que en el caso de Kafka, en la web oficial de Apache Spark se puede encontrar una sección llamada *Installation*<sup>22</sup> en la que se explica claramente cómo instalar Spark. Existen varias formas de instalar Spark. Para el caso de este trabajo, se instaló Spark mediante descarga manual, es decir, se descargaron los binarios de Spark comprimidos y se trajeron en un directorio concreto.

También es necesario remarcar que Spark posee distintas API para distintos lenguajes de programación. Para el desarrollo de este trabajo, dado que ya se ha utilizado Python para el desarrollo de la ingesta de los datos, se utilizó la API de Spark para Python llamada PySpark<sup>23</sup>.

Dentro de PySpark, se utiliza lo que Apache denomina como *Structured Streaming*<sup>24</sup>. La propia página web oficial de Apache Spark *Structured Streaming* detalla: “la idea clave en *Structured Streaming* es tratar un flujo de datos en vivo como una tabla que se está continuamente agregando lo que lo hace muy similar a un modelo de procesamiento por lotes”.

La Figura 4.11 está sacada de la página web oficial de Apache Spark *Structured Streaming* y muestra de manera esquemática su funcionamiento.

<sup>21</sup><https://github.com/jramon-mm/14MBID-TFM>

<sup>22</sup>[https://spark.apache.org/docs/latest/api/python/getting\\_started/install.html](https://spark.apache.org/docs/latest/api/python/getting_started/install.html)

<sup>23</sup><https://spark.apache.org/docs/latest/api/python/index.html>

<sup>24</sup><https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#>

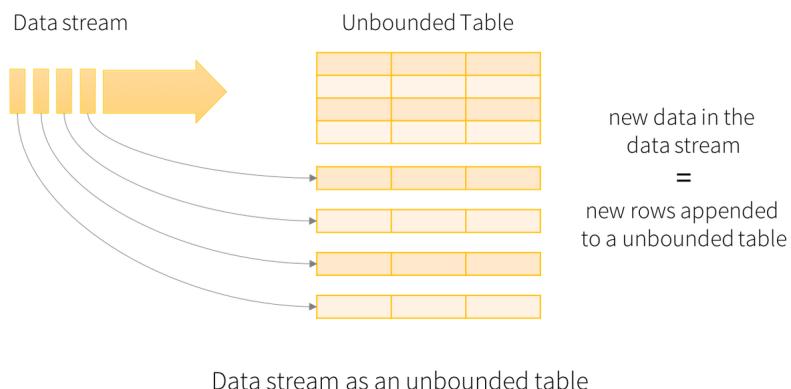


Figura 4.11: Esquema funcionamiento Spark Structured Streaming

Por último, antes de proceder con el desarrollo de los consumidores, se tuvo una reunión con la tutora Judith Cardinale y Saúl Puentes, otro estudiante que también está desarrollando un TFM dentro del contexto de UbiCDatos, para el control inteligente de riegos para cultivos. La idea es que, en un futuro, este TFM y el de Saúl Puentes se integren y, la salida de este TFM sea consumida por el TFM de Saúl Puentes, de manera que se implemente un sistema de riego inteligente. En dicha reunión, por lo tanto, se acuerdan los campos útiles que la salida ha de tener y el procesamiento que hay que aplicar a los datos.

Las variables de salida acordadas fueron: el *datetime* de la medición, la estación (utilizada además como clave del mensaje), el modelo de sensor, la profundidad y el valor de la medición. Respecto al procesamiento que hay que aplicar, se trata de un básico filtrado eliminando todas las mediciones que no tengan el valor G (*Good*) en el campo *Quality Flag* (descrito en la iteración 3 del Apartado 4.1). De esta manera se elimina cualquier medición dudosa o posiblemente errónea que haya sido ingestada en Kafka.

### Implementación de los consumidores

Teniendo Spark instalado, habiendo definido la API a utilizar y habiendo acordado el procesamiento a aplicar, se puede proceder a la implementación de los consumidores. Como se detalló en la tercera iteración del Apartado 4.1 y se mostró en la Figura 4.9, se crea un *topic* por cada tipo de sensor que se quiera simular y se utiliza el nombre de la estación como clave de mensaje. La lógica del programa PySpark que consume los datos de los *topics* de Kafka se basa principalmente en 2 bloques más el método principal:

- **Definición de variables globales:** Se inicializan las variables necesarias para el correcto funcionamiento del programa como los datos de conexión a Kafka,

---

overview

estructura del mensaje a consumir o los directorios de salida.

- **read\_from\_kafka:** Toma como parámetro de entrada la sesión de Spark. Se encarga de leer el mensaje y crear un *dataframe* con todos los campos e información del mensaje. Este proceso es posible gracias a la variable en al que se definen la estructura de los mensajes que se ha mencionado anteriormente. Finalmente, hace un primer filtrado de columnas interesantes del *dataframe* y lo devuelve.

El Algoritmo 2 muestra el pseudocódigo de la función principal del programa PySpark que consume los datos de Kafka. En primer lugar, el método *main* abre una sesión de Spark y lee los mensajes de Kafka mediante el método *read\_from\_kafka* que se ha descrito en el párrafo anterior. Esto puede verse en las líneas 1, 2 y 3 del Algoritmo 2.

Seguidamente, aplica el procesamiento acordado en la reunión con Saúl y Judith (líneas 4 y 5 del Algoritmo 2). Finalmente, filtra las columnas de salida en base a lo comentado en dicha reunión y escribe el resultado en formato CSV (líneas 6 - 14) del Algoritmo 2).

---

#### Algorithm 2 Consumer Main Method

---

```
1: spark ← SparkSession.builder.appName('KafkaSoilMoistureConsumer').  
2:   .getOrCreate()  
3: kafka_df ← read_from_kafka(spark)  
4: g_flags_df ← kafka_df.filter(col('ismn_quality_flag') == 'G')  
5: g_flags_df ← g_flags_df.select(['measurement_datetime', 'station', 'sensor_model', 'depth_from', 'measurement'])  
6: function SAVE_BATCH_TO_CUSTOM_PATH(batch_df, batch_id)  
7:   for each row in batch_df.collect() do  
8:     output_final_path ← "{output_base_path}/{topic}/{row['station']}"  
9:     batch_df.coalesce(1).write.csv(output_final_path, mode='overwrite',  
  header=True)  
10:  end for  
11: end function  
12: query ← g_flags_df.writeStream.foreachBatch(save_batch_to_custom_path)  
13:   .option('checkpointLocation', checkpoint_base_path).start()  
14: query.awaitTermination()
```

---

Finalmente, es necesario remarcar que un mismo consumidor puede estar suscrito a uno o varios *topics*. De esta manera y, a partir del código de un consumidor, pueden generarse consumidores de otros *topics*, pudiendo alcanzar el nivel de paralelismo que se deseé.

Al igual que en el caso del productor, el código completo del consumidor está

publicado y puede ser consultado en GitHub<sup>25</sup>. Para más información acerca del código o el repositorio dirigirse al Apéndice A.

## 4.2. Fase 2: Detección de enfermedades de cultivos a partir de imágenes

Al igual que en la fase 1, descrita en la Sección 4.1, es necesario, antes que nada, definir los requisitos que la solución ha de cumplir:

- **REQ1:** Los datos de entrada del sistema han de ser imágenes de cultivos.
- **REQ2:** El sistema tendrá que detectar la enfermedad de como mínimo un cultivo.
- **REQ3:** Para que un modelo sea considerado válido tendrá que superar el 90 % de precisión con los datos de prueba.
- **REQ4:** Se han de generar varios modelos válidos para poder elegir entre ellos.

### Iteración 1: Búsqueda del conjunto de datos

Antes de poder entrenar siquiera cualquier modelo, es necesario encontrar un conjunto de datos que se ajuste a las necesidades de la tarea que se quiere desempeñar. Atendiendo a lo descrito en el Apartado 2.1, la detección de enfermedades de cultivos a partir de imágenes, supone un claro problema de clasificación multiclas, en este caso, a partir de imágenes.

Por este motivo, esta primera iteración se dedica a buscar un *dataset* que cumpla los requisitos necesarios para resolver un problema de clasificación de imágenes multiclas. Pero, ¿qué requisitos ha de tener el conjunto de datos para que sea válido para la tarea que se tiene entre manos?

Tal y como se comentó en el Apartado 2.3, las hojas de los cultivos son un elemento clave a la hora de valorar la salud, el entorno y la madurez de los cultivos. Se debe buscar, por tanto, conjuntos de datos que contengan, a poder ser, miles de imágenes de hojas de distintas plantas.

Además, las imágenes han de estar etiquetadas ya que, de acuerdo a lo descrito en el Apartado 2.1, se está ante un enfoque de aprendizaje supervisado en el que los datos deben estar etiquetados y cuyo objetivo es predecir un valor discreto, en este caso, el estado de salud de una planta.

---

<sup>25</sup><https://github.com/jramon-mm/14MBID-TFM>

Para buscar el *dataset* se utilizó, principalmente, Google *Dataset Search*<sup>26</sup> y Kaggle<sup>27</sup>. Finalmente, fue en Kaggle donde se encontró el conjunto de datos que cumplía con todos los requisitos que se estaba buscando.

El *dataset* con el que se trabajó se llama *Plant Disease Expert*<sup>28</sup>, fue subido a Kaggle por Sadman Sakib Mahi en 2022 (Mahi, 2022) y contiene imágenes de hojas de 20 cultivos distintos, de los cuales, 10 tienen más de una categoría, con lo que resultan aptos para usarlos en un problema de clasificación multiclas.

De esos 10 cultivos con más de una categoría, se descartaron aquellos que no contaran con cuatro o más categorías o que en alguna de sus categorías tuviesen menos de 1000 imágenes. Como resultado de dicho descarte, fueron cuatro los cultivos que quedaron: manzana, maíz, tomate y uva.

Finalmente y, a pesar de que una de sus categorías está muy descompensada en cuanto a número de imágenes con las demás, se eligió el cultivo de la uva. La principal razón para elegirlo fue que, exceptuando una de las categorías, era el cultivo con más imágenes disponibles, aunque vale mencionar que la condición de Manchego del autor de este TFM tuvo también algo que ver en la decisión.

## Iteración 2: Carga e inspección del conjunto de datos

Una vez definidos los datos con los que se va a trabajar es el momento de elegir el entorno de trabajo. Al igual que ocurrió en la fase 1, abordada en el Apartado 4.1, existen dos principales opciones: trabajar en local o trabajar en un entorno *Cloud*.

Puesto que el PC del que se dispone no es precisamente potente (en términos de hardware) y el entrenamiento de modelos basados en CNN suelen requerir bastantes recursos de computación, se decide utilizar un entorno *Cloud* como es Google Colab<sup>29</sup>.

Google Colab es una plataforma en línea gratuita ofrecida por Google que permite a los usuarios escribir y ejecutar código en Python a través de un navegador web. Aparte de no requerir ningún tipo de configuración, también permite acceder, gratuitamente, a hardware acelerado por *GPU*<sup>30</sup> lo que facilita el entrenamiento de modelos de aprendizaje profundo y el procesamiento de datos intensivos.

Dado que se va a trabajar en Google Colab, es necesario subir los datos con los que se va a trabajar a la sesión de Google Colab. Debido a que son casi 36000

<sup>26</sup><https://datasetsearch.research.google.com/>

<sup>27</sup><https://www.kaggle.com/>

<sup>28</sup><https://www.kaggle.com/datasets/sadmansakibmahi/plant-disease-expert>

<sup>29</sup><https://colab.research.google.com/>

<sup>30</sup><https://colab.research.google.com/notebooks/gpu.ipynb>

imágenes, cargar los datos cada vez que se va a trabajar a la sesión del *notebook* de Colab es totalmente inviable. Para evitar esta carga repetida, se decide subir los datos a Google Drive<sup>31</sup> y montar, en el cuaderno de Colab, el directorio de Drive donde se han guardado las imágenes.

Una vez que se tiene alcance a los datos de trabajo desde el *notebook* de Colab, se puede empezar a inspeccionar los datos. El número de categorías en las que se dividen los datos, el número de imágenes que hay en cada categoría o el tamaño de las imágenes son factores clave que hay que conocer antes de poder crear una estrategia de entrenamiento.

Tal y como se comentó en el apartado anterior, finalmente se decidió focalizar la detección de enfermedades en el cultivo de la uva. Para el cultivo de la uva, el conjunto de datos distingue cuatro categorías: cultivo sano, *Black Rot* (Szabó et al., 2023), *Esca* (Ouadi et al., 2019), y *Isariopsis* (Welter et al., 2019).

Antes de pasar a detallar cómo se llevó a cabo la inspección de los datos, es necesario mencionar que el código completo del *notebook* de Colab está también disponible en GitHub<sup>32</sup>. Para más información acerca del código o el repositorio dirigirse al Apéndice A.

Básicamente, la inspección de los datos consiste en recorrer los directorios de las distintas categorías y creando un *dataframe* por categoría con la siguiente información: *filepath*, dimensión y categoría. De esta manera, una vez acabado el proceso, se tiene la siguiente información:

- **Número de imágenes de cada categoría:** Viene dado por el número de filas del *dataframe*. La Tabla 4.3 muestra la cantidad de imágenes existentes en cada categoría.
- **Dimensión y canales de las imágenes:** Se consultan los distintos valores existentes en la columna Dimensión. De esta manera, se comprueba que todas las imágenes son a color, es decir, tienen 3 canales y una dimensión de 256x256 píxeles.

Tabla 4.3: Número de imágenes en cada categoría de la uva

Categorías	Cultivo sano	<i>Black Rot</i>	<i>Esca</i>	<i>Isariopsis</i>
Número de imágenes	1017	11328	13284	10332

Finalmente y, a modo de *checkpoint*, se guarda un CSV de cada *dataframe* para evitar tener que repetir este proceso que, al tratarse de casi 36000 imágenes, es bastante costoso. Como consecuencia de este *checkpoint* surge la sección *Load original CSV* del cuaderno de Colab.

<sup>31</sup><https://www.google.es/drive/>

<sup>32</sup><https://github.com/jramon-mm/14MBID-TFM>

## Iteración 3: División de los datos de trabajo y desarrollo de funciones auxiliares

Conociendo a fondo los datos con los que se va a trabajar, el siguiente paso es elaborar la estrategia de entrenamiento. Para el caso concreto del cultivo de la uva, existe un problema el cual se ve claramente en la Tabla 4.3. Las categorías que corresponden a las enfermedades de la uva tienen todas más de 10000 imágenes mientras que, la categoría *healthy* cuenta tan sólo con 1017 imágenes.

Este terrible desequilibrio entre categorías plantea un problema de cara al entrenamiento ya que es posible que 1017 fotos no sean suficientes para obtener un modelo válido. En ese caso, en las otras categorías pueden aumentarse las muestras pero, en el caso de la categoría *healthy* no. Entrenar un modelo con más muestras de ciertas categorías puede resultar en un modelo sesgado que aprenda demasiado bien las categorías de las que más imágenes tiene y falle cuando se enfrente a la categoría con menos muestras.

Para solventar este problema existen dos principales opciones: buscar muestras adicionales en otros conjuntos de datos de la categoría que tiene pocas imágenes o probar a entrenar un modelo con tantas muestras por categoría como número de imágenes tiene la categoría con menos muestras. Puesto que se quería evitar tener que buscar de nuevo datos con los que trabajar, se decidió entrenar un modelo con 1000 imágenes de cada categoría y ver qué resultados se obtenían.

### División de los datos y lectura de los mismos

El primer paso antes de entrenar cualquier modelo es dividir los datos en datos de entrenamiento, validación y test. En este contexto, a partir de los *dataframes* generados en la iteración anterior, se seleccionan aleatoriamente 1000 registros de cada uno y se les añaden dos columnas más:

- **Category Tag:** Valor entero que representa a la categoría a la que la imagen pertenece.
- **Partition:** Cadena que indica a que partición de datos a la que imagen pertenece. Los porcentajes usados para la división de los datos han sido: 70 % datos de entrenamiento, 20 % datos de validación y 10 % datos de test.

Utilizando la etiqueta *Partition*, a partir de los *dataframes* con 1000 registros de cada categoría se generan tres nuevos *dataframes*, uno con los datos de entrenamiento, otro con los de validación y, finalmente, uno con los test. Por último, una vez juntados los datos de todas las categorías en el *dataframe* que les corresponda según a la partición de datos a la que pertenezcan, se desordenan los datos de manera aleatoria.

Teniendo ya, no los datos en si pero, si su información, dividida en entrenamiento, validación y test; se procede a cargar los datos en las variables que posteriormente se usaron para entrenar. Es necesario mencionar que, para llevar a cabo el entrenamiento se necesitan principalmente cuatro variables:

- **x\_train:** *Numpy Array*<sup>33</sup> que contiene los datos normalizados de las imágenes de entrenamiento.
- **y\_train:** *Numpy Array* que contiene las etiquetas correspondientes a las imágenes contenidas en x\_train en formato *integer*.
- **x\_val:** *Numpy Array* que contiene los datos normalizados de las imágenes de validación.
- **y\_val:** *Numpy Array* que contiene las etiquetas correspondientes a las imágenes contenidas en x\_val en formato *integer*.

Una vez se han rellenado las variables x\_train, y\_train, x\_val e y\_val con los datos pertinentes, se puede proceder a construir el modelo y entrenarlo. Sin embargo, antes de ello, se desarrollaron unas funciones auxiliares que fueron de gran ayuda, especialmente para generar varios modelos válidos.

## Desarrollo de funciones auxiliares

A continuación, se pasan a detallar las funciones auxiliares desarrolladas y su objetivo:

- **save\_model\_topology:** Guarda la topología de un modelo creado en formato JSON.
- **load\_model\_topology:** Carga la topología de un modelo previamente guardado desde un archivo JSON.
- **load\_model\_weights:** Carga los pesos de un archivo HDF5<sup>34</sup> en un modelo existente.
- **save\_training\_info:** Guarda el historial de entrenamiento de un modelo en formato *Pickle*<sup>35</sup>.
- **load\_training\_info:** Carga la información del historial de entrenamiento de un modelo desde un archivo en formato *Pickle*.
- **show\_training\_info:** Muestra gráficamente información sobre el entrenamiento de un modelo.

---

<sup>33</sup><https://numpy.org/doc/stable/reference/generated/numpy.array.html>

<sup>34</sup>[https://www.tensorflow.org/tutorials/keras/save\\_and\\_load?hl=es-419#hdf5\\_format](https://www.tensorflow.org/tutorials/keras/save_and_load?hl=es-419#hdf5_format)

<sup>35</sup><https://docs.python.org/3/library/pickle.html>

- **save\_testing\_info:** Guarda información sobre las pruebas de un modelo en un archivo de texto.
- **show\_images:** Muestra imágenes de un *dataframe* de muestra en un formato de cuadrícula, donde se muestran cinco imágenes por fila (por defecto) y se pueden comparar las predicciones con las etiquetas originales si se proporcionan.
- **get\_category\_text:** Utilizada por la función anterior, mapea una etiqueta de categoría en formato *integer* a su correspondiente texto de categoría.

Una vez se tienen los datos divididos en entrenamiento, validación y test, cargados en las variables de entrenamiento y, habiendo desarrollado funciones auxiliares que ayudan a persistir los modelos que se van a ir creando y entrenando, se procede al entrenamiento de los modelos. Como se comentó en el Apartado 4.2, se han de generar varios modelos válidos (precisión en test mayor al 90 %) para poder elegir entre ellos.

Por último, mencionar de nuevo que el código completo del *notebook* de Colab está disponible en GitHub<sup>36</sup> y que en el Apéndice A hay más información relativa al código y al repositorio.

## Iteración 4: Entrenamiento de los modelos

En primer lugar, es necesario aclarar que, para llevar a cabo la construcción, entrenamiento y testeо de los modelos se va a utilizar la librería TensorFlow<sup>37</sup>. TensorFlow es una biblioteca de código abierto desarrollada por Google que se utiliza principalmente para la creación y entrenamiento de modelos de aprendizaje automático.

Antes de pasar a detallar los modelos entrenados, es importante presentar el principal problema que se encontró a la hora de llevar a cabo los entrenamientos. Tal y como se analizó en la iteración 2 del Apartado 4.2, todas las imágenes con las que se va a trabajar son imágenes a color con una dimensión de 256x256 píxeles. Trabajar con los datos de ese tamaño resultó imposible ya que el entorno de Google Colab colapsaba debido a la limitación de memoria RAM.

Para solventar esta situación, se hicieron *resizes* a la hora de cargar los datos en las variables de entrenamiento (iteración 3 del Apartado 4.2). Obviamente esto funcionó ya que, al reducir el tamaño de los datos de entrada se utiliza, por ende, menos memoria RAM. Los tamaños reducidos que se han utilizado son 128x128 y 64x64 píxeles.

<sup>36</sup><https://github.com/jramon-mm/14MBID-TFM>

<sup>37</sup><https://www.tensorflow.org>

## Aspectos comunes de modelos entrenados

Antes de pasar a describir los modelos que se han obtenido, se enumeran a continuación ciertos aspectos que todos los modelos tienen en común con el fin de evitar repeticiones cuando se describe específicamente cada modelo:

- **Funciones de activación de las capas de salida:** En todas las capas de salida de todos los modelos entrenados se utiliza la función de activación Softmax<sup>38</sup>. La utilización de Softmax se debe a que la capa de salida de todos los modelos es de 4 neuronas (tantas neuronas como categorías a predecir). Para más información acerca de las funciones de activación consultar el documento publicado por Sharma et al. (2017).
- **Funciones de activación del resto de capas:** En todas las capas que no son de salida de todos los modelos entrenados se utiliza la función de activación ReLU<sup>39</sup>. Esta función de activación es ampliamente conocida por su eficiencia computacional y por favorecer la convergencia gracias a no saturar. Para mas información acerca de las funciones de activación consultar el documento publicado por Sharma et al. (2017).
- **Padding:** Todas las CNN de todos los modelos entrenados tienen *padding* por lo que no se pierde ningún tipo de información de los bordes de la imagen durante la convolución.
- **Optimizador:** El optimizador utilizado para compilar todos los modelos entrenados ha sido Adam ya que está demostrado que mejora el rendimiento (Jais et al., 2019). A pesar de ello, la tasa de aprendizaje que se le introduce como parámetro al optimizador si cambia entre modelos.
- **Función de pérdida:** La función de pérdida utilizada para compilar todos los modelos entrenados ha sido *Sparse Categorical Crossentropy*<sup>40</sup>. El uso de dicha función viene condicionado por tener las etiquetas en formato *integer*.
- **Métrica:** La métrica especificada al compilar todos los modelos entrenados ha sido *Accuracy*<sup>41</sup>.

Además, se definieron dos *callbacks* que se utilizaron en los entrenamientos de todos los modelos construidos:

- **Early stopping:** Tiene como objetivo monitorizar las pérdidas del modelo con el conjunto de validación para tratar de evitar el sobreajuste ya que, cuando la pérdida en el conjunto de datos de validación deja de mejorar, es un indicio de que el modelo podría haber alcanzado su capacidad máxima de generalización

<sup>38</sup>[https://www.tensorflow.org/api\\_docs/python/tf/keras/activations/softmax](https://www.tensorflow.org/api_docs/python/tf/keras/activations/softmax)

<sup>39</sup>[https://www.tensorflow.org/api\\_docs/python/tf/keras/activations/relu](https://www.tensorflow.org/api_docs/python/tf/keras/activations/relu)

<sup>40</sup>[https://www.tensorflow.org/api\\_docs/python/tf/keras/losses/](https://www.tensorflow.org/api_docs/python/tf/keras/losses/)

*SparseCategoricalCrossentropy*

<sup>41</sup>[https://www.tensorflow.org/api\\_docs/python/tf/keras/metrics/Accuracy?hl=en](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/Accuracy?hl=en)

por lo que continuar entrenando solo resultaría en un ajuste excesivo a los datos de entrenamiento.

- **Model checkpoint:** Se utiliza para guardar automáticamente el mejor modelo obtenido durante el entrenamiento. El modelo se guarda en formato HDF5 de manera que dicho modelo puede cargarse posteriormente mediante la función auxiliar `load_model_weights` definida en la iteración 3 del Apartado 4.2.

## Entrenamiento del modelo 1

Este primer modelo ha sido entrenado con imágenes redimensionadas 64x64 píxeles. Se trata de un modelo que tiene tres Bloques Convolucionales para reducir dimensionalidad y, tras un aplanado, dos Capas Densas de 512 y 4 neuronas respectivamente.

La Tabla 4.4 muestra los hiperparámetros utilizados para el entrenamiento de este primer modelo.

Tabla 4.4: Hiperparámetros de entrenamiento del modelo 1

<b>Learning Rate</b>	<b>Epochs</b>	<b>Batch Size</b>
0.001	50	32

La Figura 4.12 muestra la topología de este primer modelo mientras que la Figura 4.13 muestra los resultados del entrenamiento de dicho modelo.

## Entrenamiento del modelo 2

La idea de este modelo es mantener la estructura del primer modelo pero variar la dimensión de los datos de entrada. Con esta idea, se cargan los datos en las variables de entrenamiento pero, esta vez, con un tamaño de 128x128 píxeles.

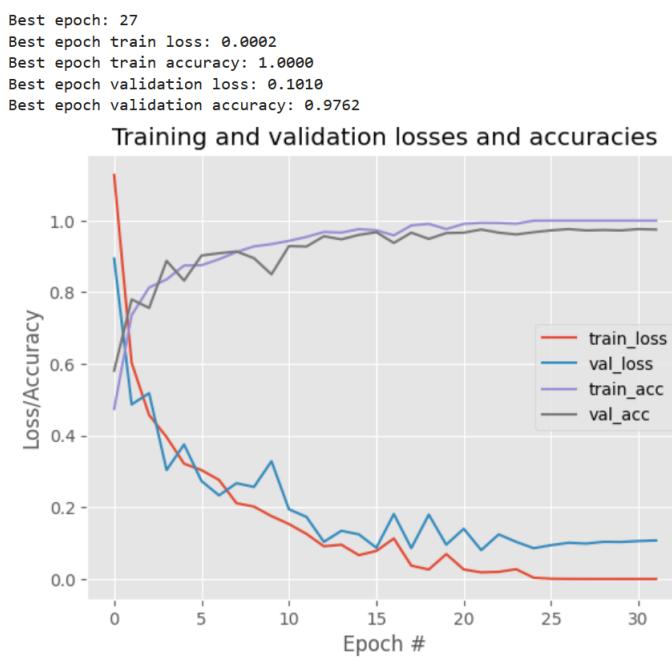
La Figura 4.14 muestra la topología del modelo 2. Como se puede comprobar, la decisión de usar datos de entrada de 128x128 píxeles resulta en un aumento significativo de los parámetros entrenables. Adicionalmente, la Figura 4.15 muestra los resultados del entrenamiento del modelo 2.

Para este segundo modelo, no se variaron los hiperparámetros de entrenamiento con respecto al entrenamiento del primer modelo. La Tabla 4.5 muestra los hiperparámetros utilizados para el entrenamiento del modelo 2.

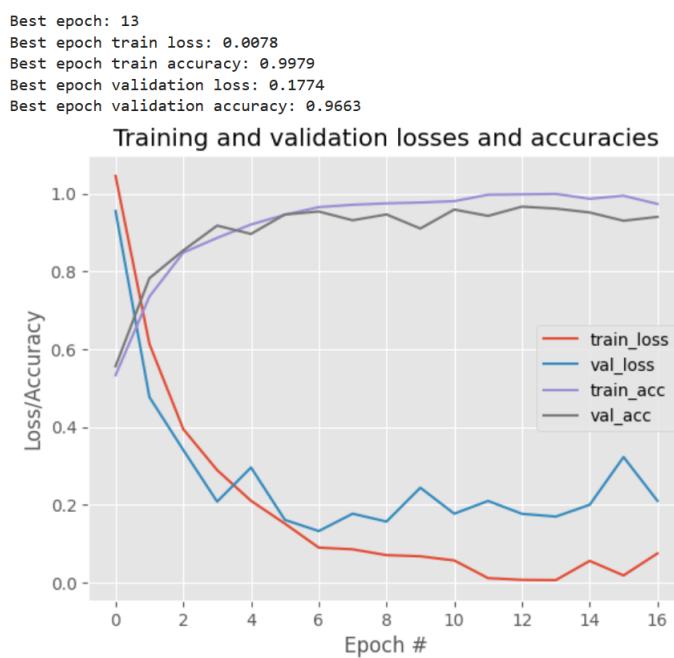
Tabla 4.5: Hiperparámetros de entrenamiento del modelo 2

<b>Learning Rate</b>	<b>Epochs</b>	<b>Batch Size</b>
0.001	50	32

Layer (type)	Output Shape	Param #
<hr/>		
conv2d (Conv2D)	(None, 64, 64, 16)	448
conv2d_1 (Conv2D)	(None, 64, 64, 16)	2320
max_pooling2d (MaxPooling2D)	(None, 32, 32, 16)	0
conv2d_2 (Conv2D)	(None, 32, 32, 32)	4640
conv2d_3 (Conv2D)	(None, 32, 32, 32)	9248
max_pooling2d_1 (MaxPooling2D)	(None, 16, 16, 32)	0
conv2d_4 (Conv2D)	(None, 16, 16, 64)	18496
conv2d_5 (Conv2D)	(None, 16, 16, 64)	36928
max_pooling2d_2 (MaxPooling2D)	(None, 8, 8, 64)	0
flatten (Flatten)	(None, 4096)	0
dense (Dense)	(None, 512)	2097664
dense_1 (Dense)	(None, 4)	2052
<hr/>		
Total params: 2171796 (8.28 MB)		
Trainable params: 2171796 (8.28 MB)		
Non-trainable params: 0 (0.00 Byte)		

Figura 4.12: Topología del modelo 1Figura 4.13: Resultados del entrenamiento del modelo 1

Layer (type)	Output Shape	Param #
<hr/>		
conv2d (Conv2D)	(None, 128, 128, 16)	448
conv2d_1 (Conv2D)	(None, 128, 128, 16)	2320
max_pooling2d (MaxPooling2D)	(None, 64, 64, 16)	0
conv2d_2 (Conv2D)	(None, 64, 64, 32)	4640
conv2d_3 (Conv2D)	(None, 64, 64, 32)	9248
max_pooling2d_1 (MaxPooling2D)	(None, 32, 32, 32)	0
conv2d_4 (Conv2D)	(None, 32, 32, 64)	18496
conv2d_5 (Conv2D)	(None, 32, 32, 64)	36928
max_pooling2d_2 (MaxPooling2D)	(None, 16, 16, 64)	0
flatten (Flatten)	(None, 16384)	0
dense (Dense)	(None, 512)	8389120
dense_1 (Dense)	(None, 4)	2052
<hr/>		
Total params: 8463252 (32.28 MB)		
Trainable params: 8463252 (32.28 MB)		
Non-trainable params: 0 (0.00 Byte)		

Figura 4.14: Topología del modelo 2Figura 4.15: Resultados del entrenamiento del modelo 2

## Entrenamiento del modelo 3

La idea de este modelo es mantener la dimensión de los datos de entrada del segundo modelo y añadir Capas Densas decrecientes entre la primera Capa Densa de 512 neuronas y la Capa Densa de salida, es decir, se mantienen los tres Bloques Convolucionales para reducir dimensionalidad y, tras realizar el aplanado se acaba con cuatro Capas Densas de 512, 256, 128 y 4 neuronas respectivamente.

Obviamente, al mantener los datos de entrada de 128x128 píxeles y añadir más capas, los parámetros entrenables del modelo aumentan, siendo este modelo 3 el modelo con más parámetros entrenables que se ha obtenido. Debido a este aumento de la complejidad del modelo, se tuvo que reducir la tasa de aprendizaje utilizada en los dos primeros modelos ya que se observó un aprendizaje demasiado rápido con respecto a los modelos anteriores.

La Tabla 4.6 muestra los hiperparámetros del entrenamiento del modelo 3. La Figura 4.16 muestra la topología de este tercer modelo mientras que la Figura 4.17 muestra los resultados del entrenamiento de dicho modelo.

Tabla 4.6: Hiperparámetros de entrenamiento del modelo 3

<b>Learning Rate</b>	<b>Epochs</b>	<b>Batch Size</b>
0.0005	50	32

## Entrenamiento del modelo 4

La idea de este modelo es llevar a cabo una vuelta a los orígenes a la vez que se mantiene la adición de Capas Densas decrecientes llevada a cabo en el modelo 3. Con esta idea, se vuelven a cargar los datos en las variables de entrenamiento con un tamaño de 64x64 píxeles.

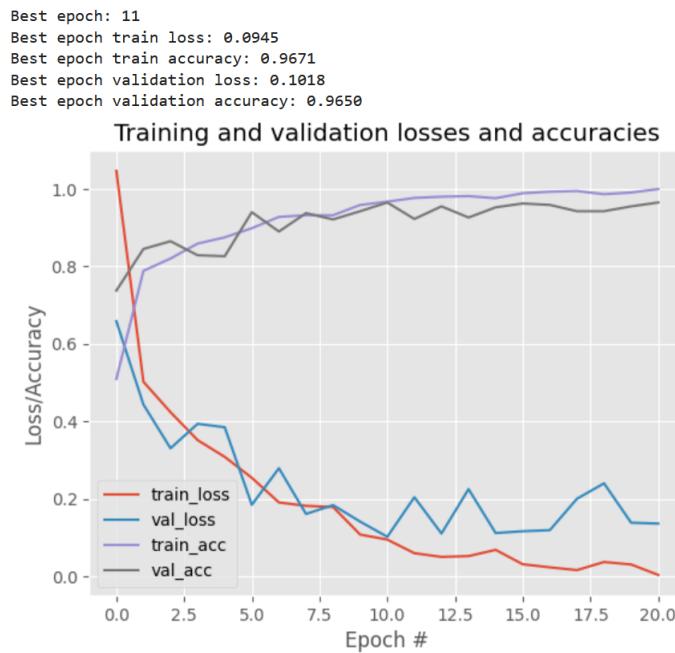
La Figura 4.18 muestra la topología del modelo 4. Como se puede comprobar, la reducción de dimensionalidad de los datos de entrada disminuye significativamente los parámetros entrenables motivo por el cual se volvió a la tasa de aprendizaje utilizada en los modelos 1 y 2.

La Tabla 4.7 muestra los hiperparámetros utilizados para el entrenamiento del modelo 4. Finalmente, la Figura 4.19 muestra los resultados del entrenamiento del modelo 4.

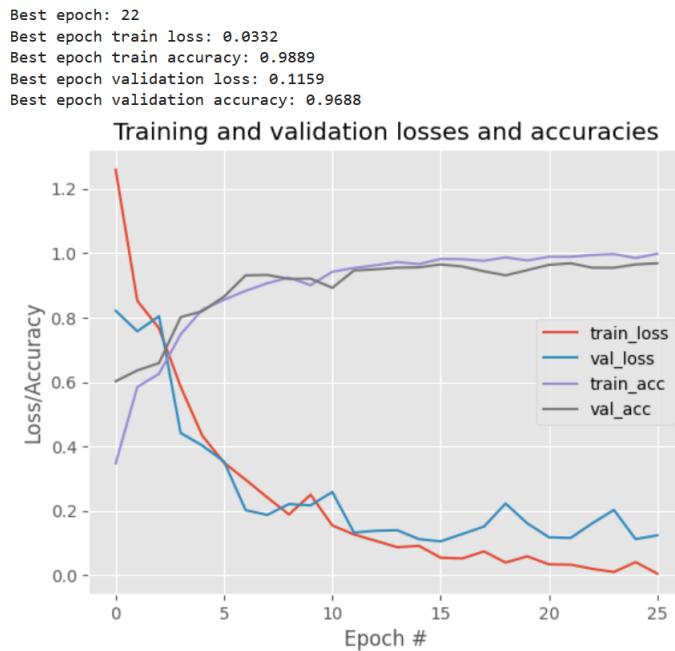
Tabla 4.7: Hiperparámetros de entrenamiento del modelo 4

<b>Learning Rate</b>	<b>Epochs</b>	<b>Batch Size</b>
0.001	50	32

Layer (type)	Output Shape	Param #
<hr/>		
conv2d_6 (Conv2D)	(None, 128, 128, 16)	448
conv2d_7 (Conv2D)	(None, 128, 128, 16)	2320
max_pooling2d_3 (MaxPooling2D)	(None, 64, 64, 16)	0
conv2d_8 (Conv2D)	(None, 64, 64, 32)	4640
conv2d_9 (Conv2D)	(None, 64, 64, 32)	9248
max_pooling2d_4 (MaxPooling2D)	(None, 32, 32, 32)	0
conv2d_10 (Conv2D)	(None, 32, 32, 64)	18496
conv2d_11 (Conv2D)	(None, 32, 32, 64)	36928
max_pooling2d_5 (MaxPooling2D)	(None, 16, 16, 64)	0
flatten_1 (Flatten)	(None, 16384)	0
dense_2 (Dense)	(None, 512)	8389120
dense_3 (Dense)	(None, 256)	131328
dense_4 (Dense)	(None, 128)	32896
dense_5 (Dense)	(None, 4)	516
<hr/>		
Total params: 8625940 (32.91 MB)		
Trainable params: 8625940 (32.91 MB)		
Non-trainable params: 0 (0.00 Byte)		

Figura 4.16: Topología del modelo 3Figura 4.17: Resultados del entrenamiento del modelo 3

Layer (type)	Output Shape	Param #
<hr/>		
conv2d (Conv2D)	(None, 64, 64, 16)	448
conv2d_1 (Conv2D)	(None, 64, 64, 16)	2320
max_pooling2d (MaxPooling2D)	(None, 32, 32, 16)	0
conv2d_2 (Conv2D)	(None, 32, 32, 32)	4640
conv2d_3 (Conv2D)	(None, 32, 32, 32)	9248
max_pooling2d_1 (MaxPooling2D)	(None, 16, 16, 32)	0
conv2d_4 (Conv2D)	(None, 16, 16, 64)	18496
conv2d_5 (Conv2D)	(None, 16, 16, 64)	36928
max_pooling2d_2 (MaxPooling2D)	(None, 8, 8, 64)	0
flatten (Flatten)	(None, 4096)	0
dense (Dense)	(None, 512)	2097664
dense_1 (Dense)	(None, 256)	131328
dense_2 (Dense)	(None, 128)	32896
dense_3 (Dense)	(None, 4)	516
<hr/>		
Total params: 2334484 (8.91 MB)		
Trainable params: 2334484 (8.91 MB)		
Non-trainable params: 0 (0.00 Byte)		

Figura 4.18: Topología del modelo 4Figura 4.19: Resultados del entrenamiento del modelo 4

## Resumen de los entrenamientos de modelos

En primer lugar, es importante enfatizar, una vez más, que estos modelos son los modelos válidos obtenidos. Entre medias de la obtención de estos modelos existen numerosos entrenamientos cambiando parámetros y topología. Esto es debido a que el *Deep Learning* es una ciencia experimental y requiere de mucha experimentación para obtener un modelo válido.

La Tabla 4.8 presenta un resumen con los hiperparámetros y resultados de entrenamiento de todos los modelos.

Tabla 4.8: [Resumen de hiperparámetros y entrenamiento de todos los modelos](#)

Modelo	Learn. Rate	Epochs	Batch Size	Train. Param.	Val. Accuracy
1	0.001	50	32	2171796	97.65 %
2	0.001	50	32	8463252	96.63 %
3	0.0005	50	32	8625940	96.50 %
4	0.001	50	32	2334484	96.88 %

## 5. Evaluación

El objetivo de este capítulo es evaluar los resultados obtenidos y compararlos con lo que se propuso al principio de cada fase. Por ello, es importante resaltar el contraste entre la Figura 4.1 y la Figura 5.1 que se muestra a continuación, siendo la primera un esquema del sistema deseado al inicio del trabajo y, siendo la última el esquema de lo conseguido.

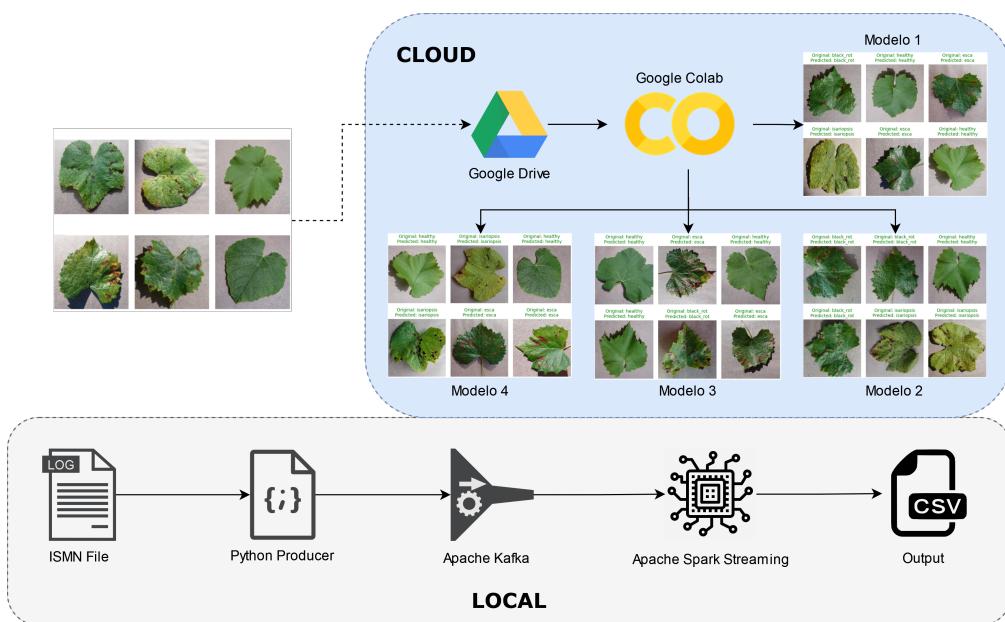


Figura 5.1: Boceto de soluciones alcanzadas

En la Figura 5.1 puede verse como se han obtenido dos soluciones independientes. En la parte de abajo de la Figura 5.1 puede verse el boceto de la solución de procesamiento de datos de calidad de suelo en *streaming* mientras que, en la parte superior de la Figura 5.1 puede contemplarse la solución de detección de enfermedades de cultivos a través de imágenes.

En el caso del procesamiento de datos en tiempo real, se ve como no se han usado sensores IoT reales, sustituyéndolos *scripts* en Python que leen información del *ISMN* y la envían a Kafka. Finalmente, Spark procesa los datos ingestados por Kafka y saca una salida en formato CSV.

Para el caso de la detección de enfermedades de cultivos a través de imágenes, se contempla como se han guardado las imágenes de hojas de vid en Google Drive. Posteriormente se monta el directorio que contiene dichas imágenes en Google Co-

lab y se entrenan cuatro modelos distintos.

A continuación y, puesto que se tratan de soluciones independientes, se procede a evaluar cada fase del trabajo por separado.

## 5.1. Procesamiento *streaming* de datos de calidad de suelo

En primer lugar, es necesario recordar los requisitos definidos al inicio de la fase 1 (Apartado 4.1): utilización de datos de calidad de suelo, salida en formato CSV y procesamiento en *streaming*.

Tal y como se detalló en la iteración 3 del Apartado 4.1, los datos que se han utilizado son datos de estaciones del *ISMN*. Dichas estaciones están equipadas con varios tipos de sensores capaces de medir características del suelo. Teniendo esto en cuenta, se puede concluir que el requisito REQ1 está cumplido.

Respecto al tipo de procesamiento que se lleva a cabo, en la iteración 2 del Apartado 4.1 se explica que se decidió utilizar Apache Spark *Streaming* pese a ser un motor de procesamiento que trabaja mediante micro-lotes. Sin embargo, aunque Spark *Streaming* utilice micro-lotes para procesar los datos, consigue llevar a cabo un procesamiento en *streaming* gracias a, entre otras cosas, trabajar con los datos en memoria. Además Lopez et al. (2016) demostraron que Spark *Streaming* es muy robusto frente a caídas de nodo. En este contexto, puede afirmarse que el requisito REQ2 se cumple incluso utilizando Apache Spark *Streaming*.

Respecto al formato de salida, en la iteración 4 del Apartado 4.1 se menciona cómo, en una reunión de con Judith y Saúl se acuerda el formato de salida es CSV. Además, en el Algoritmo 2 se ve como se configura la salida del consumidor para que se guarde en CSV. Por tanto, es evidente que el requisito REQ3 también se cumple.

Llegados a este punto, es evidente que, pese a haber claras diferencias entre la Figura 4.1 del Capítulo 4 y la Figura 5.1, se cumplen todos los requisitos definidos en el Apartado 4.1.

Por último, la Figura 5.2 muestra los datos de entrada del sistema de procesamiento de datos de calidad de suelo y, la Figura 5.3 muestra la salida que el consumidor escribe en formato CSV.

	measurement_id	station	sensor	model	depth	timestamp	value	quality
1	XMS-CAT	XMS-CAT	Clarella		42.11946			
				2.28235	974.0	0.05000	0.05000	CS655
2	2023/01/01	00:00	0.11	G M				
3	2023/01/01	01:00	0.11	G M				
4	2023/01/01	02:00	0.109	G M				
5	2023/01/01	03:00	0.109	G M				
6	2023/01/01	04:00	0.109	G M				
7	2023/01/01	05:00	0.11	G M				

Figura 5.2: Datos enviados por el productor

	measurement_datetime	station	sensor	model	depth_from	measurement
1	2023/01/01 00:00	Clarella	CS655	0.05	0.11	
2	2023/01/01 01:00	Clarella	CS655	0.05	0.11	
3	2023/01/01 02:00	Clarella	CS655	0.05	0.109	
4	2023/01/01 03:00	Clarella	CS655	0.05	0.109	
5	2023/01/01 04:00	Clarella	CS655	0.05	0.109	
6	2023/01/01 05:00	Clarella	CS655	0.05	0.11	

Figura 5.3: Salida escrita por el consumidor

## 5.2. Detección de enfermedades de cultivos a partir de imágenes

Antes de pasar a analizar el cumplimiento de los los requisitos definidos al inicio de la fase 2 (Apartado 4.2), a continuación se evalúa cada modelo por separado en base a la precisión que se obtiene con dichos modelos al trabajar con los datos de test.

Tal y como se detalló en la iteración 3 del Apartado 4.2, se designó el 10 % del total de los datos como datos de prueba. Para calcular la precisión con los datos de test, se utilizó la función `accuracy_score`<sup>1</sup> de Scikit-Learn.

Dicha función compara las etiquetas reales con las etiquetas predichas y calcula la proporción de predicciones correctas sobre el total de predicciones realizadas. Esta proporción se expresa como un valor decimal entre 0 y 1, donde 1 representa una precisión perfecta (todas las predicciones son correctas) y 0 representa una precisión nula (ninguna predicción es correcta).

A continuación se pasará a evaluar cada modelo por separado en base a la predicción obtenida con los datos de test. También se mostrarán 10 predicciones aleatorias de cada uno de los modelos.

<sup>1</sup>[https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy\\_score.html](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html)

## Evaluación del modelo 1

La Figura 5.4, muestra la evaluación del modelo 1. Como se puede ver, el modelo 1 obtiene un 99.75 % de precisión al trabajar con los datos de test, fallando tan sólo una predicción de 400.

```
13/13 [=====] - 2s 152ms/step
Correct predictions: 399 out of 400 possible
Accuracy with test data: 0.9975
```

Figura 5.4: Precisión del modelo 1 con los datos de test

Por último, la Figura 5.5 muestra 10 predicciones aleatorias realizadas con el modelo 1 comparándolas con la etiqueta original.

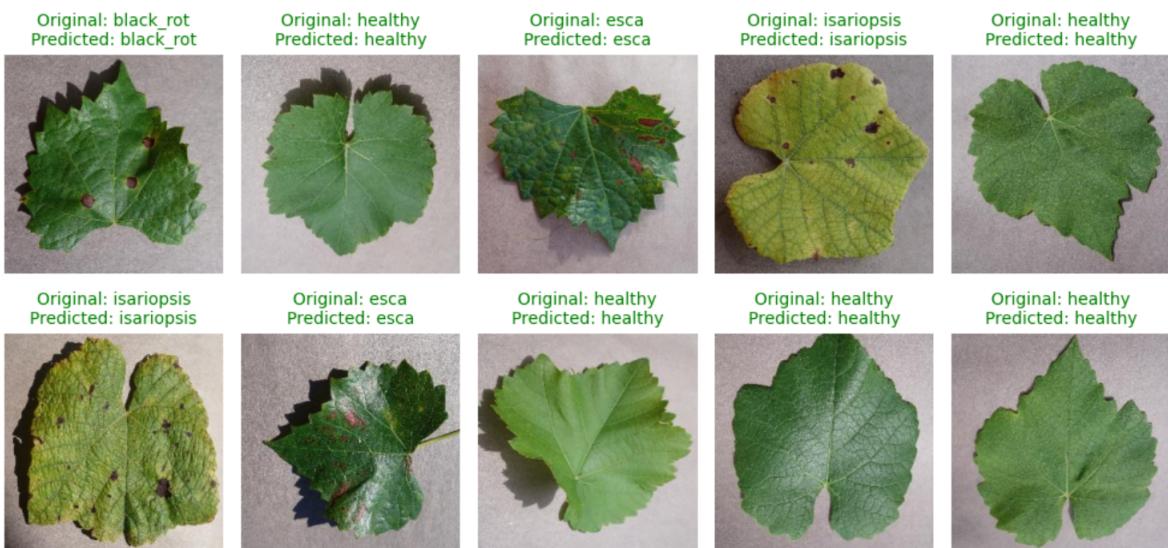


Figura 5.5: 10 predicciones aleatorias hechas con el modelo 1

## Evaluación del modelo 2

La Figura 5.6, muestra la evaluación del modelo 2. Como se puede ver, el modelo 2 obtiene un 97.5 % de precisión al trabajar con los datos de test, fallando 10 predicciones de 400.

Por último, la Figura 5.7 muestra 10 predicciones aleatorias realizadas con el modelo 2 comparándolas con la etiqueta original.

```
13/13 [=====] - 4s 314ms/step
Correct predictions: 390 out of 400 possible
Accuracy with test data: 0.975
```

Figura 5.6: Precisión del modelo 2 con los datos de test

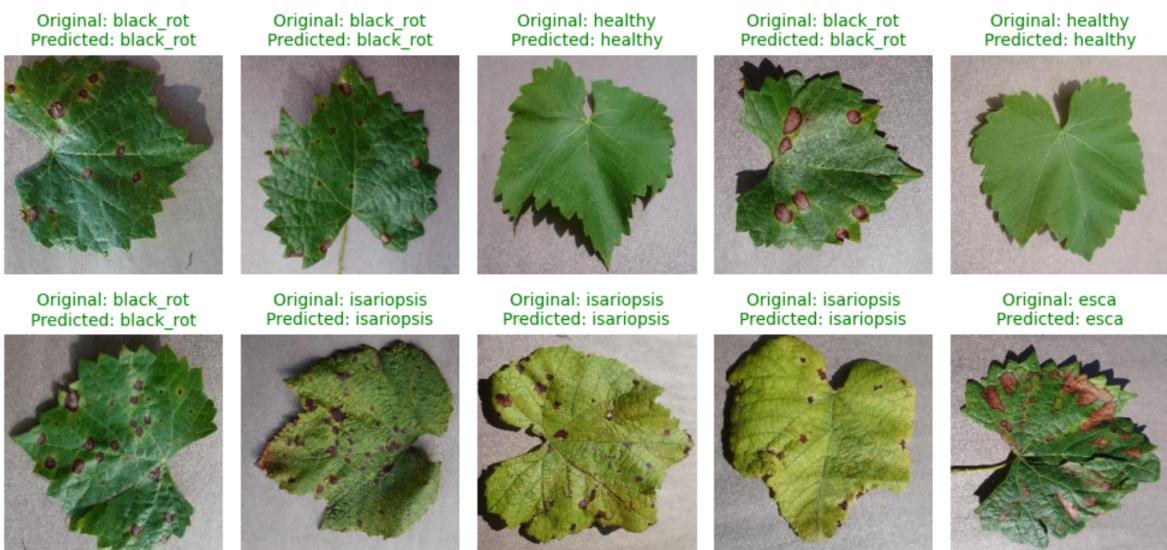


Figura 5.7: 10 predicciones aleatorias hechas con el modelo 1

### Evaluación del modelo 3

La Figura 5.8, muestra la evaluación del modelo 3. Como se puede ver, el modelo 3 obtiene un 97 % de precisión al trabajar con los datos de test, fallando 12 predicciones de 400.

```
13/13 [=====] - 4s 306ms/step
Correct predictions: 388 out of 400 possible
Accuracy with test data: 0.97
```

Figura 5.8: Precisión del modelo 3 con los datos de test

En este punto, es necesario remarcar la sorpresa de que este modelo, siendo el modelo con más parámetros entrenables debido a la dimensionalidad de entrada de 128x128 y las Capas Densas decrecientes adicionales, sea el modelo con el que menos precisión con los datos de prueba obtenga.

Por último, la Figura 5.9 muestra 10 predicciones aleatorias realizadas con el modelo 3 comparándolas con la etiqueta original.

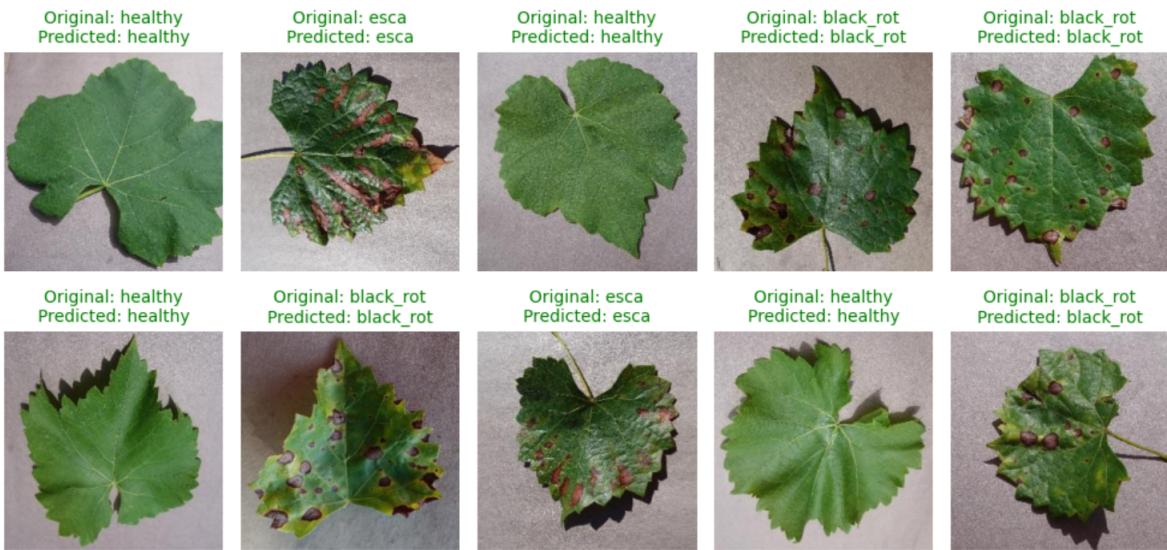


Figura 5.9: 10 predicciones aleatorias hechas con el modelo 3

## Evaluación del modelo 4

La Figura 5.10, muestra la evaluación del modelo 4. Como se puede ver, el modelo 4 obtiene un 98.25 % de precisión al trabajar con los datos de test, fallando 7 predicciones de 400.

```
13/13 [=====] - 1s 79ms/step
Correct predictions: 393 out of 400 possible
Accuracy with test data: 0.9825
```

Figura 5.10: Precisión del modelo 4 con los datos de test

Por último, la Figura 5.11 muestra 10 predicciones aleatorias realizadas con el modelo 4 comparándolas con la etiqueta original.

Como se puede comprobar, este modelo junto con el primero, ambos con una dimensionalidad de entrada de 64x64 píxeles, son los que mejores resultados han dado.

A continuación, en el siguiente apartado, se elaborará una tabla resumen en la que se mostrarán los datos de las evaluaciones de todos los modelos.

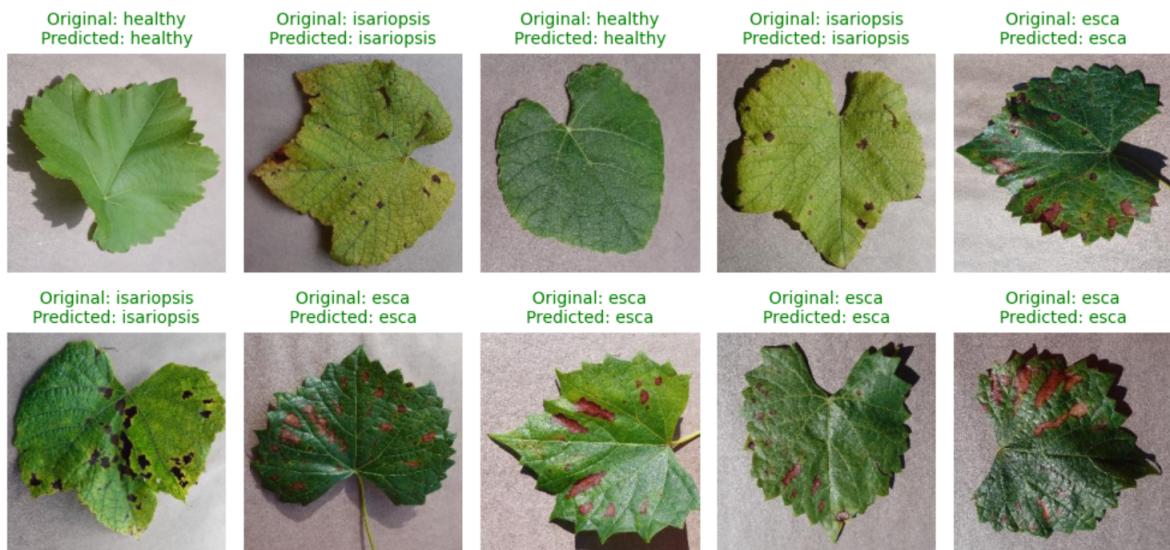


Figura 5.11: 10 predicciones aleatorias hechas con el modelo 4

## Resumen de los entrenamientos de modelos

La Tabla 5.1, compara los resultados obtenidos por los modelos con los datos de test frente a los parámetros entrenables de cada uno.

Tabla 5.1: Resumen de parámetros entrenables y precisión de los modelos

Modelo	Parámetros entrenables	% Precisión con datos de test
1	2171796	99.75
2	8463252	97.50
3	8625940	97.00
4	2334484	98.25

Tal y como se comentó al final del apartado anterior, el modelo 1 y el modelo 4, ambos con una dimensionalidad de entrada de 64x64 píxeles son los modelos con los que se han obtenido mejores resultados.

Los modelos generados con una dimensionalidad de entrada de 128x128 píxeles son igualmente buenos pero se pierde precisión con los datos de test y, además, el coste computacional es, aproximadamente, 4 veces mayor, tal y como evidencia la diferencia entre los parámetros entrenables.

Llegados a este punto, es evidente afirmar que, en caso de tener que utilizar uno de los modelos obtenidos en producción, el candidato perfecto sería el modelo 1 debido a su elevadísima precisión con los datos de test junto con su bajo coste computacional.

Por último, al igual que se remarcó previamente en la descripción de los resultados del modelo 2, es sorprendente que el modelo con más parámetros entrenables sea el modelo con menos precisión con datos de test.

## Cumplimiento de requisitos

Habiendo evaluado de manera independiente cada modelo, se procede a revisar el cumplimiento de los requisitos que se definieron al inicio de la fase 2 abordada en el Apartado 4.2.

En primer lugar, se puede afirmar que el REQ1 ha sido cumplido. Los datos de entrada utilizados han sido imágenes de cultivos. Esto se evidencia al elegir el conjunto de datos de trabajo en la primera iteración del Apartado 4.2.

Adicionalmente, el REQ2 también puede darse por cumplido. Esto queda demostrado al final de la iteración 1 del Apartado 4.2 cuando, entre varios cultivos, se selecciona la uva para llevar a cabo la detección de enfermedades. El cuaderno de Colab subido al GitHub<sup>2</sup> también demuestra que el REQ2 ha sido cumplido. De nuevo, para obtener más información acerca del código subido a GitHub, es necesario consultar el Apéndice A.

En base a lo enunciado en el REQ3, se puede decir que se han conseguido obtener cuatro modelos válidos, es decir, cuatro modelos cuya precisión con los datos de test es superior al 90 %. En este contexto, tanto el REQ3 como el REQ4 quedan cumplidos. Esto queda demostrado en la tabla 5.1, así como en el código de GitHub.

Llegados a este punto, se puede afirmar que se han cumplido todos los requisitos definidos al inicio de la fase 2. Sin embargo, a pesar de haber cumplido todos los objetivos, tanto en el caso de la fase 1 como en el caso de la fase 2, hay, todavía puntos a mejorar e interesantes objetivos adicionales que conseguir.

---

<sup>2</sup><https://github.com/jramon-mm/14MBID-TFM>

# 6. Conclusiones y trabajos futuros

En este trabajo se trata de plasmar como, a partir de la adopción de tecnologías avanzadas como el IoT, el *Big Data* o el *Deep Learning*, el sector agrícola puede llegar al siguiente nivel en términos de eficiencia y productividad, pudiendo, así, enfrentar futuras amenazas globales como la superpoblación y la seguridad alimenticia.

Con el objetivo de evidenciar los numerosos beneficios que las tecnologías anteriormente mencionadas pueden aportar a la industria agrícola, se ha llevado a cabo una aproximación a la agricultura inteligente basada, principalmente, en dos casos de uso: procesamiento de datos en *streaming* y, detección de enfermedades de cultivos a partir de imágenes.

Para el primer caso de uso se han analizado diversas tecnologías *Big Data* y, finalmente se han elegido Apache Kafka y Apache Spark *Streaming* para la implementación de la solución. Para el segundo caso de uso, se han utilizado modelos basados en Redes Neuronales Convolucionales, es decir, tecnologías de *Deep Learning*. En ambos casos, el lenguaje utilizado ha sido Python.

En el caso del sistema de procesamiento en tiempo real, al no disponer de sensores IoT reales, se han tenido que simular dichos sensores mediante *scripts* Python. Las mediciones que los sensores simulados han enviado se han sacado de sensores reales equipados en estaciones del *ISMN*<sup>1</sup>. Para el caso de los modelos de detección de enfermedades de cultivos a través de imágenes, se han utilizado las imágenes de uno de los cultivos contenidos en un conjunto de datos de Kaggle<sup>2</sup>.

El sistema de procesamiento de datos en tiempo real se ha evaluado en base al cumplimiento de los requisitos planteados al inicio del desarrollo de dicho sistema. En cambio, los modelos de detección de enfermedades de cultivos a través de imágenes, se han evaluado en base a la precisión que estos han obtenido al trabajar con los datos de test. Adicionalmente, es necesario mencionar que también se han cumplido los requisitos iniciales planteados para este último caso de uso.

## 6.1. Trabajos futuros

A pesar de los buenos resultados que se han obtenido durante el desarrollo de este trabajo, existen varios aspectos pendientes de ser pulidos para mejorar, así como la calidad de las soluciones implementadas.

<sup>1</sup><https://ismn.earth/en/>

<sup>2</sup><https://www.kaggle.com/>

## Subida del sistema al *Cloud*

La solución de procesamiento propuesta se ha implementado en un entorno Linux local. Tal y como se mencionó en el Apartado 2.2 el *Cloud Computing* es una de las tecnologías clave en la agricultura inteligente debido a que, normalmente las explotaciones agrícolas suelen estar en zonas rurales donde es difícil montar la infraestructura necesaria para ejecutar este tipo de sistemas.

Montar el sistema de procesamiento en *streaming* en la infraestructura de algún proveedor *Cloud* no solo dotaría de más fidelidad para con el *smart farming* a la solución obtenida en este TFM, sino que también la haría más flexible, segura y fiable.

Aumentar o reducir recursos computacionales en función de la carga del sistema, sistemas de replicación y mecanismos de autenticación son algunas de las funcionalidades extra que puede aportar el *Cloud* al sistema de procesamiento en *streaming* existente.

Respecto a la solución de detección de enfermedades de cultivos a partir de imágenes, se podría decir que está en la nube porque están las imágenes subidas a Google Drive pero, se podría desplegar de manera más profesional en cualquier proveedor *Cloud* que se deseé.

De esta manera, sería más fácil la integración de la solución de procesamiento y la de detección de enfermedades, es decir, sería más sencillo obtener lo planteado en la Figura 4.1.

## Uso de sensores IoT reales para la recogida de datos

Tal y como se ha comentado en el Apartado 4.1.3, al no contar con sensores IoT reales se han usado *scripts* de Python que envían datos de mediciones reales tomadas por sensores de suelo equipados en las estaciones del *ISMN*.

Uno de los trabajos pendientes, por tanto es el uso de sensores IoT reales para la recolección de los datos que alimentan el sistema de procesamiento. Llevaría el sistema de procesamiento al siguiente nivel en términos de fidelidad con un caso de uso real.

## Aumentar la cantidad de datos de entrada para el sistema de procesamiento

Dado el carácter acotado de un TFM y tal y como se detalló en el Apartado 1.1.1, el objetivo es implementar un prototipo que implemente una aproximación a la agricultura inteligente.

En este contexto, los datos que se han usado en el procesamiento de datos en tiempo real son de, como mucho, tres o cuatro sensores y, como máximo, dos estaciones distintas. Esto es un enfoque correcto para un prototipo pero no de cara a un sistema de procesamiento real.

Por ello, poner a prueba el sistema de procesamiento *streaming* aumentando enormemente los datos de entrada, es un trabajo sería interesante hacer para conocer los límites del sistema.

## Aumentar los cultivos a los que se pueda detectar enfermedades

Tal y como se menciona al final del Apartado 4.2.1, existían varios cultivos en el conjunto de datos elegido y se decidió centrar los esfuerzos en la uva. Llegados a este punto, un trabajo futuro interesante sería conseguir un modelo que sea capaz de detectar enfermedades de varios cultivos, no solo uno.

Para ello, en vez de volver a entrenar un modelo en el que se incluyan las categorías que ya se han tenido en cuenta en este TFM junto con las nuevas categorías, sería interesante hacer un modelo que sea capaz de identificar los cultivos de los cuales se quieran detectar las enfermedades y, posteriormente, entrenar un nuevo modelo de detección de enfermedades por cada cultivo.

El resultado sería un sistema de detección de cultivos para identificar el modelo de detección de enfermedades a utilizar. De esta manera se aprovecharía el trabajo llevado a cabo en este TFM a la vez que se escala progresivamente añadiendo los nuevos modelos de cada cultivo cuyas enfermedades se quieran detectar.

## Integración con otros TFMs

Como ya he comentado en la iteración 4 del Apartado 4.1, Saúl también está llevando a cabo un TFM dentro del contexto de UbiCDatos.

Llegados a este punto, puede ser buena tratar de integrar trabajos que comparten contexto, independientemente del máster o grado al que pertenezcan y, obviamente, siempre que los autores estén de acuerdo. Algunos trabajos candidatos podrían ser (González-Domínguez, 2023) o (Hernández-Pérez, 2023).

En definitiva, la unión de los esfuerzos parciales llevados a cabo en este tipo de proyectos puede ayudar a construir una solución mayor y mejor.

# Bibliografía

- Aggarwal, K., Mijwil, M. M., Sonia, Al-Mistarehi, A.-H., Alomari, S., Gök, M., Alaabdin, A. M. Z., and Abdulrhman, S. H. (2022). Has the future started? the current growth of artificial intelligence, machine learning, and deep learning. *Iraqi Journal For Computer Science and Mathematics*, 3(1):115123.
- Alfred, R., Obit, J. H., Chin, C. P.-Y., Habiluddin, H., and Lim, Y. (2021). Towards paddy rice smart farming: A review on big data, machine learning, and rice production tasks. *IEEE Access*, 9:50358–50380.
- Almalki, F. A., Soufiene, B. O., Alsamhi, S. H., and Sakli, H. (2021). A low-cost platform for environmental smart farming monitoring system based on iot and uavs. *Sustainability*, 13(11):5908.
- Alwidian, J., Rahman, S. A., Gnaim, M., Al-Taharwah, F., et al. (2020). Big data ingestion and preparation tools. *Modern Applied Science*, 14(9):12–27.
- Araújo, S. O., Peres, R. S., Barata, J., Lidon, F., and Ramalho, J. C. (2021). Characterising the agriculture 4.0 landscapeemerging trends, challenges and opportunities. *Agronomy*, 11(4):667.
- Channe, H., Kothari, S., and Kadam, D. (2015). Multidisciplinary model for smart agriculture using internet-of-things (iot), sensors, cloud-computing, mobile-computing & big-data analysis. *Int. J. Computer Technology & Applications*, 6(3):374–382.
- Chauhan, N. K. and Singh, K. (2018). A review on conventional machine learning vs deep learning. In *2018 International Conference on Computing, Power and Communication Technologies (GUCON)*, pages 347–352.
- Chergui, N. and Kechadi, M. T. (2022). Data analytics for crop management: a big data view. *Journal of Big Data*, 9(1):1–37.
- Cravero, A., Pardo, S., Sepúlveda, S., and Muñoz, L. (2022). Challenges to use machine learning in agricultural big data: A systematic literature review. *Agronomy*, 12(3):748.
- Dean, J. and Ghemawat, S. (2008). Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107113.
- Dorigo, W. A., Xaver, A., Vreugdenhil, M., Gruber, A., Hegyiová, A., Sanchis-Dufau, A. D., Zamojski, D., Cordes, C., Wagner, W., and Drusch, M. (2013). Global automated quality control of in situ soil moisture data from the international soil moisture network. *Vadose Zone Journal*, 12(3).
- Farzan, A. and Nicolet, V. (2021). Phased synthesis of divide and conquer programs. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, PLDI 2021, page 974986, New York, NY, USA. Association for Computing Machinery.
- Food and Agriculture Organization of the United Nations (2017). *The Future of Food and Agriculture: Trends and Challenges*. Food & Agriculture Organization of the United Nations (FAO).
- González-Domínguez, M. (2023). Diseño y desarrollo de un sistema iot de riego

- telemático escalable. Trabajo Final de Grado. Tutora PhD Judith Cardinale.
- Hernández-Pérez, A. (2023). Desarrollo de una aplicación para la visualización de datos provenientes de un sistema iot de monitoreo de siembras y el control de un sistema de riego. Trabajo Final de Máster. Tutora PhD Judith Cardinale.
- Jais, I. K. M., Ismail, A. R., and Nisa, S. Q. (2019). Adam optimization algorithm for wide and deep neural network. *Knowledge Engineering and Data Science*, 2(1):41–46.
- Joshi, A. V. (2020). *Introduction to AI and ML*, page 4. Springer International Publishing, Cham.
- Kumar, S., Chowdhary, G., Udutoalapally, V., Das, D., and Mohanty, S. P. (2019). gcrop: Internet-of-leaf-things (iolt) for monitoring of the growth of crops in smart agriculture. In *2019 IEEE International Symposium on Smart Electronic Systems (iSES) (Formerly iNiS)*, pages 53–56.
- Lee, K.-H., Lee, Y.-J., Choi, H., Chung, Y. D., and Moon, B. (2012). Parallel data processing with mapreduce: A survey. *SIGMOD Rec.*, 40(4):1120.
- Leijnen, S. and van Veen, F. (2020). The neural network zoo. *Proceedings*.
- Lopez, M. A., Lobato, A. G. P., and Duarte, O. C. M. B. (2016). A performance comparison of open-source stream processing platforms. In *2016 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6.
- Luis, ., Casares, P., Cuadrado-Gallego, J. J., and Patricio, M. A. (2021). Pson: A serialization format for iot sensor networks. *Sensors*, 21(13):4559.
- Mahesh, B. (2019). Machine learning algorithms - a review.
- Mahi, S. S. (2022). Plant disease expert.
- Mătăcuță, A. and Popa, C. (2018). Big data analytics: Analysis of features and performance of big data ingestion tools. *Informatica Economica*, 22(2).
- Mohanty, S. P., Hughes, D. P., and Salathé, M. (2016). Using deep learning for image-based plant disease detection. *Frontiers in Plant Science*, 7.
- Moysiadis, V., Sarigiannidis, P., Vitsas, V., and Khelifi, A. (2021). Smart farming in europe. *Computer Science Review*, 39:100345.
- Nasiri, H., Nasehi, S., and Goudarzi, M. (2019). Evaluation of distributed stream processing frameworks for iot applications in smart cities. *Journal of Big Data*, 6:1–24.
- Ongsulee, P. (2017). Artificial intelligence, machine learning and deep learning. In *2017 15th International Conference on ICT and Knowledge Engineering (ICT&KE)*, pages 1–6.
- O’Shea, K. and Nash, R. (2015). An introduction to convolutional neural networks.
- Ouadi, L., Bruez, E., Bastien, S., Vallance, J., Lecomte, P., Domec, J.-C., and Rey, P. (2019). Ecophysiological impacts of esca, a devastating grapevine trunk disease, on vitis vinifera l. *PLOS ONE*, 14(9):1–20.
- Patel, K. K., Patel, S. M., and Scholar, P. (2016). Internet of things-iot: Definition, characteristics, architecture, enabling technologies, application & future challenges. *International Journal of Engineering Science and Computing*, 6(5).
- Quy, V. K., Hau, N. V., Anh, D. V., Quy, N. M., Ban, N. T., Lanza, S., Randazzo, G., and Muzirafuti, A. (2022). Iot-enabled smart agriculture: architecture, applications,

- and challenges. *Applied Sciences*, 12(7):3396.
- Sagiroglu, S. and Sinanc, D. (2013). Big data: A review. *2013 International Conference on Collaboration Technologies and Systems (CTS)*, pages 42–47.
- Saiz-Rubio, V. and Rovira-Más, F. (2020). From smart farming towards agriculture 5.0: A review on crop data management. *Agronomy*, 10(2):207.
- Sector de normalización de las telecomunicaciones de la UIT (2012). Y.4000/Y.2060: Overview of the Internet of things. Recomendación UIT-T Y.4000/Y.2060, Unión Internacional de Telecomunicaciones.
- Sharma, G., Tripathi, V., and Srivastava, A. (2021). Recent trends in big data ingestion tools: A study. In Kumar, R., Quang, N. H., Kumar Solanki, V., Cardona, M., and Pattnaik, P. K., editors, *Research in Intelligent and Computing in Engineering*, pages 873–881, Singapore. Springer Singapore.
- Sharma, S., Sharma, S., and Athaiya, A. (2017). Activation functions in neural networks. *Towards Data Sci*, 6(12):310–316.
- Sharma, V., Rai, S., and Dev, A. (2012). A comprehensive study of artificial neural networks.
- Shinde, P. P. and Shah, S. (2018). A review of machine learning and deep learning applications. In *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, pages 1–6.
- Sinha, B. B. and Dhanalakshmi, R. (2022). Recent advancements and challenges of internet of things in smart agriculture: A survey. *Future Generation Computer Systems*, 126:169–184.
- Soumaya, O., Amine, T. M., Soufiane, A., Abderrahmane, D., and Mohamed, A. (2017). Real-time data stream processing challenges and perspectives. *International Journal of Computer Science Issues (IJCSI)*, 14(5):6–12.
- Szabó, M., Csikász-Krizzics, A., Dula, T., Farkas, E., Roznik, D., Kozma, P., and Deák, T. (2023). Black rot of grapes (*guignardia bidwellii*)a comprehensive overview. *Horticulturae*, 9(2):130.
- Tao, Y., Lin, W., and Xiao, X. (2013). Minimal mapreduce algorithms. SIGMOD '13, page 529540, New York, NY, USA. Association for Computing Machinery.
- United Nations Department of Economic and Social Affairs, Population Division (2022). World Population Prospects 2022: Summary of Results. Technical Report UN DESA/POP/2022/TR/NO. 3.
- Vassakis, K., Petrakis, E., and Kopanakis, I. (2018). Big data analytics: Applications, prospects and challenges. In Skourletopoulos, G., Mastorakis, G., Mavromoustakis, C., Dobre, C., and Pallis, E., editors, *Mobile Big Data*, chapter 1. Springer.
- Wang, L., Xu, L., Zheng, Z., Liu, S., Li, X., Cao, L., Li, J., and Sun, C. (2021). Smart contract-based agricultural food supply chain traceability. *IEEE Access*, pages 9296–9307.
- Welter, A., Nava, G., and Paulus, D. (2019). Severidade da mancha-das-folhas (*isariopsis clavigpora*) em videiras cultivadas em clima subtropical. *Brazilian Journal of Development*, 5:26264–26273.
- Wolfert, S., Ge, L., Verdouw, C., and Bogaardt, M.-J. (2017). Big data in smart farming a review. *Agricultural Systems*, 153:69–80.

Ünal, Z. (2020). Smart farming becomes even smarter with deep learninga bibliographical analysis. *IEEE Access*, 8:105587–105609.

## A. Apéndice: Repositorio del trabajo

El código completo de las dos soluciones implementadas en este TFM se encuentra publicado en GitHub<sup>1</sup>.

Puesto que el trabajo implementa dos soluciones por separado, el repositorio de GitHub también está dividido en dos soluciones. Esta división se lleva a cabo mediante carpetas.

- **IOT-Sensors:** En esta carpeta se encontrará el código correspondiente a la ingestión y procesamiento en streaming de los datos de los sensores IOT. Los ficheros de esta carpeta son:

- kafka\_clarella\_consumer.py: Consume los datos de Kafka, aplica las transformaciones y guarda la salida en CSV.
- xms\_cat\_clarella\_sm\_005.py: Lee los datos de un fichero del ISMN, extrae la información y la manda a Kafka.
- kafka\_commands.txt: Fichero de texto con comandos útiles como inicio del cluster de Kafka, creación de topics o ejecución del consumidor de Kafka.

- **Crop-Disease-Detection:** En esta carpeta se encontrará el cuaderno que implementa la solución de detección de enfermedades a través de imágenes.

---

<sup>1</sup><https://github.com/jramon-mm/14MBID-TFM>