



**UNIVERSIDAD DE CASTILLA-LA MANCHA**  
**ESCUELA SUPERIOR DE INFORMÁTICA**

**GRADO EN INGENIERÍA INFORMÁTICA**

**Ciencias de la Computación**

**TRABAJO FIN DE GRADO**

Análisis de arquitecturas de aprendizaje profundo para la  
detección de incendios mediante imágenes

Mauro García Monclú

julio, 2020





**UNIVERSIDAD DE CASTILLA-LA MANCHA  
ESCUELA SUPERIOR DE INFORMÁTICA**

**Departamento Tecnologías y Sistemas de Información**

**Ciencias de la Computación**

**TRABAJO FIN DE GRADO**

**Análisis de arquitecturas de aprendizaje profundo para  
la detección de incendios mediante imágenes**

Autor: Mauro García Monclú

Tutor: Francisco Pascual Romero Chicharro

Co-Tutor: Fernando Vallejo Banegas

julio, 2020

Detección de incendios mediante imágenes

© Mauro García Monclú, 2020

[maurogarciamonclu@gmail.com](mailto:maurogarciamonclu@gmail.com)

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Se permite la copia, distribución y/o modificación de este documento bajo los términos de la Licencia de Documentación Libre GNU, versión 1.3 o cualquier versión posterior publicada por la Free Software Foundation; sin secciones invariantes. Una copia de esta licencia está incluida en el apéndice titulado «GNU Free Documentation License».

Muchos de los nombres usados por las compañías para diferenciar sus productos y servicios son reclamados como marcas registradas. Allí donde estos nombres aparezcan en este documento, y cuando el autor haya sido informado de esas marcas registradas, los nombres estarán escritos en mayúsculas o como nombres propios.

**TRIBUNAL:**

Presidente: \_\_\_\_\_

Vocal: \_\_\_\_\_

Secretario: \_\_\_\_\_

**FECHA DE DEFENSA:** \_\_\_\_\_

**CALIFICACIÓN:** \_\_\_\_\_

PRESIDENTE

VOCAL

SECRETARIO

Fdo.:

Fdo.:

Fdo.:



## **Resumen**

El calentamiento global provocado trae multitud desgracias, una de ellas es la sequía de zonas húmedas. La sequía aumentará la probabilidad de incendios y, de esta manera, provocará la desertificación de los mismos. Ya sea por el calentamiento global o provocado por el humano los incendios forestales son un problema a la orden del día.

La inteligencia artificial avanza a pasos agigantados y no debemos quedarnos solo en la teoría, debemos ponerla en práctica. Este proyecto pretende utilizar los últimos avances en reconocimiento de imágenes, junto al desarrollo de computadores autónomos, para la detección de incendios antes de que se ganen una portada en las noticias.

El resultado consistirá en un algoritmo capaz de detectar humo en imágenes capturadas por cámaras instaladas en una computadora de bajos recursos. Durante el desarrollo, se analizará qué técnicas de *deep learning* serán las más adecuadas para la clasificación y segmentación de imágenes como objetivo de humo.



## **Abstract**

The global warming caused brings many misfortunes, one of them is the drought of wetlands. Drought will increase the probability of fires and, in this way, will cause their desertification. Forest fire from global warming or human-caused are a problem on the order of the day.

Artificial intelligence is improving by leaps and bounds and we must not just stay in theory, we must put it into practice. This project pretends to use the latest developments in image recognition and automaton computers for fire detection before they make a title page on the news.

The result is an algorithm capable of detecting smoke in images captured by cameras installed on a low-resource computer. During development, deep learning techniques will be analyzed to know which techniques are the most optimal for the classification and segmentation of images as smoke target.



# AGRADECIMIENTOS

---

Aquí se cierra una etapa. Una etapa de buenos y malos momentos, años de aprendizaje que me servirán en el futuro. Una etapa en la que he estado solo y lejos de mi familia, aunque realmente siempre he estado acompañado de amigos con los cuales he compartido esta afición.

Gracias a mis padres por darme la oportunidad de formarme en lo que quiero, darme libertad y aconsejarme para tomar buenas decisiones, a mi hermana por sacarme siempre una sonrisa y a Alexandra por aguantar mi humor, por estar siempre a mi lado en los buenos momentos y, con más fuerza, en los malos.

Agradecer a mi tutor, Francisco Pascual, por resolver mis múltiples dudas y no privarse de charlar un domingo. Agradecer que haya aceptado ser mi tutor del proyecto, pero además de permitir entablar una relación más allá de alumno-profesor.

Agradecer también a Indra. Aunque haya sido una relación profesional, he tenido el placer de conocer a fantásticos compañeros. Sobre todo, a Fernando y Dionisio que no dudaron en acogerme con los brazos abiertos, tanto profesionalmente como personalmente.

Esto no se acaba aquí, es hora de coger la llave, que he recibido gracias a las capacidades adquiridas, para abrir nuevas puertas.

*Mauro García Monclú*  
Ciudad Real, 2020



# NOTACIÓN

---

TFG	:	Trabajo Fin de Grado.
AI	:	Inteligencia artificial (Artificial Intelligence).
Machine learning	:	Aprendizaje automático.
Deep learning	:	Aprendizaje profundo.
FORTE	:	FORTalecimiento de las competencias profesionales de los graduados para la mejora de su Empleabilidad.
CO <sub>2</sub>	:	Dióxido de carbono
LoRa	:	De largo alcance (Long Range)
IoT	:	Internet de las cosas (Internet of Things)
Hats	:	Módulos para Raspberry Pi
API REST	:	Interfaz de programación de aplicaciones para transferencia de estado representacional (Application Programming Interface Representational State Transfer)
Dataset	:	Conjunto de datos
RGB	:	Rojo, verde y azul (Red, Green, Blue)
Script	:	Secuencia de comandos
Pooling	:	Agrupar
ILSVRC	:	ImageNet Large Scale Visual Recognition Challenge
Shortcut connections	:	Conexiones de acceso directo
ResNet	:	Residual Neural Network
Transpose	:	Transponer
Upsampling	:	Muestreo ascendente
Data augmentation	:	Aumentación de datos
Open-source	:	Código abierto
CPU	:	Unidad central de procesamiento (Central Processing Unit)
GPU	:	Unidad de procesamiento gráfico (Graphics Processing Unit)
TPU	:	Unidad de procesamiento tensorial (Tensor Processing Unit)
RAM	:	Memoria de acceso aleatorio (Random-Access Memory)
VRAM	:	Memoria gráfica de acceso aleatorio (Video Random-Access Memory)
USB	:	Bus universal en serie (Universal Serial Bus)
Learning rate	:	Tasa de aprendizaje
Host	:	Anfitrión
Backbone	:	Vértebra
Shell	:	Interfaz de usuario
microSD	:	micro Secure Digital card



# ÍNDICE GENERAL

---

<b>Resumen</b>	<b>vii</b>
<b>Abstract</b>	<b>ix</b>
<b>Agradecimientos</b>	<b>xI</b>
<b>Notación</b>	<b>xIII</b>
<b>Índice de figuras</b>	<b>xVII</b>
<b>Índice de tablas</b>	<b>xIX</b>
<b>Índice de listados</b>	<b>xxI</b>
<b>1. Introducción</b>	<b>1</b>
1.1. Motivación . . . . .	1
1.2. Presentación del problema . . . . .	2
1.3. Estructura del documento . . . . .	3
<b>Índice de algoritmos</b>	<b>1</b>
<b>2. Objetivo</b>	<b>5</b>
2.1. Objetivo general . . . . .	5
2.2. Objetivos parciales . . . . .	5
<b>3. Antecedentes</b>	<b>7</b>
3.1. Machine Learning y Deep Learning . . . . .	7
3.2. Redes neuronales . . . . .	7
3.2.1. Modelos para clasificación de imágenes . . . . .	8
3.2.2. Modelos para segmentación de imágenes . . . . .	13
3.3. Data Augmentation . . . . .	14
3.4. IoT . . . . .	15
3.4.1. Raspberry Pi . . . . .	15
<b>4. Metodología</b>	<b>17</b>
4.1. Metodología de desarrollo del proyecto . . . . .	17
4.1.1. Método de trabajo . . . . .	17
4.2. Metodología de Gestión del Proyecto . . . . .	18
4.3. Planificación . . . . .	19
4.4. Marco Tecnológico . . . . .	21
4.4.1. Librerías . . . . .	21
4.4.2. Herramientas de Soporte . . . . .	22

4.4.3. Hardware . . . . .	23
<b>5. Resultados</b>	<b>25</b>
5.1. Iteración 1 y 4: Dataset . . . . .	25
5.2. Iteración 2: Clasificación de imágenes . . . . .	30
5.2.1. Entrenamientos . . . . .	30
5.2.2. Conclusión . . . . .	33
5.3. Iteración 3: API REST para clasificación . . . . .	34
5.4. Iteración 5: Segmentación de imágenes . . . . .	35
5.4.1. Entrenamientos . . . . .	35
5.4.2. Muestra de resultados . . . . .	42
5.5. Iteración 6: API REST para segmentación . . . . .	43
5.6. Iteración 7: Despliegue en Raspberry Pi . . . . .	44
5.7. Iteración 8: Despliegue en Aplicación Web . . . . .	47
5.7.1. Manual de usuario . . . . .	47
<b>6. Conclusiones</b>	<b>51</b>
6.1. Objetivos alcanzados . . . . .	51
6.2. Trabajos futuros . . . . .	52
6.3. Conclusión . . . . .	52
6.4. Competencias satisfechas . . . . .	52
<b>Bibliografía</b>	<b>55</b>
<b>A. Hats para Raspberry Pi</b>	<b>59</b>
A.1. Trasmisión . . . . .	59
A.2. Alimentación . . . . .	59
A.3. Carcasa . . . . .	61
A.4. Coste . . . . .	61

# ÍNDICE DE FIGURAS

---

1.1. Esquema de sensor de humo . . . . .	1
1.2. Incendio . . . . .	2
1.3. Torres de alta tensión . . . . .	3
3.1. Función Sigmoid . . . . .	9
3.2. Capa de convolución . . . . .	9
3.3. Capa de Maxpooling . . . . .	10
3.4. Gráfica de ganadores de ImageNet . . . . .	10
3.5. Bloque residual . . . . .	11
3.6. Bloque Bottleneck . . . . .	11
3.7. Bloque Inception . . . . .	12
3.8. Modelo EfficientNetB0 . . . . .	12
3.9. Capa de deconvolución . . . . .	13
3.10. Capa de Upsampling . . . . .	13
3.11. U-Net . . . . .	14
3.12. Data Augmentation . . . . .	15
3.13. Raspberry Pi . . . . .	15
4.1. Orden de importancia de parámetros . . . . .	17
4.2. Tablero Kanban . . . . .	18
4.3. Diagrama de Gantt . . . . .	20
5.1. Imágenes del Dataset . . . . .	26
5.2. Imágenes redimensionadas . . . . .	27
5.3. Imágenes de máscaras . . . . .	27
5.4. Imágenes para clasificación . . . . .	28
5.5. Imágenes para segmentación . . . . .	29
5.6. Imágenes con Data Augmentation . . . . .	31
5.7. Reducción del Learning Rate . . . . .	31
5.8. Gráficas entrenamiento ResNet . . . . .	31
5.9. Gráficas entrenamiento InceptionResNetV2 . . . . .	32
5.10. Gráficas entrenamiento EfficientNetB6 . . . . .	32
5.11. Gráficas validación para clasificación . . . . .	32
5.12. Resultados de cada modelo para clasificación . . . . .	33
5.13. Gráficas entrenamiento Backbone InceptionResNetV2 . . . . .	36
5.14. Gráficas entrenamiento Backbone EfficientNetB6 . . . . .	36
5.15. Gráficas validación para diferentes Backbones . . . . .	37
5.16. Resultados de cada modelo con diferentes Backbones . . . . .	37
5.17. Gráficas de validación con diferentes decodificadores . . . . .	38
5.18. Resultados de cada modelo con diferentes decodificadores . . . . .	39
5.19. Diferentes Data Augmentations . . . . .	40

5.20. Gráficas validación para diferentes Data Augmentations . . . . .	42
5.21. Pantalla principal aplicación web . . . . .	47
5.22. Interfaz predicción aplicación web . . . . .	48
A.1. Hat transmisor LoRa . . . . .	59
A.2. Hat gestor alimentación . . . . .	60
A.3. Batería para Raspberry Pi . . . . .	60
A.4. Panel solar para Raspberry Pi . . . . .	60

# ÍNDICE DE TABLAS

---

4.1. Hoja de ruta .....	20
4.2. Especificaciones de cada máquina utilizada .....	23
5.1. Datos técnicos del <i>dataset</i> .....	29
A.1. Coste .....	61



# ÍNDICE DE LISTADOS

---

5.1.	Código fuente de la predicción de una imagen en la API REST para clasificación . . . . .	34
5.2.	Código fuente de la generación de imágenes y máscara con <i>data augmentation</i> para su uso en entrenamiento de segmentación . . . . .	41
5.3.	Código fuente de la predicción de una imagen en la API REST para segmentación . . . . .	43
5.4.	Código fuente del instalador en Raspberry Pi . . . . .	44
5.5.	Código fuente del <i>script</i> para la detección de humo en Raspberry Pi . . . . .	45
5.6.	Código fuente del <i>script</i> para comprobar el correcto funcionamiento de la Raspberry Pi	46
5.7.	Código fuente de las funciones que representan cada dirección de la web . . . . .	49



---

# CAPÍTULO 1

# INTRODUCCIÓN

---

En este capítulo se dará una visión general del proyecto. Se introducirá su contexto general, una motivación que se pretenderá cumplir, un resumen y una explicación de la estructura de este documento.

Este proyecto se ha desarrollado dentro del programa FORTalecimiento de las competencias profesionales de los graduados para la mejora de su Empleabilidad (FORTE) de la Escuela Superior de Informática de la Universidad de Castilla-La Mancha. El objetivo fundamental del programa FORTE es permitir que los estudiantes que están próximos a la finalización de la carrera, se sumerjan en el desarrollo de algún proyecto real que se esté llevando a cabo dentro de una empresa y realizar el Trabajo de Fin de Grado (TFG) dentro de la misma.

Dentro de este contexto, el proyecto se ha realizado en la empresa Indra Sistemas, S.A. en su centro de desarrollo software en Ciudad Real dentro del departamento de Visión Artificial enfocado en la ciencia de datos. El grupo de trabajo con el que será desarrollado se centra en el tratamiento de imágenes desde una perspectiva científica.

## 1.1. MOTIVACIÓN

La detección de humos en entornos cerrados está bastante desarrollada hoy en día. Existen multitud de empresas dedicadas al desarrollo e investigación de sensores capaces de detectar acumulaciones de CO<sub>2</sub> o humo (ver Fig. 1.1<sup>1</sup>).

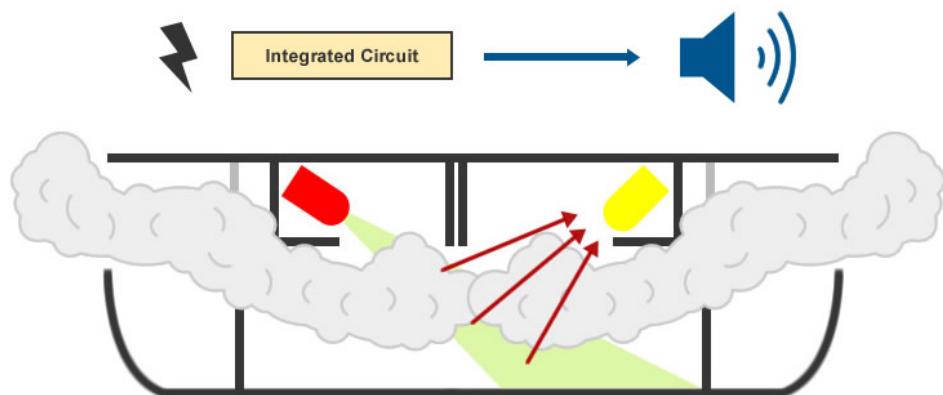


Figura 1.1: Esquema de cómo funciona un sensor de humo

En entornos cerrados el humo se dispersa por el espacio sin humo de la habitación por lo que esta tecnología es bastante eficiente. Ahora bien, en zonas abiertas es posible que el sensor se sitúe a escasos metros del fuego y sea incapaz de detectarlo ya sea por el viento o por la propia densidad del aire.

---

<sup>1</sup>[https://www.safelincs.co.uk/templates\\_core/templates/guides/images/smoke-alarms/optical-smoke-sensor.jpg](https://www.safelincs.co.uk/templates_core/templates/guides/images/smoke-alarms/optical-smoke-sensor.jpg)

Desgraciadamente, los incendios en entornos abiertos (ver Fig. 1.2<sup>2</sup>) son más devastadores que en entornos cerrados y es una problemática que actualmente sigue siendo difícil de afrontar.



**Figura 1.2:** Imagen de un incendio en Toledo (España)

Una posible opción puede ser la instalación de un número incalculable de estos sensores repartidos por la superficie con el fin de que sean capaces de detectar esos gases. Sin embargo, la instalación de todos esos sensores, el mantenimiento, el soporte a sus peticiones y el daño ambiental que supondría queda descartada para estos entornos incluso para el futuro.

Solo nos queda preguntarnos, ¿de qué forma se manifiesta un incendio? Este proyecto abordará la forma más evidente, de forma visual. No solo resolverá los problemas anteriormente citados, si no que deja la posibilidad de variar el objetivo visual que en este caso será el humo.

La principal motivación de este proyecto es la utilización de los últimos avances en el campo de la visión por computador para resolver este problema. Estos avances van desde el incremento de potencia de computación gráfica en sistemas embebidos hasta la investigación en modelos de redes neuronales (*deep learning*) para la clasificación y segmentación de imágenes .

A esto se añade que existe la demanda por parte de empresas de distribución eléctrica para la detección de incendios cerca de torres de alta tensión. Estas instalaciones son propensas a tener problemas de chispas y generar fácilmente un incendio alejado de la convivencia humana.

## 1.2. PRESENTACIÓN DEL PROBLEMA

Este proyecto buscará implementar una lógica para clasificar y segmentar humo en imágenes. Esta lógica deberá ser capaz de ejecutarse en computadoras de bajos recursos como una Raspberry Pi. Se prevé realizar esta computación en entornos sin acceso a corriente eléctrica por lo que no deberá conllevar un gran gasto energético para las baterías.

La clasificación de imágenes es la identificación entre un conjunto de categorías que pertenece una imagen. La clasificación es un ejemplo de reconocimiento de patrones. En este caso se pretenderá asignar una imagen a la clase humo o no humo.

---

<sup>2</sup><https://uh.gsstatic.es/sfAttachPlugin/909033.jpg>

La segmentación de imágenes es el proceso de separar sus píxeles en grupos distintos. Un segmento bien definido es aquel en el que los miembros del segmento son similares entre sí y también son diferentes de los miembros de otros segmentos. En este caso se separará el humo de las imágenes.

En este trabajo se pretende realizar un estudio de diferentes técnicas de aprendizaje profundo recopilando resultados hasta obtener la solución con la mejor relación precisión-eficiencia tanto para la clasificación de imágenes como para la segmentación. Además, se pretende realizar un ajuste óptimo de parámetros para la detección de humo como, por ejemplo, el número de píxeles que consideremos que hay humo en caso de segmentación.

Una vez obtenido un modelo adaptado para nuestras necesidades se realizará una configuración semiautomática de una Raspberry Pi y las pruebas correspondientes para verificar su correcto funcionamiento en este tipo de dispositivos.

Mas allá de este proyecto se realizarán otras tareas básicas para su correcto funcionamiento. Un primer paso será la realización de una serie de *scripts* para el lanzamiento de una señal LoRa (Long Range) por un computador de pequeño tamaño y un receptor que dará sus correspondientes avisos. Después se configurará estas máquinas para realizar la predicción y su previa detección a una cierta frecuencia y su control de recursos. Por último, su instalación en lugares donde se podrá visualizar el posible humo, como previamente citamos la demanda en torres de alta tensión (ver Fig. 1.3<sup>3</sup>).



**Figura 1.3:** Imagen de torres de alta tensión

### 1.3. ESTRUCTURA DEL DOCUMENTO

Este documento se estructura de la siguiente forma:

- **Capítulo 2: Objetivos.**

Describe la meta principal del proyecto y las metas secundarias que se pretenden alcanzar. Muestra el alcance del proyecto.

- **Capítulo 3: Antecedentes.**

Recoge el estado del arte de las técnicas utilizadas durante el proyecto. Se incluye información tanto de los principios del aprendizaje automático hasta de modelos de redes neuronales

---

<sup>3</sup>[https://www.lavanguardia.com/r/GODO/LV/p6/WebSite/2019/04/16/Recortada/LAVANGUARDIA\\_G\\_22221616600-kXZG-U461699409218qbH-992x558@LaVanguardia-Web.jpg](https://www.lavanguardia.com/r/GODO/LV/p6/WebSite/2019/04/16/Recortada/LAVANGUARDIA_G_22221616600-kXZG-U461699409218qbH-992x558@LaVanguardia-Web.jpg)

actuales que se utilizarán en el proyecto; además de información sobre la tecnología IoT actual.

- **Capítulo 4: Metodología.**

Describe la metodología que se ha empleado para la gestión de trabajo de este proyecto. Además del marco tecnológico que se ha ido utilizando en el desarrollo.

- **Capítulo 5: Resultados.**

Explica y muestra todos los resultados obtenidos durante el desarrollo del proyecto. Recoge todo el proceso de investigación y entrenamiento de diferentes modelos de redes neuronales, el despliegue de un sistema para la detección de humos y una visión de la aplicación web.

- **Capítulo 6: Conclusiones.**

Detalla las conclusiones obtenidas, objetivos alcanzados y se realiza un análisis de la ampliación del proyecto que se propone para un futuro.

- **Anexo A: Hats para Raspberry Pi.**

Evalúa los costes del despliegue hardware de una Raspberry Pi funcional para el proyecto.

---

## CAPÍTULO 2

# OBJETIVO

---

### 2.1. OBJETIVO GENERAL

El objetivo general de este proyecto es detectar, a partir de imágenes, posibles incendios desde torres de alta tensión con el entrenamiento de redes neuronales profundas sin supervisión.

Se desarrollará una aplicación para Raspberry Pi que detecta humo a través de una cámara. Además, se realizará una aplicación web para detectar humo en imágenes tanto en forma de clasificación, valorando si es o no humo, o en forma de segmentación, visualizando que píxeles corresponde a humo.

### 2.2. OBJETIVOS PARCIALES

Los objetivos parciales marcados para este proyecto son:

1. **Creación del conjunto de datos.**

Creación de un conjunto de imágenes a partir de búsquedas en la red. Se elegirán imágenes con humo y sin humo en cantidades iguales. Se evitarán imágenes duplicadas y de poca resolución. Además, se buscarán estas imágenes con su correspondiente máscara de humo segmentado o se realizará la propia segmentación.

2. **Análisis de arquitecturas de redes neuronales de clasificación de imágenes.**

Se ejecutarán diferentes procesos de entrenamiento con el conjunto de datos previamente creado en modelos de redes neuronales dedicadas a la clasificación de imágenes. Se visualizarán resultados y valorarán la precisión de éstos.

3. **Análisis de arquitecturas de redes neuronales de segmentación semántica.**

Se ejecutarán diferentes procesos de entrenamiento con el conjunto de datos previamente creado en modelos de redes neuronales dedicadas para la segmentación de imágenes. Se visualizarán resultados y valorarán la precisión de éstos.

4. **Creación de módulo de detección de incendios.**

Una vez elegida la red neuronal más precisa y capaz de ejecutarse en una computadora embebida de pequeño tamaño, se realizará una aplicación para detectar humo en una Raspberry Pi y se comprobará su correcto funcionamiento.

5. **Creación de aplicación web.**

Para la visualización al alcance del usuario se realizará una interfaz gráfica en web donde se podrá probar los modelos entrenados previamente y visualizar sus resultados. Además, se propone la creación de una interfaz sencilla para una explicación de la instalación y uso de esta.

6. **Creación de API REST para predecir humo.**

De cara a un ámbito más práctico y rápido se realizará una API REST con interacciones sencillas y eficientes con el fin de poder explorar el modelo creado.



---

## CAPÍTULO 3

# ANTECEDENTES

---

### 3.1. MACHINE LEARNING Y DEEP LEARNING

*Machine learning*, también llamado aprendizaje automático, es la capacidad de un sistema para adquirir nuevo conocimiento y reorganizar el conocimiento existente con el nuevo adquirido [12]. El objetivo del aprendizaje automático es encontrar una solución a un problema parecido a uno que ya se resolvió. De cara a qué procedimientos se usan para alcanzar este objetivo se pueden diferenciar tres tipos de aprendizaje:

- **Aprendizaje supervisado:** Para el aprendizaje se necesitarán datos previamente etiquetados. Se pueden realizar dos tipos de acciones con este procedimiento:
  - **Clasificación:** Esta acción predice a qué categoría pertenece una serie de características.
  - **Regresión:** A diferencia de clasificación, predice valores numéricos continuos. Esta predicción resultará en la relación de las características y el resultado.
- **Aprendizaje no supervisado:** Este tipo tratará con un aprendizaje a partir de datos no etiquetados. Su objetivo es la extracción de una estructura o información útil de los datos. Se utilizará para datos que previamente no se sabe su relación o etiquetado.
- **Aprendizaje por refuerzo:** En este tipo de aprendizaje el sistema que realiza las acciones resultantes del aprendizaje se le llama agente. El objetivo es aprender a partir de iteraciones pasadas. Por tanto, la problemática será la forma de validar un resultado.

*Deep learning* o aprendizaje profundo es una parte de la familia de técnicas de *machine learning*. Se centra en el uso de técnicas de redes neuronales artificiales con arquitecturas de redes profundas ya sean neuronales, neuronales recurrentes o de creencia. Aunque existen muchos más tipos de redes este campo está en continua evolución.

De la misma forma que se indicaba anteriormente se pueden diferenciar varios tipos de aprendizaje: supervisado, no supervisado y por refuerzo. La técnica más utilizada en clasificación de imágenes hoy en día son las redes neuronales profundas supervisadas.

### 3.2. REDES NEURONALES

Las redes neuronales artificiales son sistemas informáticos con lógicas complejas inspirados en las redes neuronales biológicas. Al considerarse dentro del conjunto de técnicas de *machine learning*, estos sistemas aprenden a dar ciertos resultados a partir de una entrada, sin ser previamente programados con lógicas específicas. Por ejemplo, en el reconocimiento de imágenes, pueden aprender a identificar imágenes que contienen humo analizando imágenes de ejemplo que se han etiquetado manualmente como “humo” o “no humo”. O bien pueden aprender a identificar si un píxel de una imagen es humo analizando imágenes de máscaras previamente segmentadas manualmente como “humo” o “no humo”.

La unidad básica de una red neuronal es el perceptrón [16]. Un perceptrón trata de una variable principal cuya fórmula matemática es:

$$\begin{aligned}
 w &: \text{Valor real} \\
 b &: \text{Sesgo} \\
 x &: \text{Valor a predecir}
 \end{aligned}$$

$$f(x) = \begin{cases} 1 & \text{if } w * x + b > 0, \\ 0 & \text{otherwise} \end{cases} \quad (3.1)$$

Una red neuronal trata de un conjunto de perceptrones con la capacidad de conectarse entre ellas. Cada conexión puede transmitir un valor a otros perceptrones. Lo más común es agruparlas en capas surgiendo así el concepto multicapa. Las redes neuronales monocapa son fáciles y rápidas de entrenar, pero solo pueden llegar a separar datos lineales. Sin embargo, con las redes multicapa abre la puerta a múltiples funciones.

En una red neuronal se puede distinguir 3 tipos de capas:

- **Capa de entrada:** Suele ser la primera capa de la red neuronal. Se encarga de pasar los datos a la capa oculta.
- **Capa oculta:** Es la capa más importante ya que se ve afectada por el aprendizaje de la red. Además, es la responsable de la lógica de la red.
- **Capa de salida:** Sus valores de salida son el resultado final de la red. Normalmente el tamaño del vector resultante de la capa es la cantidad de categorías a clasificar.

Las redes neuronales con un gran número de capas ocultas son usadas habitualmente en técnicas de *deep learning*.

### 3.2.1. Modelos para clasificación de imágenes

La clasificación de imágenes consiste agrupar imágenes en categorías semánticamente similares utilizando características visuales. En base a estas agrupaciones, se pueden percibir índices efectivos para una base de datos de imágenes. Para un aprendizaje supervisado de modelos de redes neuronales dedicados a la clasificación de imágenes esta base de datos se convierte en el *dataset* que posteriormente será dedicado al aprendizaje de los pesos.

Por norma general estos modelos tendrán las tres capas anteriormente citadas: capa de entrada, capa de salida y capa oculta.

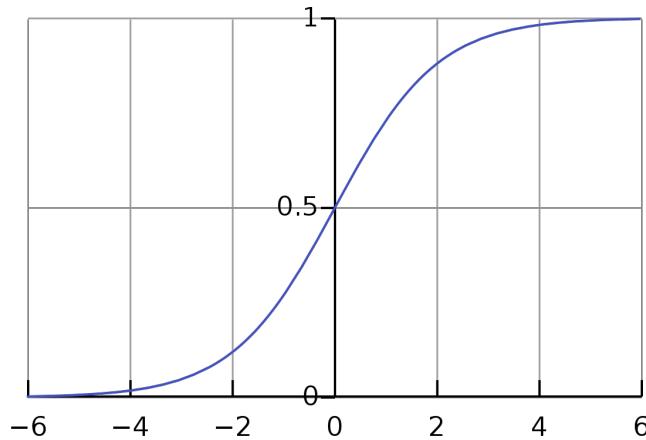
La capa de entrada de estos modelos será cada píxel de una imagen con su correspondiente valor. El valor de cada píxel puede ser diferente en cada caso de clasificación, como por ejemplo para imágenes a color, RGB, o para imágenes en blanco y negro, escala de grises o binaria (negro o blanco). Por tanto, el número de perceptrones de la capa de entrada estará limitado a el número de píxeles por el valor del color.

La capa de salida se construirá con un vector del tamaño del número de clases. Esta capa es una capa de activación siendo la más sencilla y la encargada de dar una función a la entrada del perceptrón. Estas funciones pueden variar, pero para el caso de la clasificación de imágenes las más comunes son Softmax (Ver Eq. 3.2) para clasificar varias categorías y Sigmoid (Ver Fig. 3.1<sup>1</sup>) para saber si existe solo una categoría.

---

<sup>1</sup>[https://en.wikipedia.org/wiki/Sigmoid\\_function#/media/File:Logistic-curve.svg](https://en.wikipedia.org/wiki/Sigmoid_function#/media/File:Logistic-curve.svg)

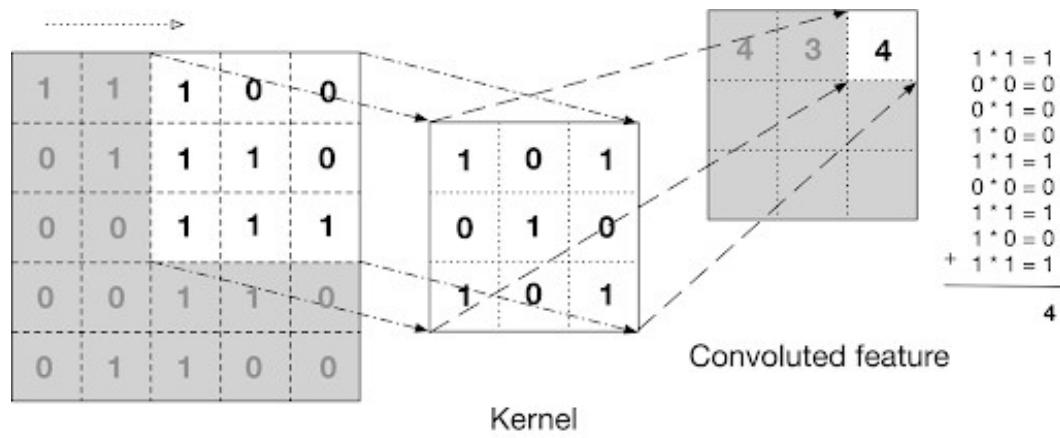
$$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}} \quad (3.2)$$



**Figura 3.1:** Función Sigmoid

Como previamente se explica los modelos habituales dedicados a la clasificación de imágenes son modelos pesados que están considerados dentro de técnicas de *deep learning*, por ello su capa oculta estará compuesta de multitud de capas y perceptrones.

Elementos fundamentales en las capas ocultas para la clasificación de imágenes son las capas de convolución. Estas capas se usan para cambiar los valores de entrada aplicando sus propios filtros que irán variando según el valor de los pesos. Una vez se ha realizado el filtro se realiza una reducción en la dimensión del vector (Ver Fig. 3.2<sup>2</sup>), aunque puede haber excepciones donde no se aplica esta reducción.



**Figura 3.2:** Imagen de una capa de convolución actuando sobre una imagen

También existen otros tipos de capas habitualmente usadas en modelos de clasificación de imágenes. Estas capas no se ven influidas por los pesos que se ajustan durante el proceso de entrenamiento:

<sup>2</sup>[https://miro.medium.com/max/875/0\\*QS1ArBEUjjySXhE.png](https://miro.medium.com/max/875/0*QS1ArBEUjjySXhE.png)

- **Capa de agrupación:** también llamada capa de *pooling*, se usa para reducir la dimensión de las imágenes. Consiste en dividir las imágenes en segmentos de un tamaño determinado y se aplica una función para transformar ese segmento a un solo valor. Esta función puede ser: el máximo valor (Ver Fig. 3.3<sup>3</sup>), el mínimo valor, la media de los valores o la suma de los valores de los segmentos.

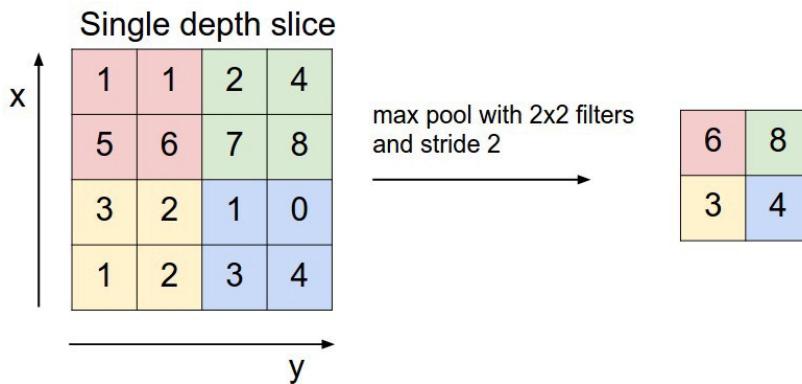


Figura 3.3: Imagen de una capa de agrupación con la función de máximos valores actuando

- **Capa de normalización:** esta capa normaliza los valores del vector.
- **Capa de concatenación:** esta capa se usa para unir los valores de diferentes capas, por tanto su salida tendrá un tamaño igual a la suma de las entradas de la capa.

El uso de redes neuronales en clasificación de imágenes se ha extendido después de que en 2012 la red neuronal AlexNet sobreponerse a los resultados obtenidos en el reto ImageNet (ILSVRC) [4] siendo anteriores ganadores y competidores algoritmos de procesamiento de imágenes sin uso de técnicas de *deep learning*. Desde esa rotunda victoria los siguientes años fueron dominados por multitud de tipos de redes neuronales. Además, ImageNet se convirtió en un reto referente en la investigación de nuevos modelos de redes neuronales capaces de mejorar la clasificación de imágenes. Desde 2013 realiza retos de segmentación semántica de objetos y desde 2015 de segmentación de objetos en vídeo.

Los resultados de algunos modelos de clasificación que recoge ImageNet se ven reflejados en la gráfica 3.4<sup>4</sup>.

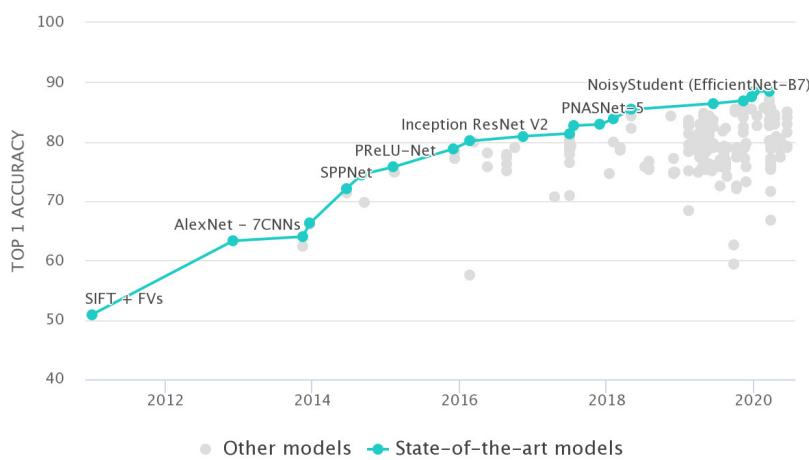


Figura 3.4: Gráfica con los ganadores de ImageNet según el año

<sup>3</sup>[https://cdn-images-1.medium.com/max/1000/1\\*GksqN5XY8HPpIddm5wzm7A.jpeg](https://cdn-images-1.medium.com/max/1000/1*GksqN5XY8HPpIddm5wzm7A.jpeg)

<sup>4</sup><https://paperswithcode.com/sota/image-classification-on-imagenet>

### ResNet

ResNet, acrónimo de *Residual Neural Network*, está desarrollado por Microsoft y fue el ganador del reto ImageNet en 2015 con solo un error del 3,57 %. Este modelo tiene variantes con diferentes números de capas. El modelo que resultó ganador tenía 152 capas, aunque existen variantes con 50 y 101 capas más eficiente que consiguen buenos resultados.

La diferencia de este modelo respecto a un modelo plano de capas de convolución son los accesos directos de capas anteriores, *shortcut connections*. Como se ve en la figura 3.5<sup>5</sup> la red residual recibe señal tanto de su capa anterior como la que tuvo la capa predecesora a su predecesora y posteriormente pasada por una capa de activación. Su diseño final fue más profundo y se basa en la construcción de modelos a partir de un bloque de capas llamado “Bottleneck” (Ver Fig. 3.6).

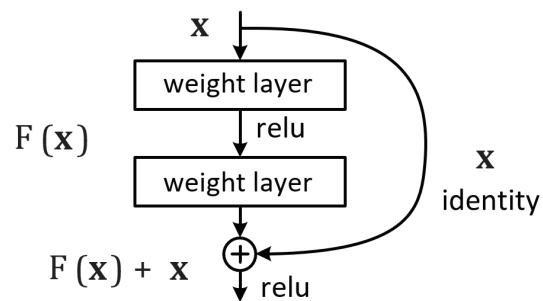


Figura 3.5: Imagen de un bloque residual

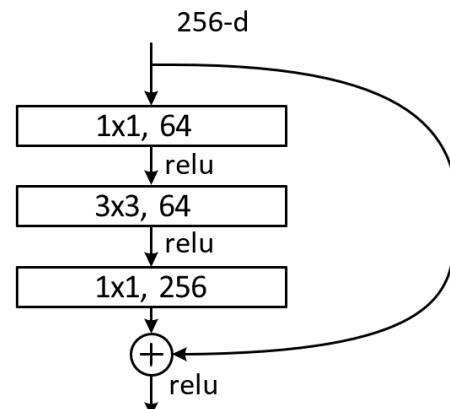


Figura 3.6: Imagen de un bloque Bottleneck

### Inception

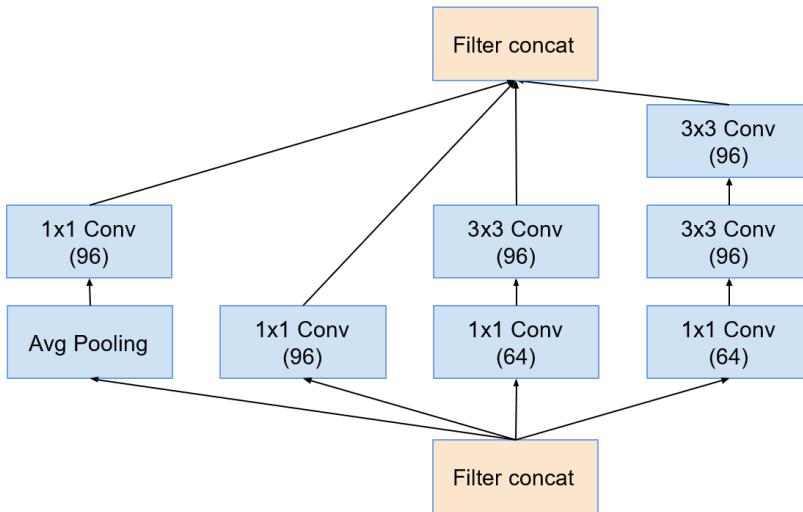
InceptionNet está desarrollado por Google y constituye una serie de versiones de modelos (InceptionNetV3, InceptionNetV4...) [19]. Estos modelos se construyen a partir de un bloque de capas llamado “Inception” que consiste en realizar diferentes convoluciones variando el tamaño del segmento a filtrar y la capa de *maxpooling* con la misma entrada en paralelo para, posteriormente, realizar una concatenación de todas las salidas (Ver Fig. 3.7<sup>6</sup>).

### Inception-ResNet

Se constituye de una combinación de bloques Inception y Bottleneck (ResNet). Con su segunda versión llegó a ser ganador del reto ImageNet de 2017 con un error de tan solo 4,9 %.

<sup>5</sup><https://arxiv.org/pdf/1512.03385.pdf>

<sup>6</sup><https://arxiv.org/pdf/1602.07261.pdf>



**Figura 3.7:** Imagen de un bloque Inception

### EfficientNet

También investigadores de Google son los autores del modelo EfficientNet el cual es ganador de ImageNet 2019 y 2020 [20]. Además de conseguir una precisión mejor que ResNet e Inception-ResNet en ImageNet, es un modelo con menor cantidad de parámetros y de capas, por tanto, mucho más eficiente como su propio nombre indica. Actualmente es considerado el mejor modelo de clasificación de imágenes por sus resultados.

El modelo EfficientNetB0 se puede ver en la figura 3.8<sup>7</sup>. Los demás modelos se basan en la misma arquitectura, pero con diferentes valores de configuración en cada capa.

Stage $i$	Operator $\hat{F}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	$224 \times 224$	32	1
2	MBConv1, k3x3	$112 \times 112$	16	1
3	MBConv6, k3x3	$112 \times 112$	24	2
4	MBConv6, k5x5	$56 \times 56$	40	2
5	MBConv6, k3x3	$28 \times 28$	80	3
6	MBConv6, k5x5	$14 \times 14$	112	3
7	MBConv6, k5x5	$14 \times 14$	192	4
8	MBConv6, k3x3	$7 \times 7$	320	1
9	Conv1x1 & Pooling & FC	$7 \times 7$	1280	1

**Figura 3.8:** Imagen del modelo EfficientNetB0

<sup>7</sup><https://arxiv.org/pdf/1905.11946.pdf>

### 3.2.2. Modelos para segmentación de imágenes

La segmentación de imágenes consiste en clasificar cada píxel de una imagen en una categoría. Se podría explicar también como la generación de una imagen con la profundidad que indica cada categoría. Para un aprendizaje supervisado de redes neuronales dedicadas a la segmentación de imágenes será necesario un conjunto de datos con imágenes originales e imágenes de máscaras binarias por cada categoría.

La capa de entrada de estos modelos será igual que en clasificación mientras que la capa de salida se construirá con un vector del tamaño del número de píxeles de la imagen original con la profundidad del número de clases.

Además de las capas explicadas en los modelos para clasificación de imágenes, para la segmentación se utilizan dos tipos de capas más:

- **Capa de deconvolución:** al igual que la capa de convolución esta capa realiza un filtro en los valores de entrada variando según el valor de los pesos, pero esta capa aumenta la dimensión del vector (Ver Fig. 3.9<sup>8</sup>), aunque, al igual que la convolución, puede haber excepciones donde no se aplica este aumento. Su uso para la estructura de una red se llama decodificación *transpose*.

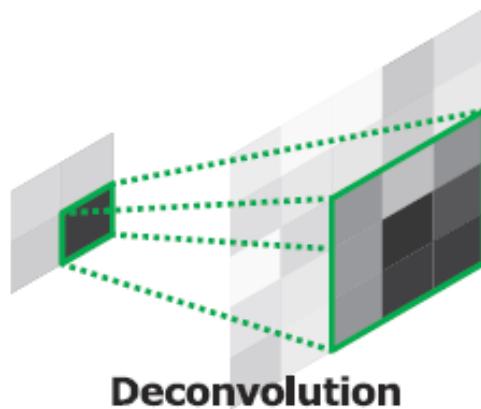


Figura 3.9: Imagen de una capa de deconvolución actuando

- **Capa de *upsampling*:** esta capa aumenta la dimensión de las imágenes. Hace una redimensión rellenando los píxeles con valores iguales (Ver Fig. 3.10<sup>9</sup>).

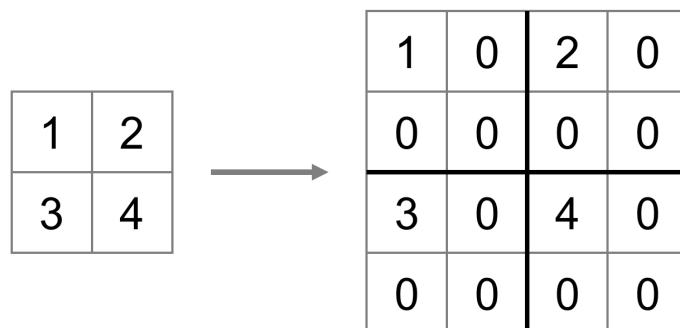


Figura 3.10: Imagen de una capa de *upsampling* actuando

<sup>8</sup>[https://miro.medium.com/max/679/1\\*AbCrAqPBfkqGRdhKtiZQqA.png](https://miro.medium.com/max/679/1*AbCrAqPBfkqGRdhKtiZQqA.png)

<sup>9</sup><https://www.oreilly.com/library/view/deep-learning-for/9781788295628/assets/a4df8c96-4e64-450f-b891-9efb18fc7368.png>

### U-Net

U-Net es un modelo de segmentación de imágenes que fue creada para el reto de Detección de Caries en Radiografías de Mandíbulas en 2015 [6].

Su estructura se trata de un primer modelo de clasificación variante, después deconvoluciona de forma exponencial hasta llegar a la dimensión deseada. Además, concatena los valores que la red de clasificación calcula antes de reducir la dimensión junto con los valores de la capa de deconvolución previa para posteriormente deconvolucionar. La figura 3.11<sup>10</sup> muestra el modelo completo de una U-Net.

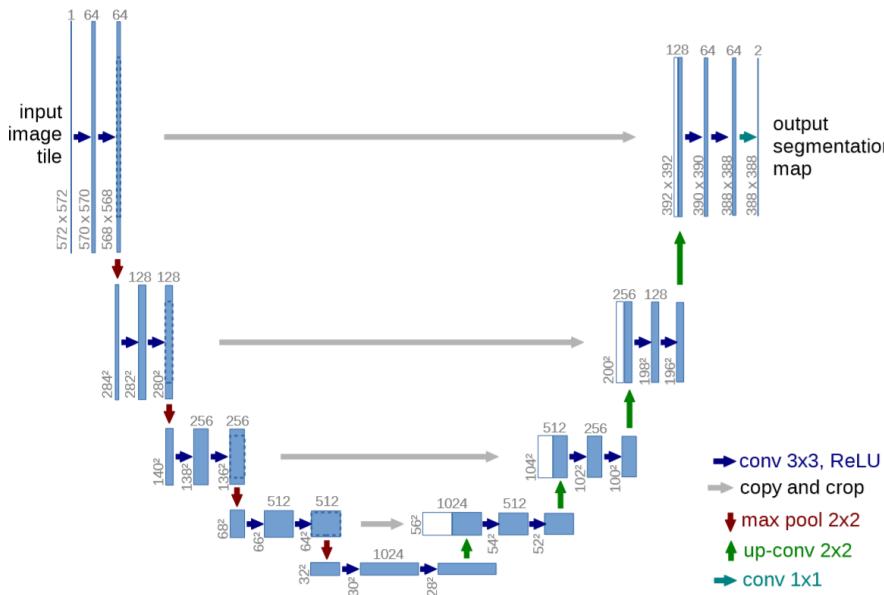


Figura 3.11: Imagen del modelo U-Net

## 3.3. DATA AUGMENTATION

Un problema común a la hora de abordar un caso de uso en el ámbito de *machine learning* es la falta de una gran cantidad de datos para llevar a cabo el proceso de entrenamiento/aprendizaje. *Data augmentation* sirve para ampliar la base de datos o más bien dar aleatoriedad a los datos en un entrenamiento supervisado.

Realmente es una herramienta de utilidad cuando no existen multitud de datos y nuestro caso de uso no es englobado por todos ellos. Aunque no se pretenderá crear nuevas categorías que abordar con solo *data augmentation*, éste también ayuda a controlar el sobreaprendizaje.

En el análisis de imágenes mediante técnicas de *deep learning* el *data augmentation* [13] es comúnmente usado, esto es debido a que, la edición de imágenes a partir de filtros automatizados y aleatorios son una gran herramienta para aumentar la cantidad de datos (Ver Fig. 3.12<sup>11</sup>).

Dentro de este proceso se utilizan herramientas como filtros para voltear imágenes, efectuar rotaciones o incluso el escalado de las mismas. Por otro lado, según el caso de uso concreto, se pueden llegar utilizar diferentes filtros como, por ejemplo, variaciones de brillo en un caso de uso que haya posibilidad de imágenes con diferente brillo, o un cambio de color en un caso de uso que los colores de cada categoría sean variantes y no sea una característica útil.

<sup>10</sup><https://arxiv.org/pdf/1505.04597.pdf>

<sup>11</sup>[https://nanonets.com/blog/content/images/2018/11/1\\_dJNIEc7yf93K4pjRJL55PA--1-.png](https://nanonets.com/blog/content/images/2018/11/1_dJNIEc7yf93K4pjRJL55PA--1-.png)

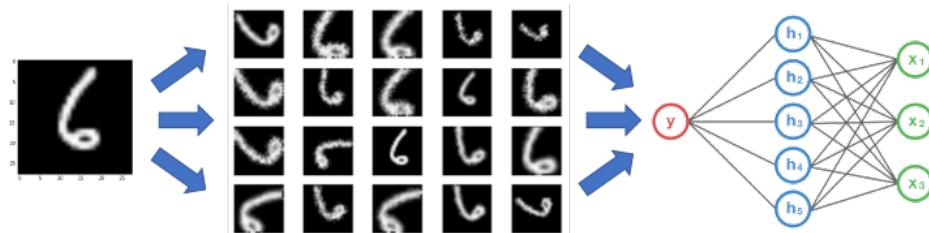


Figura 3.12: Imagen ejemplo de Data Augmentation

### 3.4. IOT

*Internet of Things*, también llamado internet de las cosas, es un sistema de dispositivos informáticos interrelacionados y de funcionamiento autónomo. Normalmente suelen ser artefactos de pequeño tamaño o placas de computación.

El ámbito de aprendizaje automático es uno de los objetivos principales de aplicación en la actualidad para esta tecnología. El avance de redes neuronales enfocadas al tratamiento de imágenes hacen de las tecnologías IoT una herramienta eficaz para la búsqueda de datos.

Empresas como NVIDIA, Google y Microsoft ya tienen máquinas preparadas para actuar directamente con redes neuronales complejas en entornos IoT. Por ejemplo, NVIDIA Jetson Nano o Google Coral.

#### 3.4.1. Raspberry Pi

Es la alternativa más común para el uso de una máquina en IoT. Con el paso del tiempo han ido mejorando su modelo hasta llegar al actual, Raspberry Pi 4 modelo B (Ver Fig. 3.13<sup>12</sup>). Este modelo se compone de:

- Procesador: Broadcom BCM2711 1.5 GHz Quad core A72 64-bit.
- Procesador gráfico: Broadcom VideoCore VI.
- RAM: Variable entre 2, 4 o 8 GB.
- Almacenamiento: microSD a elección del usuario.
- Alimentación: 5V, 3A.



Figura 3.13: Imagen de una Raspberry Pi 4 Modelo B

También se desarrolló un sistema operativo específicamente para estos modelos. Raspbian es una variación del sistema operativo Debian (Linux) enfocada en la arquitectura de Raspberry Pi.

<sup>12</sup>[https://images-na.ssl-images-amazon.com/images/I/71IOISwSYZL.\\_AC\\_SX425\\_.jpg](https://images-na.ssl-images-amazon.com/images/I/71IOISwSYZL._AC_SX425_.jpg)



---

## CAPÍTULO 4

# METODOLOGÍA

---

En este capítulo se describirá la metodología de desarrollo, la metodología de gestión y la planificación del proyecto. También se hablará del marco tecnológico que se utilizará a lo largo del mismo.

### 4.1. METODOLOGÍA DE DESARROLLO DEL PROYECTO

La metodología utilizada en este proyecto será una metodología ágil tal y como se ha realizado en otros proyectos similares [21]. El proyecto buscará mantener ciclos de desarrollo iterativos e incrementales, aunque haya sido diseñado y desarrollado por una sola persona. Por su característica ágil este proyecto podría ser realizado en menos tiempo y costes por más de una persona.

La búsqueda de resultados cada vez más óptimos y los parámetros de reajustes en los entrenamientos de redes neuronales conllevan un proceso de **Implementación-Prueba-Visualización-Error**. Este proceso permitirá más agilidad en las iteraciones de pruebas. También llevará a una visualización de los resultados incrementales siendo los últimos resultados más óptimos que los primeros.

#### 4.1.1. Método de trabajo

A grandes rasgos los entrenamientos de redes neuronales dedicadas a la imagen varían por ciertos parámetros. Estos parámetros se irán modificando de mayor a menor importancia en el resultado. El orden de mayor a menor será respectivamente (Ver Fig. 4.1):

1. **Dataset**: Este parámetro será constante por el bajo número de posibilidades que existen, por lo que se tiene dos únicos conjuntos: para segmentación y para clasificación.
2. **Modelo de red neuronal**: Se realizarán pruebas con modelos explicados en el capítulo 3, priorizando los modelos más actuales y con mejores resultados.
3. **Bloque de decodificación** (solo en segmentación): Existen dos tipos de decodificación: *transpose* y *upsampling*.
4. **Data augmentation**: De este parámetro existen multitud de variantes que sería imposible probar todas y seguramente no se conseguirá probar la óptima.

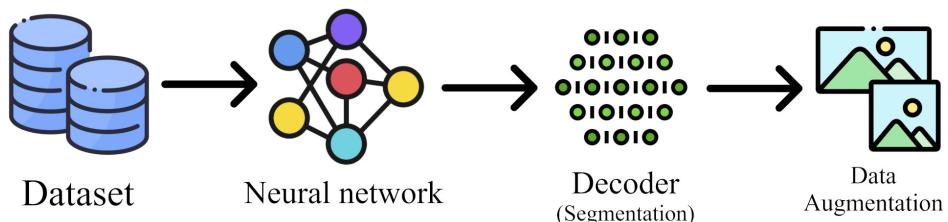


Figura 4.1: Ilustración del orden de importancia de los parámetros para entrenamientos

## 4.2. METODOLOGÍA DE GESTIÓN DEL PROYECTO

Para la gestión de tareas se utilizará la técnica de tableros Kanban. Kanban se define por el sistema de producción JIT (Just-in-Time) de Toyota. Un sistema de producción empíricamente eficiente. Es común ver usar Kanban en la gestión de proyectos con metodologías ágiles.

El sistema consta de un tablero con varias columnas donde se irán situando nombres de tareas y, opcionalmente, detalladas. Este tablero es simbolizado típicamente por una tabla de corcho pegada en la pared con las columnas pintadas y las tareas escritas en *post-its* (Ver Fig. 4.2<sup>1</sup>).

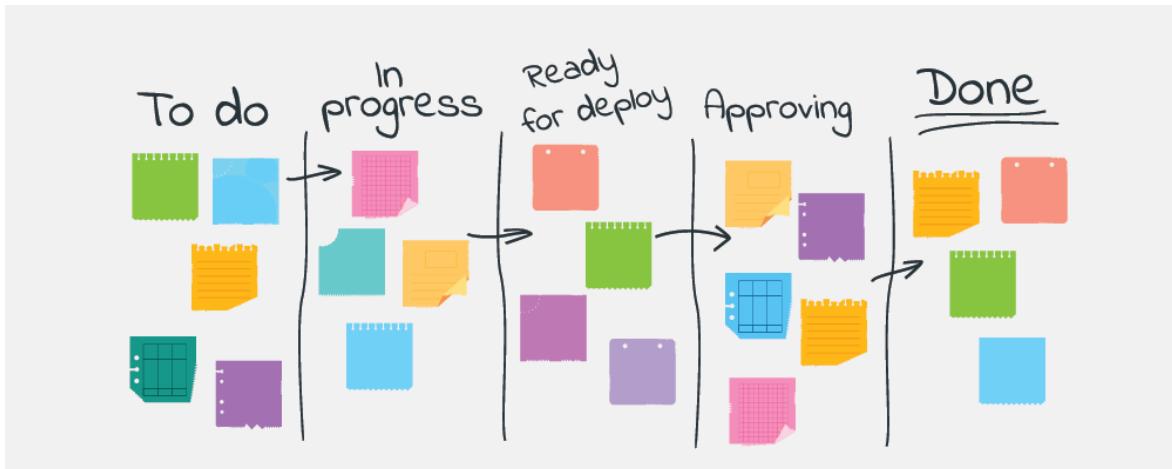


Figura 4.2: Imagen de ejemplo de un tablero Kanban

Las columnas del tablero pueden variar según el proyecto o incluso el grupo de trabajo. La composición de columnas que se utilizará en este proyecto consta de la siguiente estructura:

- **To do:** Serán tareas que están pendientes de realizar. Normalmente se introducen todas las tareas que han sido planificadas en el análisis del proyecto, pero siempre se pueden añadir nuevas tareas que van apareciendo en el transcurso del desarrollo del proyecto.
- **Doing:** Serán tareas que se encuentran en desarrollo o están siendo realizadas en ese momento.
- **Review:** Una vez una tarea es terminada de la columna Doing será almacenada en esta columna. Cuando se comprueba si realmente está terminada y no se encuentran errores será asignada a la columna Done. Si contiene algún error o falta de desarrollo volverá a la columna Doing.
- **Done:** Serán tareas que han sido finalizadas y previamente comprobada su funcionalidad. Si se realiza un cambio en el diseño del proyecto durante la etapa de desarrollo e influye a una tarea finalizada esta tarea podrá moverse a otra columna o eliminarse del tablero según corresponda.

Típicamente esta técnica se usa en equipos de trabajo de dos o más personas por tanto cada tarea es asociada a una persona, pero como este proyecto se desarrollará por una sola todas las tareas son asignadas a ella.

Además, existirán niveles de prioridades para las tareas. Por tanto, si una tarea tiene una prioridad más alta en una columna se realizará la función que corresponda antes que otras de menor prioridad. Esta prioridad será valorada en To do, Doing y Review. En cualquier momento la prioridad de una tarea podrá ser modificada.

<sup>1</sup><https://t8f8b3g9.stackpathcdn.com/wp-content/uploads/2020/03/Kanban-board-examples.png>

### 4.3. PLANIFICACIÓN

Para la planificación de este proyecto se usarán iteraciones agrupadas en versiones como resultado de utilizar una metodología ágil.

#### Versión 0: Diseño

- **Iteración 0:** *Análisis y diseño del problema.* Para el desarrollo del proyecto se realizará un análisis y diseño general, se elegirá la metodología de gestión, se marcará el marco tecnológico y se realizará un diagrama de Gantt aproximado del proyecto. Además, se planificará las iteraciones que tendrán su propio análisis y diseño. Esta iteración tendrá una duración de 1 semana.

#### Versión 1: Clasificación

- **Iteración 1:** *Generación de un dataset dedicado a clasificación de imágenes.* El producto de esta iteración proporcionará un conjunto de imágenes. Se realizarán búsquedas por la red. Esta iteración tendrá una duración de 2 días.
- **Iteración 2:** *Realización de entrenamientos de modelos de redes neuronales para clasificación de imágenes.* Para la realización de esta iteración se utilizará la ya mencionada metodología basada en **Implementación-Prueba-Visualización-Error** considerando ciertos parámetros para la mejora de los resultados. Se habrán realizado entrenamientos hasta llegar al resultado óptimo. La duración de esta iteración es de 2 semanas.
- **Iteración 3:** *Creación de una API REST dedicada a la predicción de clasificación de imágenes.* El desarrollo de la API REST proporcionará una herramienta de predicción rápida para las imágenes de prueba y servirá para posteriores iteraciones. También se desarrollará un cliente por comandos que permitirá el envío de múltiples imágenes para su predicción. Además, permitirá una fácil implementación de sistema distribuido en un futuro. La duración de esta iteración será de 1 semana.

#### Versión 2: Segmentación

Las iteraciones de esta versión son semejantes a las de la versión 1 solo que el tipo de datos que se será utilizado es diferente y por ello con diferente desarrollo.

- **Iteración 4:** *Generación de un dataset dedicado a segmentación de imágenes.* El producto de esta iteración proporcionará un conjunto de imágenes. Se realizarán búsquedas por la red y segmentación propia. Esta iteración tendrá una duración de 1 semana.
- **Iteración 5:** *Realización de entrenamientos de modelos de redes neuronales para segmentación de imágenes.* Para la realización de esta iteración se ha utilizado la ya mencionada metodología basada en **Implementación-Prueba-Visualización-Error** considerando ciertos parámetros para la mejora de los resultados. Se han ido realizando entrenamientos hasta llegar al resultado óptimo. La duración de esta iteración es de 2 semanas.
- **Iteración 6:** *Funciones añadidas a la API REST dedicadas a la predicción de segmentación de imágenes.* El desarrollo de la API REST proporcionará una herramienta de predicción rápida. La duración de esta iteración será de 3 días.

#### Versión 3: Interfaz Exterior

- **Iteración 7:** *Configuración de Raspberry Pi y realización de pruebas.* Se utilizará una Raspberry Pi para probar los mejores resultados de iteraciones pasadas. Se pretenderá realizar *scripts* de instalación para este dispositivo. Esta iteración durará 1 semana.

- **Iteración 8:** Creación de una aplicación web. Para una visualización a nivel usuario se realizará un interfaz web donde se podrá predecir imágenes de forma intuitiva. La duración de esta iteración será de 1 semana.

El diagrama de Gantt está desarrollado entorno a las iteraciones y versiones (Ver Fig. 4.3).

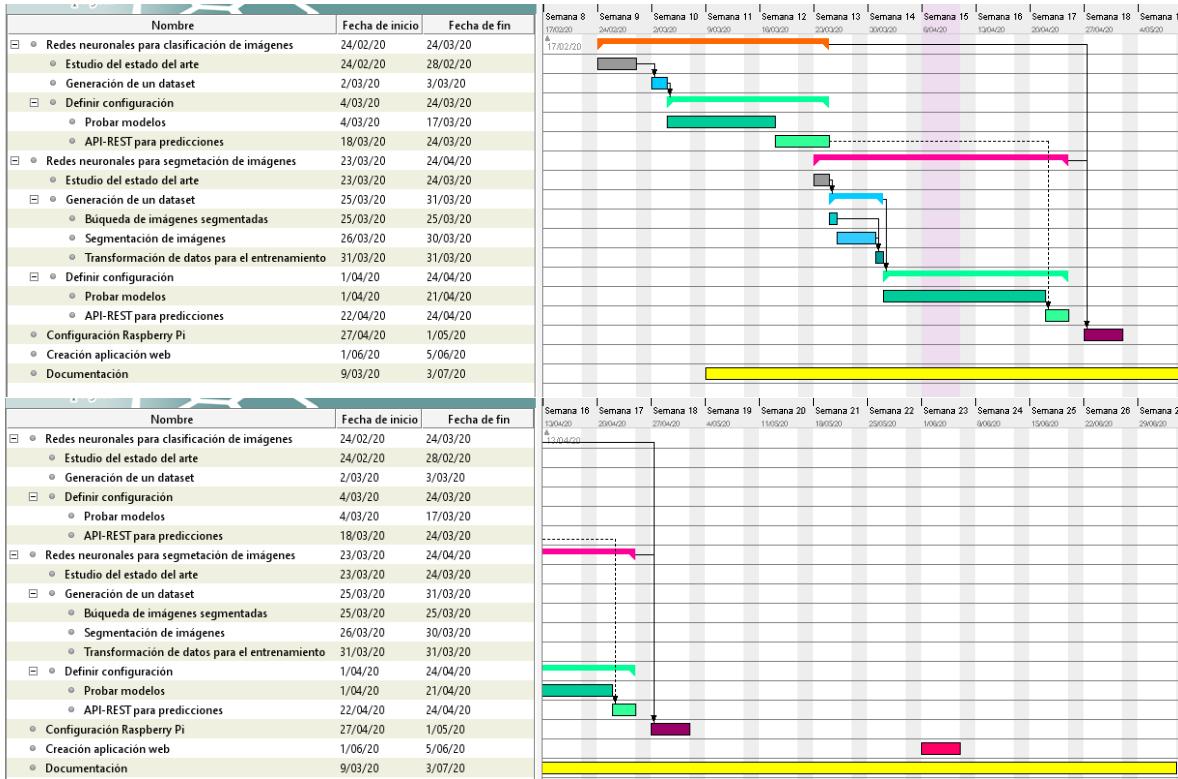


Figura 4.3: Diagrama de Gantt Marzo-Julio

La hoja de ruta se engloba en el periodo marzo-junio de 2020 y, a diferencia de las demás planificaciones, añade un seguimiento de esta documentación (Ver Tabla 4.1).

Tabla 4.1: Hoja de ruta

	<b>31 de marzo</b>	<b>30 de abril</b>	<b>31 de mayo</b>	<b>15 de junio</b>	<b>30 de junio</b>
<b>Desarrollo Software</b>	Clasificación de imágenes	Segmentación de imágenes	API REST Raspberry Pi	Aplicación web	Pruebas Revisión
<b>Documentación</b>	Propuesta	Estructura	Introducción Metodologías Antecedentes	Resultados Conclusiones	Anexos Revisión

## 4.4. MARCO TECNOLÓGICO

Para este proyecto se utilizará tecnología actual dedicada al *deep learning*. También se dará importancia a las herramientas y librerías multiplataforma y con licencias de uso *open-source*, es decir, gratuitas y con uso permitido para proyectos empresariales, además de tener acceso al código fuente de la herramienta.

### 1. Librerías:

- [Python \(Anaconda\)](#)
- [TensorFlow](#)
- [Keras](#)
- [OpenCV](#)
- [ImgAug](#)
- [Django](#)
- [FastAPI](#)

### 2. Herramientas de soporte

- [Visual Studio Code](#) (Desarrollo)
- [GanttProject](#) (Planificación)
- [LaTEX/Overleaf](#) (Documentación)

### 4.4.1. Librerías

#### Python

Python es un lenguaje de programación orientado a objetos. Sus características más destacables son:

- Una sintaxis simple. Los programas resultan fáciles de leer.
- Fácil de aprender y de usar. Esto hace que sea ideal para la realización de prototipos.
- Una biblioteca estándar dedicada a tareas de servidores y expresiones regulares.
- Se ejecuta en cualquier sistema operativo.
- *Open-source*.

La versión que se utilizará de este lenguaje será la 3.7.6. Para la instalación y uso se instalará Anaconda. Anaconda es un software libre que instala librerías básicas para *data science* y *machine learning*, contiene un manejador de entornos virtuales propio y está disponible para Windows, MacOS y Linux. Además, instala herramientas de edición de código como son Jupyter o Spyder.

#### TensorFlow

Es una librería, desarrollada por Google, de matemática simbólica que incluye aplicaciones de aprendizaje automático como Keras. Google lo usa de herramienta de investigación y de producción. Esta biblioteca está adaptada para usarse con diferentes tipos de unidad de procesamiento ya sea CPU (Central Processing Unit), GPU (Graphics Processing Unit) y TPU (Tensor Processing Unit) siendo este último una unidad de procesamiento desarrollada por el mismo desarrollador que la librería, por tanto, su implementación es especialmente eficiente.

La versión 2.1 de TensorFlow será la elegida para el proyecto.

#### Keras

Keras es una librería dedicada a las redes neuronales. Esta librería fue diseñada para una rápida y fácil experimentación de redes neuronales profundas. En 2017 TensorFlow añadió Keras a su librería.

Existen una sublibrería en Keras donde se encuentran modelos de redes neuronales enfocados en la clasificación de imágenes. Estos modelos son de aplicación a gran escala y realmente relevantes dentro de la investigación de redes neuronales.

Además, se añadirán librerías las cuales implementan modelos de clasificación como EfficientNet y de segmentación como U-Net. Estas librerías están implementadas en Keras, pero son de autores independientes.

### **OpenCV**

Para el tratamiento de imágenes a nivel computacional se utilizará la librería OpenCV para Python. OpenCV es una de las librerías de visión artificial con mayor eficiencia.

Aunque esta librería está enfocada a la visión artificial y *machine learning*, solo se hará uso de lectura de imágenes, escritura de imágenes y creación de máscaras.

### **ImgAug**

ImgAug es una librería dedicada al *data augmentation* de imágenes para *machine learning*. Contiene multitud de funciones para la edición de imágenes.

### **Django**

Django es un *web framework* para Python. Su tarea principal será la facilitación de la creación de la página web. Sus características principales son:

- Servidores web livianos e independientes.
- Sistema de plantillas.
- Sistema de serialización.
- Kit de desarrollo de API REST.

### **FastAPI**

FastAPI es un *web framework* de alto rendimiento dedicado principalmente a la creación de APIs. Se empezó a desarrollar en 2019 por lo que es un software realmente nuevo. Sus principales características son:

- Rápido.
- Intuitivo.
- Fácil.
- Robusto.

Como indican populares pruebas de eficiencia, FastAPI es el *web framework* más rápido actualmente. Por ello será la librería elegida para realizar la API REST.

#### **4.4.2. Herramientas de Soporte**

##### **Visual Studio Code**

Visual Studio Code es un editor de código. Visual Studio Code es *open-source* y compatible con Windows, Linux y MacOS. Este software soporta multitud de lenguajes de programación con resaltado de sintaxis, plegado de código e incluso soporte de IntelliSense, un soporte para mejorar la rapidez de búsqueda de errores.

Para el desarrollo del proyecto este software será la principal herramienta.

##### **GanttProject**

GanttProject se trata de una herramienta para la gestión de proyectos software *open-source* y multiplataforma. Su objetivo principal es la creación de diagramas de Gantt. También permite la generación

de tablas PERT a partir de estos diagramas. Esta herramienta se utilizará para la planificación del proyecto y creación del diagrama de Gantt.

### LaTeX/Overleaf

Overleaf es un editor de LaTeX basado en la nube. Se asocia con una amplia gama de editores científicos para proporcionar plantillas oficiales de LaTeX para revistas y enlaces de envío directo. Permite la colaboración en tiempo real de varios usuarios en un proyecto.

Overleaf será utilizado para la elaboración de esta documentación.

#### 4.4.3. Hardware

En la parte hardware gracias a la empresa Indra se podrá disponer de varias máquinas dedicadas en su arquitectura al entrenamiento de redes neuronales; algo que en un proyecto sin ayuda externa sería imposible por su alto precio. Por lo tanto, se dispondrá de cuatro máquinas, tres de ellas suministradas por la empresa. Las especificaciones de cada máquina se reflejan en la Tabla 4.2.

**Tabla 4.2:** Especificaciones de cada máquina utilizada

	Máquina 1 (Indra)	Máquina 2 (Indra)	Máquina 3 (Indra)	Máquina 4
<b>Procesador</b>	Intel® Core™ i5-8500 CPU 3.00 GHz 6 Cores	Intel® Core™ i5-7500 CPU 3.40 GHz 4 Cores	Intel® Core™ i5-6300U CPU 2.40 GHz 2 Cores	Intel® Core™ i7-9750H CPU 2.60 GHz 6 Cores
<b>Memoria física instalada (RAM)</b>	16.0 GB	16.0 GB	8.0 GB	16.0 GB
<b>Tarjeta gráfica</b>	Nvidia GeForce GTX 1080 8 GB	Nvidia Quadro M4000 16 GB	Intel® HD Graphics 520	Nvidia GeForce RTX 2070 8 GB

El componente más importante en estas máquinas es la tarjeta gráfica y su VRAM (Memoria gráfica). A diferencia de las CPU's las tarjetas gráficas realizan un mayor número de cálculos simultáneamente, pero con una complejidad menor por lo que están orientadas a las matemáticas con matrices. Los cálculos en redes neuronales son operaciones matemáticas matriciales siendo, en este contexto, las tarjetas gráficas, mejores unidades de procesamiento que las CPU's. La mayor capacidad de VRAM permitirá el almacenamiento en memoria de imágenes de grandes dimensiones.

- **Máquinas 1 y 2:** serán las más utilizadas ya que son máquinas dedicadas exclusivamente al uso de *deep learning* tanto en arquitectura como en lugar de instalación.
- **Máquina 3:** es un ordenador portátil a disposición de la empresa. Su uso en el entrenamiento de redes neuronales no será frecuente. Sus características hacen que los entrenamientos de redes complejas duren semanas.
- **Máquina 4:** es un ordenador portátil personal. Las características permitirán el entrenamiento de redes neuronales, pero su lugar de uso no está habilitado para un continuo funcionamiento.

Además de estas máquinas, se dispondrá de una Raspberry Pi para las pruebas y desarrollo de iteraciones. El modelo que se utilizará será Raspberry Pi 3 B+. Este modelo de Raspberry Pi se componen de:

- Procesador: Broadcom BCM2837B0 1.4GHz Quad core A53 64-bit.
- Procesador gráfico: Broadcom Videocore IV.
- RAM: 1GB.

Para las pruebas de captura de imagen se dispondrá de una webcam USB.



---

## CAPÍTULO 5

# RESULTADOS

---

Para conseguir una lógica que detecte humo a partir de imágenes se necesitará hacer varias pruebas de modelos de redes neuronales por lo que en este capítulo se visualizarán resultados y concluirán cuál de ellos será mejor para nuestro caso de uso.

Como es realmente complicado predecir qué resultados obtendremos mediante el uso de redes neuronales profundas para la clasificación o segmentación de imágenes como las que se utilizan actualmente se va a utilizar la metodología de trabajo de la sección 4.1.1.

Todas las pruebas de rendimiento se realizarán en la máquina 4 (Ver tabla 4.2), es decir, en el portátil personal. La duración de estas pruebas dependerá del hardware utilizado.

### 5.1. ITERACIÓN 1 Y 4: DATASET

Como previamente se mencionaba en antecedentes 3, en el funcionamiento de técnicas de *machine learning* y redes neuronales con supervisión los datos de entrenamiento son fundamentales y se debe crear un *dataset* consistente.

Los elementos principales del *dataset* del proyecto son imágenes en las que se visualiza humo y otras donde no se visualice. La extracción de imágenes que no contienen humo es relativamente fácil por la web, en cambio la creación de un *dataset* de un número consistente de imágenes en las que se pueda ver humo ha sido complicada.

Las fotografías a incendios prematuros son escasas por eso se han añadido al *dataset* imágenes con humo procedentes de diferentes situaciones, como por ejemplo automóviles, chimeneas o incluso fumadores (ver Fig. 5.1<sup>1</sup>). Esto tiene ventajas y desventajas. Una ventaja sería que es capaz de aprender más situaciones donde se pueda encontrar el humo y con un poco más de contenido poder realizar un detector de humos en múltiples ámbitos, mientras que la desventaja sería que, en el ámbito de nuestro caso de uso, incendios forestales, no llegará a ser tan preciso.

Otra problemática que ha surgido para la búsqueda de imágenes es el tipo de conjunto de imágenes que se han encontrado. Muchos de estos tipos no son completamente válidos por la duplicación de imágenes en el *dataset* o simplemente porque se centran en *frames* de vídeos. Este hecho puede suponer el sobreaprendizaje del modelo por lo que se debe intentar de no añadir imágenes iguales en el *dataset* y de abordar todos los posibles casos. Todo esto se ha ido resolviendo hasta crear un *dataset* con 2496 de imágenes con humo. Para completar el *dataset* se han añadido otras 2496 imágenes que no contienen humo. La escala 1:1 del número de imágenes sobre si contiene humo o no, es la más recomendada.

Respecto a las dimensiones de las imágenes las redes neuronales necesitan una dimensión determinada para todas las imágenes que se procesan en la red, pero las imágenes encontradas son de

---

<sup>1</sup>Las imágenes han sido generadas por la librería Matplotlib de Python.

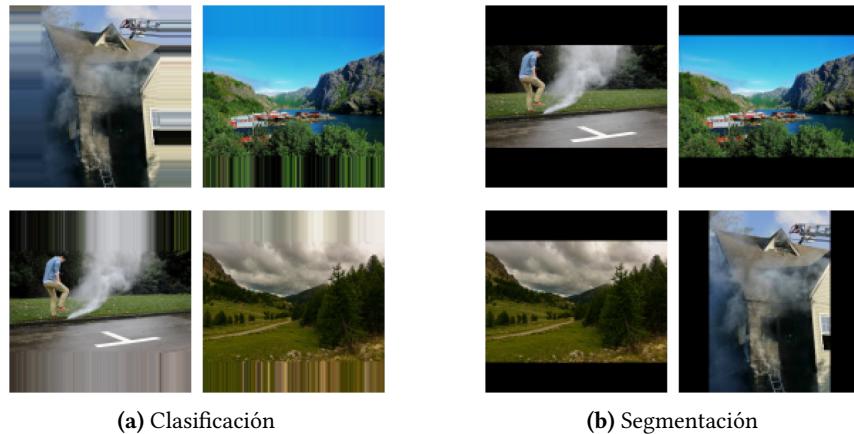


**Figura 5.1:** Imágenes elegidas aleatoriamente de todo el *dataset*

diferentes dimensiones. Por la ya anunciada escasez de imágenes con humo esto se resuelve haciendo una transformación en la dimensión de las imágenes que dependerá de si es para clasificación o segmentación. Por generalizar se realizará imágenes cuadradas, es decir, mismo ancho y largo.

- **Para clasificación** (ver Fig. 5.2a) se realizará una redimensionación a partir del píxel más cercano al borde. Primero se realiza una redimensionación a escala sobre su lado más grande. Después se dibujará por el lado opuesto con el color del píxel del borde correspondiente. Para no hacer crecer la diferencia entre clases por la frecuencia de imágenes no cuadradas se evita utilizar redimensiones con bordes de colores constantes y la redimensionación por píxel más cercano servirá de aleatoriedad.

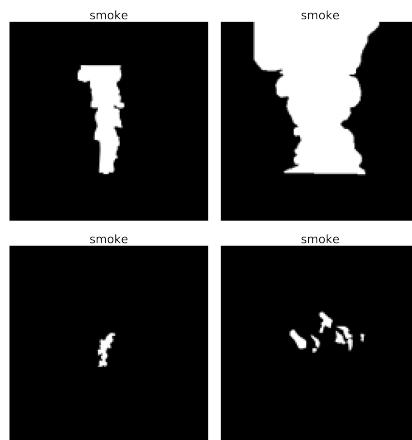
- Para segmentación (ver Fig. 5.2b) se realizará una redimensión con borde negro. Primero se realiza una redimensión a escala sobre su lado más grande igual que en clasificación. Después se dibujará por el lado opuesto con el color negro. Como el humo que posteriormente se segmentará no será un negro puro ni tendrá dimensiones cuadradas o rectangulares, esta redimensión será más eficaz que los píxeles más cercanos al borde. Además, las máscaras no se verán afectadas gracias a esta redimensión.



**Figura 5.2:** Imágenes redimensionadas

Una vez obtenida todas estas imágenes hay que dar un etiquetado. Existirá una gran diferencia de etiquetado entre un *dataset* para clasificación y segmentación.

- Clasificación:** Cada imagen del *dataset* tendrá dos posibilidades de clase: *humo* o *no humo*; *no humo* también se llamará neutral. Para su creación basta con separar las imágenes en carpetas que posteriormente el *script* diferenciará entre si la etiqueta es humo o neutral.
- Segmentación:** Para saber la ubicación del humo necesitamos una imagen resultado que nos proporcione los píxeles donde se encuentra el humo la cual se llamará máscara (ver Fig. 5.3). Como ya se abordó en el capítulo 3, las máscaras son imágenes de la misma dimensión que su imagen asociada en blanco y negro teniendo únicamente dos valores 0 (negro) y 255 (blanco). El valor 255 representa el valor donde se ubica el humo. Para la creación de estas máscaras, el 65 % se ha realizado de forma manual con la herramienta LabelMe<sup>2</sup> y el resto han sido extraídos de internet. Por la dificultad de creación de este *dataset* se verá reducido el número de elementos a 705 imágenes con humo segmentado. Además, se añadirán otras 705 imágenes sin humo con la máscara completamente en negro.

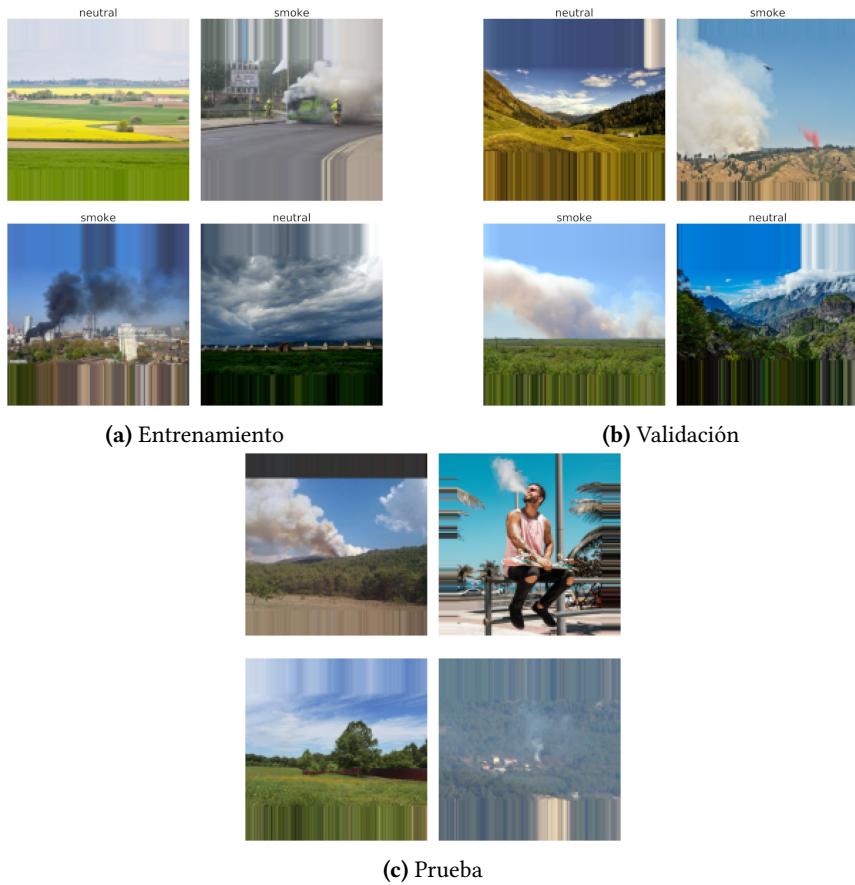


**Figura 5.3:** Máscaras ejemplo del *dataset* de segmentación

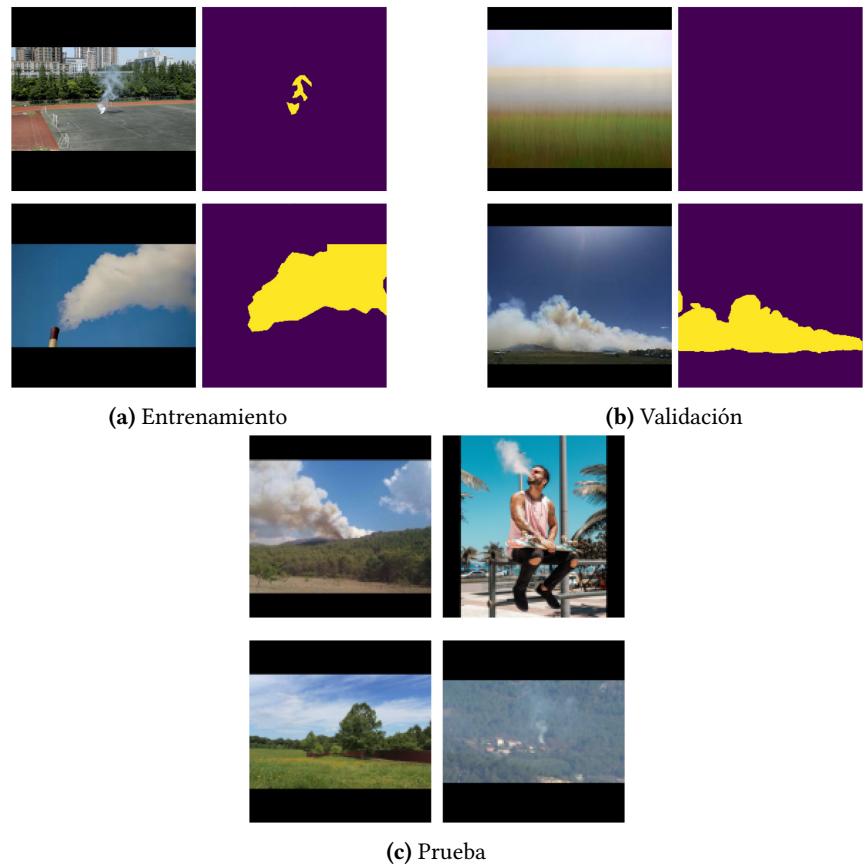
<sup>2</sup><https://github.com/wkentaro/labelme>

Para abordar los conjuntos de datos para la aplicación de técnicas de *deep learning*, se diferencian 3 grupos de imágenes:

- **Entrenamiento:** este conjunto hace variar los pesos de la red neuronal por la métrica que se calcula por el resultado de la red neuronal y el resultado real. El número de imágenes en este conjunto varía si es para clasificación (ver Fig. 5.4a): 2048 de humo y 2048 de neutral, o para segmentación (ver Fig. 5.5a): 1175 con humo segmentado.
- **Validación:** lo habitual en estas técnicas es equilibrar el tipo de imagen de entrenamiento y validación, pero para aumentar la robustez del caso de uso del proyecto en este conjunto hay un mayor porcentaje de imágenes relacionadas con paisajes sin humo e incendios forestales. Al igual que en entrenamiento, el número de imágenes varía si es para clasificación (ver Fig. 5.4b): 448 de humo y 448 de neutral, o para segmentación (ver Fig. 5.5b): 235 con humo segmentado.
- **Prueba:** está formado por 64 imágenes sin clasificación ni segmentación previa. Este conjunto es común para clasificación y segmentación y es el encargado de dar un resultado visual a las pruebas de los modelos entrenados (ver Fig. 5.4c y Fig. 5.5c).



**Figura 5.4:** Imágenes ejemplo del conjunto para clasificación



**Figura 5.5:** Imágenes ejemplo del conjunto para segmentación

Para concluir la creación del *dataset* se muestra datos técnicos en la tabla 5.1.

**Tabla 5.1:** Datos técnicos del *dataset*

	Tamaño (Bytes)	Nº archivos	Dimensión (Píxeles)	
			Máximo	Mínimo
<b>Total</b>	<b>1133 MB</b>	<b>7940 archivos</b>	<b>9498x3156</b>	<b>117x150</b>
Clasificación	956 MB	5056 archivos	9498x3156	117x150
Prueba	42 MB	64 archivos	5616x3744	256x256
Entrenamiento	639 MB	4096 archivos	6000x4000	117x150
Neutral	554 MB	2048 archivos	2816x2120	250x167
Humo	85 MB	2048 archivos	6000x4000	117x150
Validación	275 MB	896 archivos	9498x3156	96x203
Neutral	218 MB	448 archivos	9498x3156	256x256
Humo	57 MB	448 archivos	6488x4325	96x203
Segmentación	177 MB	2884 archivos	5616x3744	97x200
Prueba	42 MB	64 archivos	5616x3744	256x256
Entrenamiento	117 MB	2350 archivos	2048x512	97x200
Images	115 MB	1175 archivos	2048x512	97x200
Labels/humo	2 MB	1175 archivos	2048x512	97x200
Validación	18 MB	470 archivos	1920x1080	177x284
Images	17.5 MB	235 archivos	1920x1080	177x284
Labels/humo	0.5 MB	235 archivos	1920x1080	177x284

## 5.2. ITERACIÓN 2: CLASIFICACIÓN DE IMÁGENES

Las métricas [2] para la clasificación de imágenes que se utilizan en los entrenamientos son el porcentaje de acierto y la pérdida. Aunque se divida las clases humo y neutral se podría decir que realmente es humo y no humo por lo que se usará la alternativa binaria para los modelos de redes neuronales, es decir, la salida nos proporcionaría un valor de 0 a 1 siendo 1 humo y 0 neutral. De esta forma se evita realizar un preproceso de los datos como *one hot encoding* (preproceso para pasar de diferentes valores en una columna, a varias columnas para cada valor posible en forma booleana) por lo que se realizará menos cálculos.

La pérdida es el resultado de la entropía cruzada entre etiquetas verdaderas y etiquetas predichas. La función de pérdida es:

$$\begin{aligned} EReal &: \text{Etiqueta real} \\ EResul &: \text{Etiqueta resultado} \\ Perdida(EReal, EResul) &= \max(EResul, 0) - EReal * EResult + \log(e^{-|EResul|}) \end{aligned} \quad (5.1)$$

Por cada valor que sea procesado por la red se calculará su pérdida y se sumará a la pérdida previa siendo inicialmente 1.

La precisión es el resultado de calcular con qué frecuencia coinciden la etiqueta predicha con la real. La función que realiza simplemente divide el total de aciertos entre el total de predicciones. La función de precisión es similar a la función de pérdida, excepto que los resultados de la evaluación de la precisión no se usan al entrenar el modelo. Para el caso de uso el resultado de estos entrenamientos debe ser de un alto nivel de precisión y en caso de error preferir falsos positivos que falsos negativos.

Existen otras métricas enfocadas en la clasificación como realizar la media de la pérdida o el error medio, pero no servirán para este caso.

### 5.2.1. Entrenamientos

Se han elegido los modelos de redes neuronales ResNet152V2, EfficientNetB6 y InceptionResNetV2 para hacer las primeras pruebas de entrenamiento. Son modelos con alta precisión en el reto ImageNet como se muestra en la gráfica 3.4. Además, son modelos con un reducido número de parámetros y bastante ligeros comparados con otros modelos populares de clasificación de imágenes.

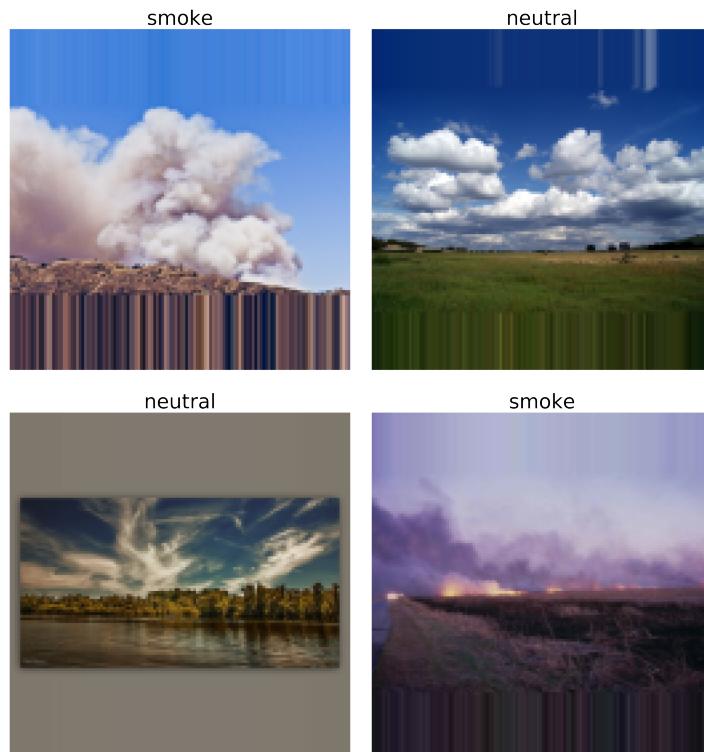
Aunque en el reto de ImageNet el orden de precisión es EfficientNetB6 > InceptionResNetV2 > ResNetV2, en este caso no tiene por qué ser así.

Para estos entrenamientos se usa un *data augmentation* básico, pero posteriormente se probarán más tipos. Este *data augmentation* básico (ver Fig. 5.6) consta de rotaciones en la imagen, variaciones en la escala, dar la vuelta tanto horizontal como vertical, variación de brillo y una pequeña variación de color. Se elige estos parámetros porque dará mayor profundidad al *dataset*, sin aprender de forma errónea y ajustará la curva de aprendizaje para evitar el sobreaprendizaje.

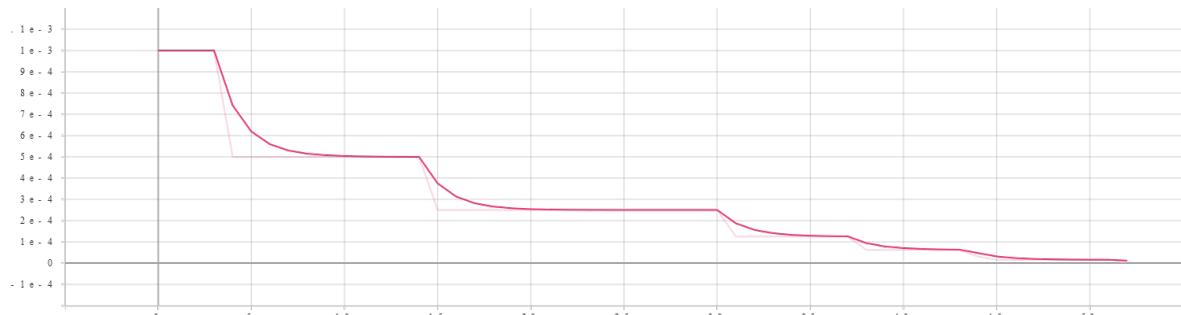
Además, se utilizará un algoritmo para reducir el *learning rate* si la predicción del conjunto de validación no mejora (Ver Fig. 5.7<sup>3</sup>). Tal que, si cada 4 épocas la precisión de validación no mejora, se reduce el *learning rate* a la mitad y si pasan 16 épocas sin mejorar, se reducirá hasta un cuarto. Esto ayudará a los entrenamientos para que no sobreaprendan y poder llegar a una precisión más alta.

Primero se realizará el entrenamiento con el modelo ResNet152V2. Las gráficas 5.8 muestran como existe un sobreaprendizaje rápido. La línea azul es entrenamiento y la línea roja validación. Desde la

<sup>3</sup>Las gráficas han sido extraídas de la herramienta Tensorboard de Tensorflow.

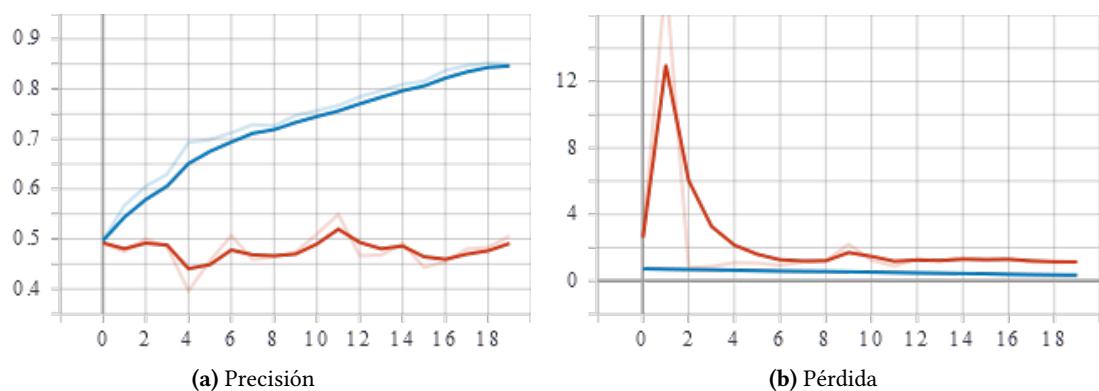


**Figura 5.6:** Imágenes ejemplo con *data augmentation* básico



**Figura 5.7:** Gráfica ejemplo de la reducción del *learning rate* en un entrenamiento

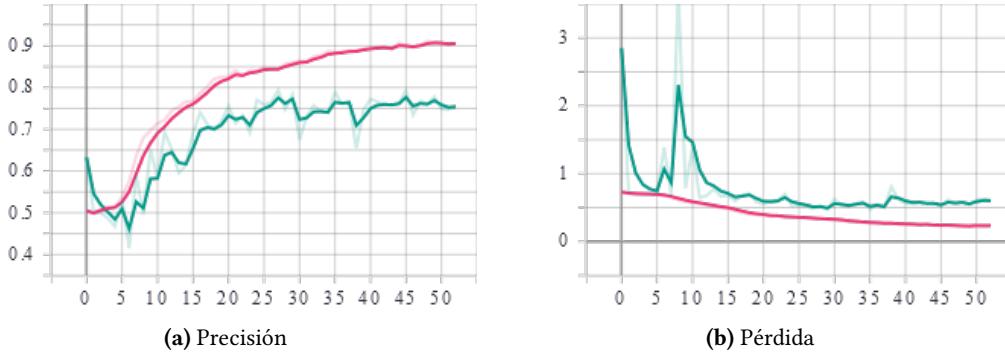
época 12 la precisión de validación cae por el aprendizaje incorrecto. La mejor época coincide con la caída de la precisión. Tiene una precisión del 54,91 %. Este resultado será poco óptimo ya que está cercano al 50 %.



**Figura 5.8:** Gráficas de las métricas del entrenamiento del modelo ResNet

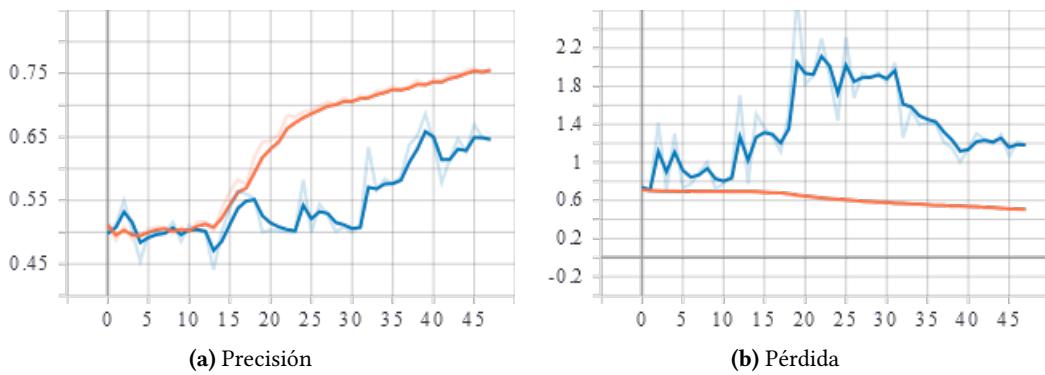
A continuación, se probará el modelo InceptionResNetV2. Las gráficas 5.9 muestran un crecimiento

a la par, una distribución sorprendentemente buena. Se puede deducir que la combinación con bloques Inception reduce la probabilidad de sobreaprendizaje. La línea rosa es entrenamiento y la línea verde validación. Se puede ver la aleatoriedad de los bloques Inception al principio de las épocas como la precisión de validación es incluso más alta que la de entrenamiento, pero acaba reduciéndose y siguiendo el crecimiento por debajo de sus valores. La mejor precisión que alcanzada el modelo es 79,35 % en la época 28. El resultado es aceptable pero posteriormente se realizarán pruebas de falsos negativos.



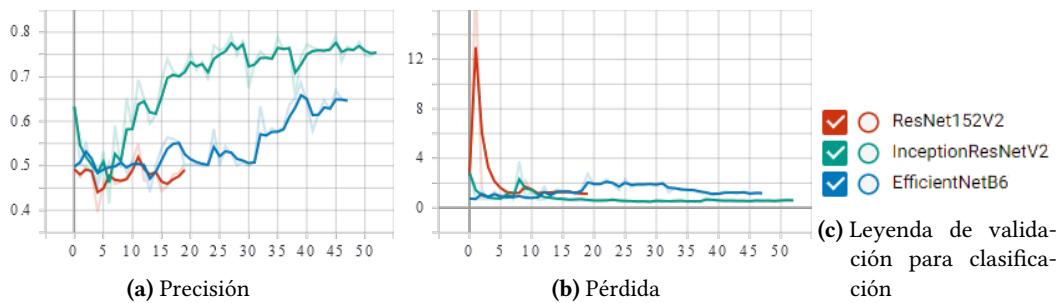
**Figura 5.9:** Gráficas de las métricas del entrenamiento del modelo InceptionResNetV2

Por último, se realizará el entrenamiento con el modelo EfficientNetB6. Las gráficas 5.10 muestran una distribución irregular, pero mantiene la ausencia de sobreaprendizaje temprano. La línea naranja es entrenamiento y la línea azul validación. La mejor precisión que alcanza el modelo es 68,64 % en la época 40. Sorprendentemente es un resultado bastante menor que InceptionResNetV2.



**Figura 5.10:** Gráficas de las métricas del entrenamiento del modelo EfficientNetB6

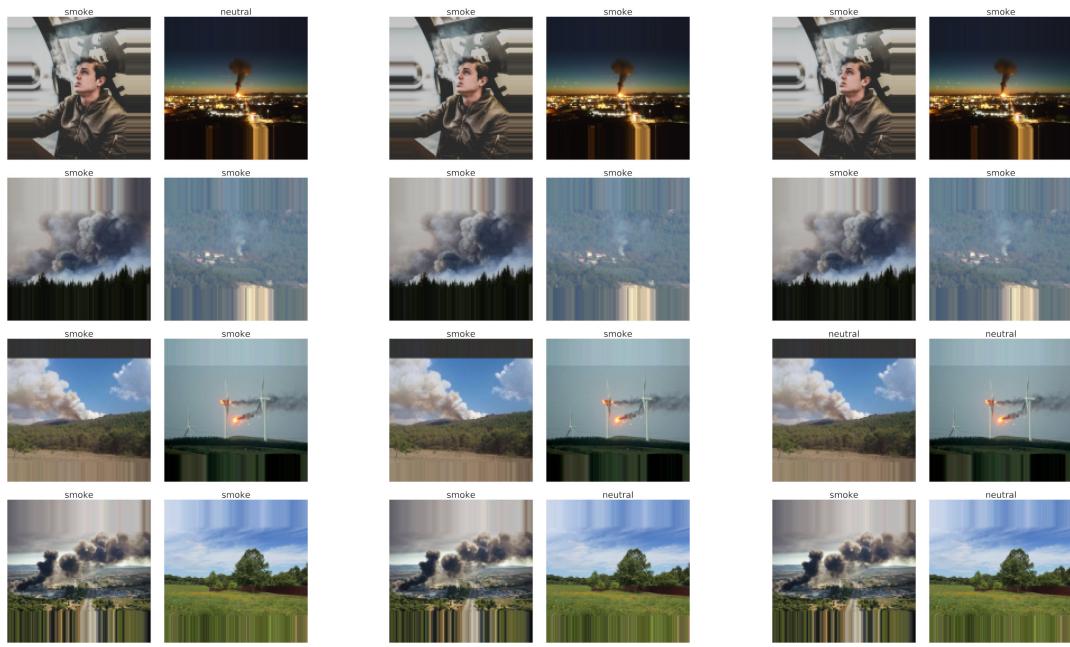
Se puede observar que la diferencia entre modelos es realmente amplia. El tipo de modelo es el parámetro más importante de los entrenamientos. Se muestra la diferencia de precisiones y perdidas en la gráfica 5.11.



**Figura 5.11:** Gráficas de las métricas de validación de los modelos usados para clasificación

Para una visión más intuitiva se ha realizado una predicción de diferentes imágenes del conjunto de Prueba con cada modelo.

- ResNet152V2 (Ver Fig. 5.12a): se puede observar que falla tanto para detectar humo y no humo. Tarda 0,293 segundos en predecir una imagen.
- InceptionResNetV2 (Ver Fig. 5.12b): el resultado es 100 % certero, pero no implica que con un número mayor de ejemplos sea igual de efectivo. Tarda 0,239 segundos en predecir una imagen.
- EfficientNetB6 (Ver Fig. 5.12c): se puede ver como falla en el reconocimiento de humo. El modelo tarda en predecir 0,158 segundos, siendo el más rápido.



(a) Resultados de ResNet152V2

(b) Resultados de InceptionResNetV2

(c) Resultados de EfficientNetB6

**Figura 5.12:** Imágenes resultado de los modelos entrenados para clasificación

### 5.2.2. Conclusión

También se realiza una prueba de predicción exclusiva para comprobar los falsos negativos. InceptionResNetV2 tiene una precisión de 79,17 % en situaciones de humo por lo que un 20,83 % no es capaz de detectar humo donde existe. EfficientNetB6 tiene una precisión del 83,73 % por tanto, aunque tenga una precisión menor general, es capaz de tener menos falsos negativos. Aun así, EfficientNetB6 tiene una precisión de 55,65 % en situaciones sin humo, siendo una precisión baja.

Aunque aún quede por variar el parámetro *data augmentation*, los resultados no son lo suficientemente precisos y una variación de éste no implicará una gran mejora. Por la dependencia de modelos de clasificación en los modelos de U-Net para segmentación se probará alternativas similares en segmentación de imágenes.

Remarcando que una de las razones por la que no alcanza altas precisiones es la calidad y tamaño del *dataset*.

### 5.3. ITERACIÓN 3: API REST PARA CLASIFICACIÓN

Estas predicciones han sido realizadas con la API REST desarrollada para la predicción masiva de imágenes para clasificar junto con un cliente ligero por comandos. Tanto la API REST como el cliente tienen asociado un archivo de configuración donde para API REST se puede introducir: el directorio principal donde se extraerán los modelos y se guardarán imágenes clasificadas por la predicción, dirección del *host* y puerto; y para el cliente se puede introducir: la dirección de la carpeta de imágenes para predecir y donde se guardarán las predicciones, el nombre de los modelos que se desearán utilizar junto a sus clases, la dirección del *host* y el puerto, estos últimos deberán coincidir con las de la API REST. El código de predicción se puede ver en el listado 5.1.

**Listado 5.1:** Código fuente de la predicción de una imagen en la API REST para clasificación

```

1 # Recibe la imagen serializada y manda predicción sobre clasificación
2 @app.post("/image-classification")
3 async def uploadImageClassification(imageRequest: ImageJSON):
4     # Decodifica la imagen previamente serializada
5     image = cvtColor(imdecode(frombuffer(
6         b64decode((imageRequest.image).encode('ascii')), ←
7             uint8), 1), COLOR_BGR2RGB)
8
9     # Prepara la imagen con una correcta redimensión
10    imageResized = ←
11        resizeClassification([int(gl.Model.input.shape[1]), int(
12            gl.Model.input.shape[2])]).augment_image(image)
13
14    model = gl.Model
15    imagePredicted = array([imageResized/255.0]).astype(float32)
16
17    # Predice la imagen calculando el tiempo
18    before = datetime.now()
19    prediction = model.predict(imagePredicted, batch_size=1)
20    after = datetime.now()
21
22    # Comprueba que clase ha predicho
23    if len(gl.ClassNames) > 2:
24        predictClass = gl.ClassNames[argmax(prediction[0], axis=1)]
25    else:
26        predictClass = gl.ClassNames[0 if prediction[0][0] < ←
27            0.5 else 1]
28
29    # Guarda la imagen original clasificada en el servidor
30    classDir = join(predictionsDir, gl.ModelName, predictClass)
31    imwrite(join(classDir, imageRequest.filename),
32            cvtColor(image, COLOR_RGB2BGR))
33
34    # Prepara la respuesta al cliente
35    imageDict = imageRequest.dict()
36    imageDict.update({"model": gl.ModelName})
37    imageDict.update({"prediction": predictClass})
38    imageDict.update({"time": (after - before).total_seconds()})
39
40    return imageDict

```

## 5.4. ITERACIÓN 5: SEGMENTACIÓN DE IMÁGENES

Las métricas para la segmentación de imágenes que se utilizarán en los entrenamientos son IOUScore [23] como porcentaje de acierto y la perdida Focal+Dice. Como solo se realizará la segmentación de una categoría se utilizará la alternativa binaria de la perdida Focal+Dice. Esta pérdida es popular en los modelos de segmentación de imágenes por sus resultados.

La pérdida híbrida suma los valores de perdida de Focal y Dice. Las fórmulas de estas pérdidas son:

$$\begin{aligned} MReal &: \text{Mascara real} \\ MResul &: \text{Mascara resultado} \\ N &: \text{numero de pixeles} \end{aligned} \quad (5.2)$$

$$FocalPerdida(MReal, MResul) = \frac{\sum_{p=1}^N -MReal_p * 0,25 * (1 - \minmax(MResul_p, 0, 1))^2 * \log(\minmax(MResul_p, 0, 1)) - (1 - MReal_p) * (1 - 0,25) * \minmax(MResul_p, 0, 1)^2 * \log(1 - \minmax(MResul_p, 0, 1))}{N} \quad (5.3)$$

$$DicePerdida(MReal, MResul) = 1 - \frac{\sum_{p=1}^N \frac{2 * MResul_p - MReal_p * MResul_p}{3 * MResul_p - 3 * MReal_p * MResul_p + MReal_p}}{N} \quad (5.4)$$

$$FocalDicePerdida(MReal, MResul) = FocalPerdida(MReal, MResul) + DicePerdida(MReal, MResul) \quad (5.5)$$

Por cada valor que sea procesado por la red se calculará su pérdida y se sumará a la pérdida previa siendo inicialmente 1.

La precisión será correspondiente al índice de Jaccard o también llamado intersección sobre la unión y el coeficiente de similitud de Jaccard. Se usa para comparar la similitud y diversidad de conjuntos de muestras con valores múltiples. La fórmula del índice de Jaccard es:

$$\begin{aligned} MReal &: \text{Mascara real} \\ MResul &: \text{Mascara resultado} \end{aligned} \quad (5.6)$$

$$Jaccard(MReal, MResul) = \frac{MReal \cap MResul}{MReal \cup MResul}$$

### 5.4.1. Entrenamientos

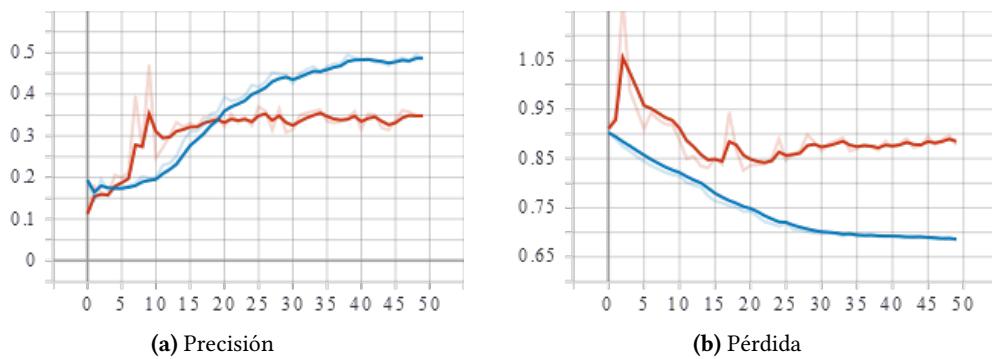
Como modelo de segmentación de imágenes se ha elegido el modelo de red neuronal U-Net. Como se explica en la sección 3.2.2 este modelo primero realiza una codificación del vector hasta llegar a un vector del tamaño de las categorías a clasificar y después decodifica el vector hasta llegar a la dimensión original, pero con la profundidad del número de categorías. Esta red neuronal nos permite elegir un modelo de clasificación como parte de convolución. Esta parte de la red se llamará *backbone*. También nos permite variar la forma de decodificación de las imágenes ya sea *upsampling* o *transpose*.

Para estos entrenamientos se usa el *data augmentation* básico anteriormente citado en clasificación (ver Fig. 5.6).

### Backbone

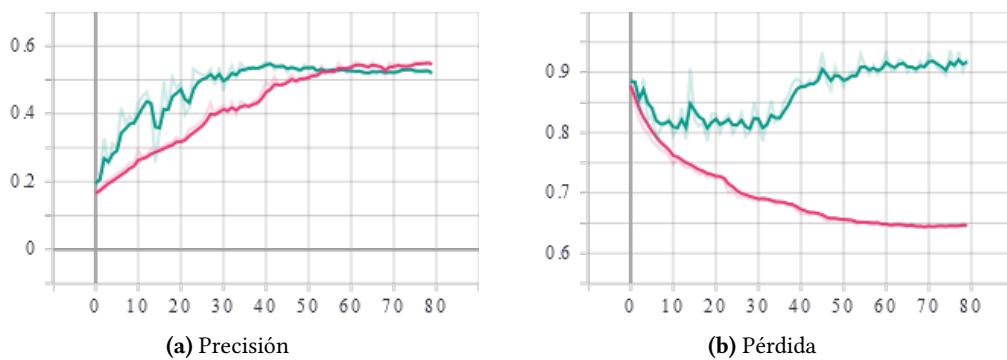
Se realizan pruebas variando el *backbone* con los modelos de clasificación InceptionResNetV2 y EfficientNetB6. Estos modelos dieron los mejores resultados en la parte de clasificación.

El entrenamiento con el *backbone* InceptionResNetV2. Las gráficas 5.13 muestran como existe un sobreaprendizaje rápido y existe la aleatoriedad de Inception que provoca la mejora rápida del resultado de validación. La línea azul es entrenamiento y la línea naranja validación. La mejor época es la 10. Tiene una precisión del 47,04 %. Aunque hay que destacar que esta precisión no describe si la red es capaz de detectar o no humo en una imagen ya que es la precisión por píxel clasificado. Por tanto, puede ser que no sea capaz de segmentar todo el humo correctamente, pero si es capaz de detectar que el humo existe en la imagen.



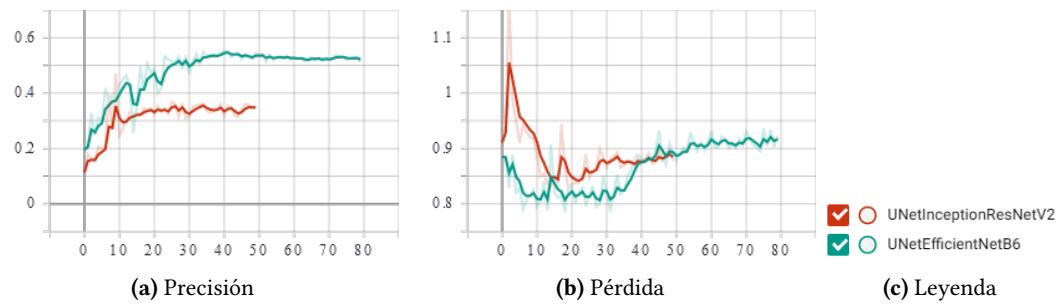
**Figura 5.13:** Gráficas de las métricas del entrenamiento del modelo U-Net con *backbone* InceptionResNetV2

Por último, se realizará el entrenamiento con el *backbone* EfficientNetB6. Las gráficas 5.14 muestran una distribución continua y mantiene la ausencia de sobreaprendizaje temprano. La línea púrpura es entrenamiento y la línea verde validación. La mejor precisión que alcanza el modelo es 55,27 % en la época 41. Sorprendentemente es un resultado mayor que InceptionResNetV2 el cual previamente resultó ser más preciso en clasificación.



**Figura 5.14:** Gráficas de las métricas de los resultados del modelo U-Net con *backbone* EfficientNetB6

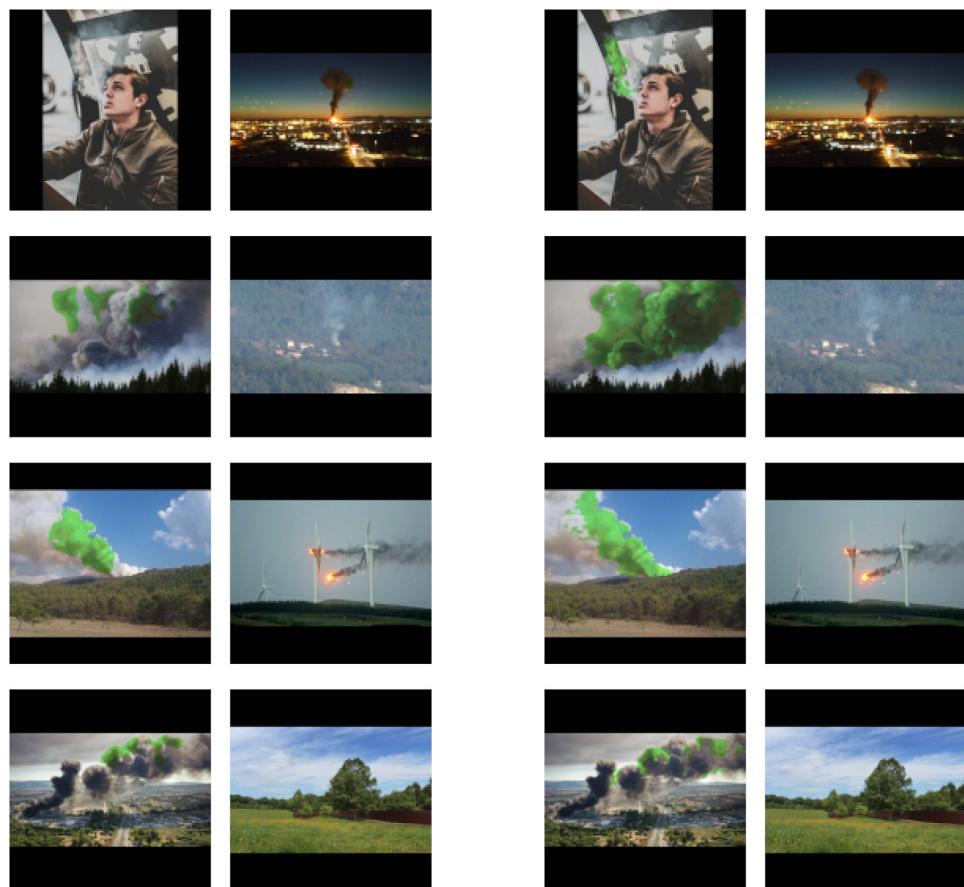
Se puede observar que la diferencia entre modelos es realmente amplia. El tipo de *backbone* es el parámetro más importante de los entrenamientos en segmentación. Se muestra la diferencia de precisiones y perdidas en la gráfica 5.15.



**Figura 5.15:** Gráficas de las métricas de los resultados con diferentes *backbones*

Para una real detección de humo cuando exista un píxel segmentado de humo se considerará que existe humo en la imagen. Se ha realizado una prueba para prever falsos negativos, además de una prueba de eficiencia con el tiempo que tarda en predecir una imagen. El humo detectado en las pruebas se visualizará de color verde.

- InceptionResNetV2 (Ver Fig. 5.16a): El porcentaje de humo detectado es de 69,55 %. Un resultado menor que en clasificación. Tarda 0,334 segundos en predecir una imagen.
  - EffientNetB6 (Ver Fig. 5.16b): El porcentaje de humo detectado es de 64,82 %. Un resultado también menor que en clasificación, pero mejora en precisión general con un 73,3 %. El modelo tarda en predecir 0,1863 segundos, siendo el más rápido como en clasificación.



**(a)** Resultados de *backbone* InceptionResNetV2

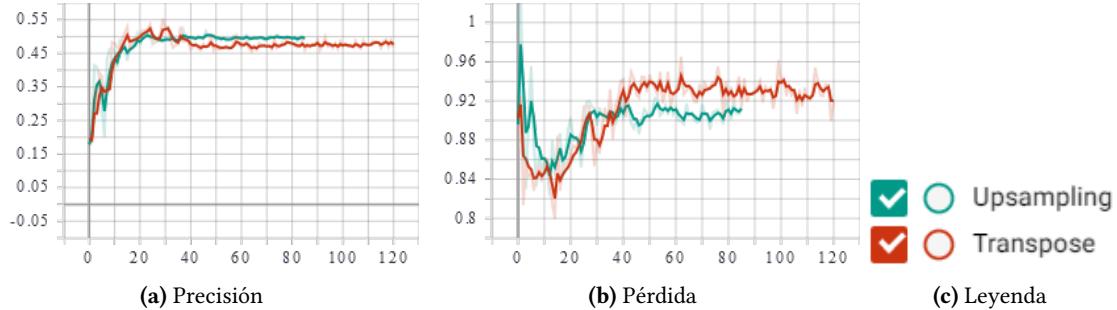
**(b)** Resultados de *backbone* EfficientNetB6

**Figura 5.16:** Imágenes resultado de los modelos entrenados con diferentes *backbones*

### Decodificador

Aunque está característica no supondrá una gran mejora en la precisión se probarán los posibles valores con el fin de realizar un análisis de esta variación para futuros entrenamientos. Además, se probará un *backbone* nuevo para posibles mejoras.

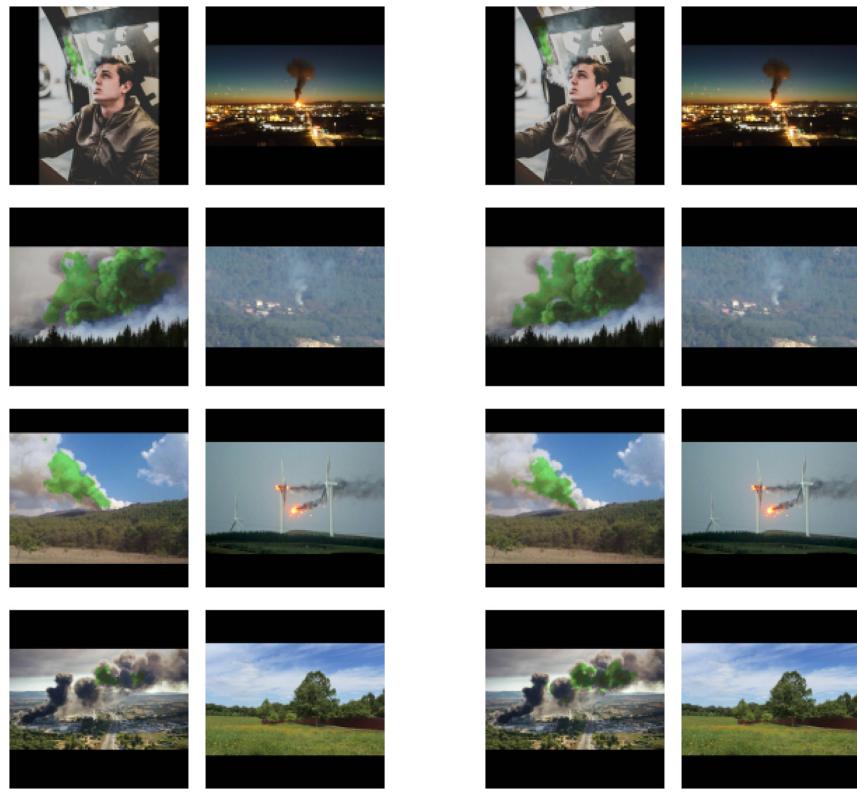
Se ha elegido el *backbone* EfficientNetB4 para aumentar la eficiencia de la red. Se puede observar que la variación de decodificadores no es realmente relevante, pero como se ve en la gráfica 5.17 el decodificador en *transpose* da un resultado más preciso. El resultado es esperado ya que el tipo *transpose* tiene más capas entrenables que *upsampling*.



**Figura 5.17:** Gráficas de las métricas de validación con diferentes decodificadores

Se realizan pruebas de falsos positivos. Por otro lado, el tipo de decodificador no afectará al rendimiento de manera considerable.

- *Upsampling* (Ver Fig. 5.18a): El porcentaje de humo detectado es de 55,17 %.
- *Transpose* (Ver Fig. 5.18b): El porcentaje de humo detectado es de 66,47 % siendo claramente la mejor alternativa de decodificador.

(a) Resultados de decodificador *upsampling*(b) Resultados de decodificador *transpose***Figura 5.18:** Imágenes resultado de los modelos entrenados con diferentes decodificadores

### Data Augmentation

Al igual que la decodificación está característica no va a proporcionar una mejora considerable, pero se probará para entrenamientos futuros. También se entrenará con un *backbone* nuevo, esta vez con EfficientNetB0 y una decodificación en *transpose*.

Se ha elegido el *backbone* EfficientNetB0 para optimizar el tiempo de entrenamiento ya que es la red más eficiente de todas las variaciones de EfficientNet.

La *data augmentation* puede variar ilimitadamente, sin embargo, solo existirán un cierto tipo de filtros para mejorar el aprendizaje del entrenamiento en un caso de uso específico. Se debe elegir los filtros que realmente añadan información por lo que serán filtros que hagan imágenes que se vayan a añadir en el conjunto de entrenamiento. La razón por la que se utiliza esta técnica y no simplemente se aumenta el número de elementos del *dataset* con elementos previos editados es porque se añade aleatoriedad al entrenamiento y ayuda a combatir el sobreaprendizaje.

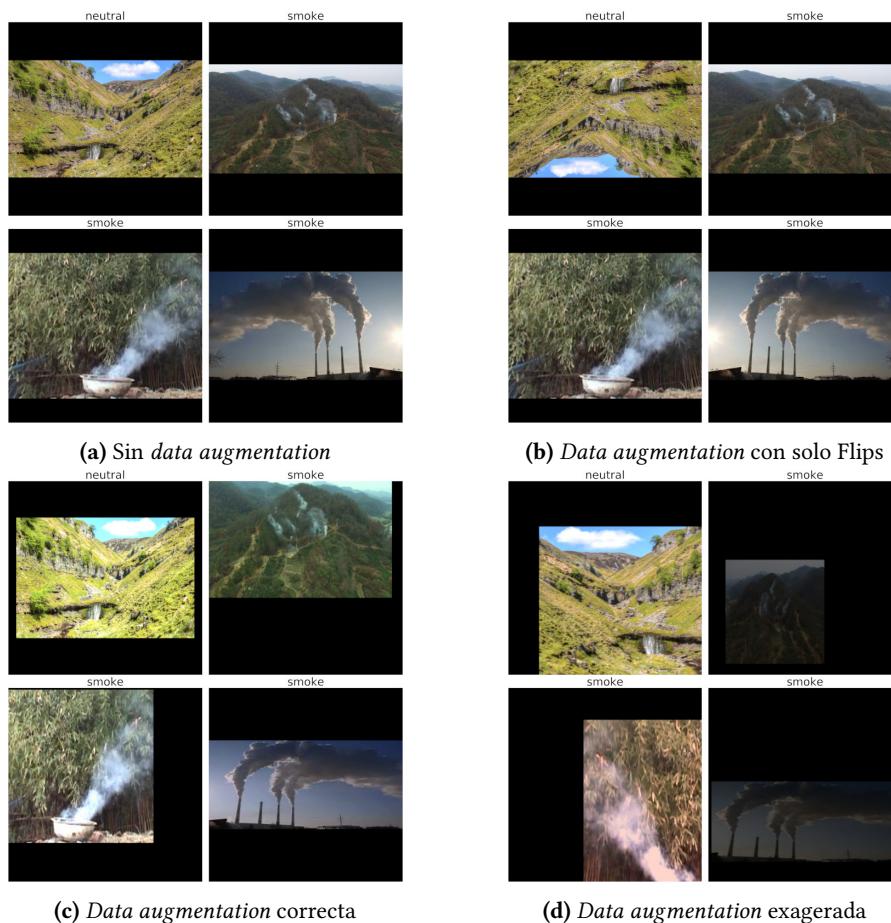
Una de las pautas que se debe seguir a la hora de elegir una correcta *data augmentation* es que las nuevas imágenes generadas puedan considerarse una nueva imagen en el *dataset* y que esté no implique una duplicación de otras. Otra alternativa para utilizar la *data augmentation* es añadir ruido, es decir, aportar confusión al *dataset*. Esto para un pequeño porcentaje de las imágenes se puede utilizar con una red que tenga facilidad para sobreaprender, por lo que provocará una reducción del sobreaprendizaje y mejorará el entrenamiento.

En el caso de humo, como se ha visualizado anteriormente, no existe un sobreaprendizaje rápido ni brusco. Por tanto, la *data augmentation* se usará para ampliar el *dataset*. Para aumentar esa variedad de características del humo se deberá variar: el brillo de tal forma que simule una menor o mayor intensidad de luz; débilmente el tono de color para considerar otros formatos de cámara, el color de la luz o el material que se está quemando; y de múltiples formas la estructura de la imagen ya sea

en rotación, escala, traslación o dar la vuelta. Como ya se comentó las variaciones son ilimitadas y quizás no se estén utilizando todas las apropiadas para una *data augmentation* óptima.

Se van a comparar 4 tipos de *data augmentation*:

- Sin *data augmentation* (Ver Fig. 5.19a): no usará ningún filtro.
- Flips (Ver Fig. 5.19b): Se voltearán imágenes con un 50 % de probabilidad de forma horizontal y con un 12,5 % de forma vertical.
- Correcta acorde con el caso de uso (Ver Fig. 5.19c): Se realizarán varios filtros:
  - Volteo horizontal con un 50 % de probabilidad
  - Volteo vertical con un 12,5 % de probabilidad.
  - Variación en +-37,5 % de la iluminación.
  - Variación en +-12,5 % en el valor de colores.
  - Escalación de la imagen en +-25 % con una probabilidad del 87,5 %.
  - Traslación de la imagen en +-25 % con una probabilidad del 87,5 %.
- Exagerada (Ver Fig. 5.19d): Se realizarán varios filtros:
  - Volteo horizontal con un 50 % de probabilidad
  - Volteo vertical con un 50 % de probabilidad.
  - Variación en +-75 % de la iluminación.
  - Variación en +-25 % en el valor de colores.
  - Escalación de la imagen en +-50 %.
  - Traslación de la imagen en +-50 %.



**Figura 5.19:** Imágenes ejemplo de diferentes *data augmentations* usados para el entrenamiento

La generación de la *data augmentation* para la parte de segmentación ha sido modificada drásticamente para su correcto funcionamiento. Parte del código de generación de imágenes con *data*

*augmentation* se puede ver en el listado 5.2.

**Listado 5.2:** Código fuente de la generación de imágenes y máscara con *data augmentation* para su uso en entrenamiento de segmentación

```

1  # Da formato correcto a la máscara
2  def prepareLabels(indexes, imagesLabels):
3      imagesLabelsList = []
4      # Cambia el formato (tamaño del batch, número de clases, dimensión X, dimensión Y, 1)
5      # a (tamaño del Batch, dimensión X, dimensión Y, número de clases)
6      for k in range(len(indexes)):
7          img = None
8          for i in range(len(imagesLabels)):
9              imgLabel = imagesLabels[i][k]
10             if img is None:
11                 img = zeros(
12                     (imgLabel.shape[0], imgLabel.shape[1], ↪
13                      ↪ len(imagesLabels)), uint8)
14                 for x in range(len(imgLabel)):
15                     for y in range(len(imgLabel[x])):
16                         img[x][y][i] = imgLabel[x][y]
17             # Transforma los valores de 0-255 a 0.0-1.0
18             imagesLabelsList.append(img/255.0)
19             # Devuelve las máscaras convertidas a lista de floats
20             return array(imagesLabelsList).astype('float32')
21
22
23 class DataGenerator(Sequence):
24
25     def __getitem__(self, index):
26         indexes = self.indexes[index * self.batchSize: (index + 1) * ↪
27                                     ↪ self.batchSize]
28
29         # Lee como imágenes las direcciones de ficheros que contiene el conjunto de datos
30         images = [cvtColor(imread(str(self.imagePaths[k][0])), COLOR_BGR2RGB) for k in indexes]
31         imagesLabels = [[imread(str(l[k][0]), IMREAD_UNCHANGED) for k in indexes] for l in ↪
32                         ↪ self.labelPaths]
33
34         if self.resize != None:
35             # Realiza la redimensión correspondiente tanto a la imagen como a las
36             # máscaras
37             images = self.resize.augment_images(images)
38             imagesLabels = [[lsi.get_arr() for lsi in ls] for ↪
39                             ↪ ls in [self.resize.augment_segmentation_maps(
40                                 [SegmentationMapsOnImage(l[k], ↪
41                                     ↪ shape=l[k].shape) for k in ↪
42                                     ↪ range(len(indexes))]) for l in imagesLabels]]
43
44         if self.augmenters != None:
45             # Determina el data augmentation para que sea el mismo para cada imagen y
46             # sus correspondientes máscaras
47             self.augmenters = self.augmenters.to_deterministic()
48             images = self.augmenters.augment_images(images)
49             imagesLabels = [[lsi.get_arr() for lsi in ls] for ↪
50                             ↪ ls in [self.augmenters.augment_segmentation_maps(
51                                 [SegmentationMapsOnImage(l[k], ↪
52                                     ↪ shape=l[k].shape) for k in ↪
53                                     ↪ range(len(indexes))]) for l in imagesLabels]]
54
55         images = array([img/255.0 for img in ↪
56                         ↪ images]).astype('float32')
57         imagesLabels = prepareLabels(indexes, imagesLabels)
58
59         return images, imagesLabels

```

Tal y como se esperaba, la gráfica 5.20 muestra como la mejor precisión es obtenida utilizando la *data augmentation correcta* según el caso de uso. También se puede ver como baja la precisión de la *exagerada* según pasan las épocas. Realmente se ve que la *data augmentation* es necesaria para una mejora en la precisión de los resultados.

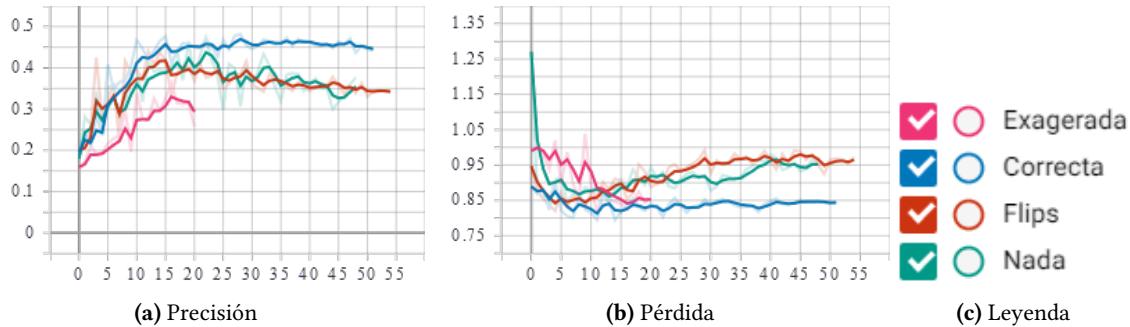


Figura 5.20: Gráficas de las métricas de los resultados con diferentes *data augmentations*

Los falsos positivos no variaran de forma considerable entre diferentes *data augmentation* respecto a su precisión total. Tampoco afectará al rendimiento de ninguna manera, ya que su variación solo se contempla en el procesamiento de las imágenes en el momento de entrenar.

#### 5.4.2. Muestra de resultados

Se puede apreciar que el resultado obtenido en parte de segmentación de humo es diferente al de clasificación. El resultado con mayor precisión ha sido el *backbone* EfficientNetB6 para la U-Net. Sin embargo, no ha sido mejor que el resultado obtenido por InceptionResNetV2 en la parte de clasificación.

En el sentido de rendimiento los modelos de EfficientNet tanto en clasificación como *backbone* para segmentación han sido mucho más eficientes. De hecho han llegado a ser el doble de rápidos. Además, en estos últimos entrenamientos de segmentación ha conseguido buenos resultados. Por tanto, EfficientNetB6 como *backbone* en la U-Net con decodificador *transpose* es el mejor candidato para ser el modelo de detección de humos en Raspberry Pi.

Volver a remarcar que la calidad y, sobre todo en segmentación, el tamaño del *dataset* son los culpables de que la precisión no sea lo suficientemente alta.

## 5.5. ITERACIÓN 6: API REST PARA SEGMENTACIÓN

En esta interacción se ha aprovechado el trabajo de la iteración 3 (Ver Sec. 5.3) se ha realizado un añadido para la predicción de imágenes para segmentar. Se ha añadido la opción para elegir el color de segmentación. El código de predicción se puede ver en el listado 5.3.

**Listado 5.3:** Código fuente de la predicción de una imagen en la API REST para segmentación

```

1 # Recibe la imagen serializada y manda predicción sobre segmentación
2 @app.post("/image-segmentation")
3 async def uploadImageSegmentation(imageRequest: ImageJSON):
4     # Decodifica la imagen previamente serializada
5     image = cvtColor(imdecode(frombuffer(
6         b64decode((imageRequest.image).encode('ascii')), ←
7             ↪ uint8), 1), COLOR_BGR2RGB)
8     (x, y, _) = image.shape
9     # Prepara la imagen con una correcta redimensión
10    imageResized = resizeSegmentation(...)
11    ...
12    # Predice la imagen calculando el tiempo
13    before = datetime.now()
14    prediction = model.predict(imagePredicted, batch_size=1)
15    after = datetime.now()
16    # Crea la máscara predicha por categoría
17    listPrediction = []
18    for i in range(prediction.shape[-1]):
19        imagePrediction = (prediction[0][..., ←
20            ↪ i]*255.0).astype(uint8)
21        _, thresh = threshold(imagePrediction, 127, 255, ←
22            ↪ THRESH_BINARY)
23        thresh = thresh.astype(uint8)
24        ...
25        colortrans = addWeighted(imageResized, 0.5, ←
26            ↪ bitwise_and(bitwise_or(merge([mask, mask, mask]), ←
27            ↪ bitwise_and(
28                merge([thresh, thresh, thresh]), blankImage, None), ←
29            ↪ None), imageResized, None), 0.5, 1)
30        imagePrepared = None
31        ...
32        # Codifica las imágenes resultado junto a la máscara para mandarlo al cliente
33        listPrediction.append(gl.ClassNames[i])
34        listPrediction.append((b64encode(data.tobytes()))...)
35        listPrediction.append((b64encode(dataThresh.tobytes()))...)
36        ...
37        imwrite(join(imageDir, (imageRequest.filename).split(
38            '.')[0] + '.png'), cvtColor(image, COLOR_RGB2BGR))
39        imwrite(join(modelDir, gl.ClassNames[i], ←
39            ↪ (imageRequest.filename).split(
39            '.')[0] + '.png'), thresh)
# Prepara la respuesta al cliente
...
37    imageDict.update({"prediction": listPrediction})
...
39    return imageDict

```

## 5.6. ITERACIÓN 7: DESPLIEGUE EN RASPBERRY PI

El resultado del despliegue proporcionará un computador de escala menor realizando una predicción de las instantáneas de sus cámaras USB conectadas y un apagado automático una vez realizada la predicción. Además, se proporcionará una herramienta para parar ese apagado automático y realizar pruebas para comprobar su correcto funcionamiento.

Se ha instalado una derivación de Raspbian llamado Dietpi. Dietpi es una configuración de Raspbian para que las librerías y aplicaciones instaladas sean mínimas para el funcionamiento de la *shell* y así tener a plena disposición en las tareas que se le encargue.

A continuación, se ha instalado Python 3.7.6 para Raspberry Pi junto al instalador de librerías pip. Después se instala todas las dependencias para usar Tensorflow 2.1, ImgAug y OpenCV. Se ha facilitado un instalador .sh para instalar tanto Python como todas las demás dependencias. Además, este instalador añade en la configuración de arranque de Dietpi la ejecución del *script* que realiza la predicción. El código del instalador se puede ver en el listado 5.4.

**Listado 5.4:** Código fuente del instalador en Raspberry Pi

```

1  #!bin/bash
2  cd /
3  sudo apt-get update -y
4  sudo apt-get upgrade -y
5  sudo mkdir /media
6  sudo mkdir /media/usb
7  sudo chown -R root:root /media/usb
8  sudo mount /dev/sda1 /media/usb
9  sudo cp -r /media/usb/Smoke /root
10 sudo umount /media/usb
11 sudo apt-get install -y python3.7 ...
12 sudo mkdir /root/Smoke/Libraries
13 cd /root/Smoke/Libraries
14 sudo apt-get download -y python3-distutils
15 sudo dpkg-deb -x python3-distutils_3.7.3-1_all.deb /
16 sudo curl https://bootstrap.pypa.io/get-pip.py | python3.7
17 sudo wget https://www.piwheels.org/simple/...
18 sudo wget ←
    ← https://github.com/lhelontra/tensorflow-on-arm/releases/...
19 sudo pip install --upgrade pip
20 sudo pip install ...
21 cd /
22 sudo echo "export ←
    ← LD_PRELOAD=/usr/lib/arm-linux-gnueabihf/libatomic.so.1" ←
    ← >>/root/.bashrc
23 source /root/.bashrc
24 sudo echo "sudo bash /root/Smoke/Run.sh" >>/etc/profile
25 sudo rm -r /root/Smoke/Libraries
26 python3.7 /root/Smoke/Run.py
27 python3.7 /root/Smoke/Test.py
28 source /root/Smoke/Settings.sh
29 sudo echo Loop = $Loop
30 sudo echo GetData = $GetData

```

El modelo definitivo para la detección de humo en Raspberry Pi es U-Net con *backbone* EfficientNetB6 *transpose*. Aunque el modelo de clasificación InceptionResNetV2 sea casi un 15 % más preciso en situaciones de humo el modelo elegido es el doble de eficiente.

La ejecución hará una instantánea por cada cámara conectada y pasará a predecir cada imagen. Si el modelo detecta que existe humo se realizará la acción que se elija en cada caso, ya sea mandar

una señal LoRa, un mensaje a una dirección IP o simplemente accionar cualquier tipo de alarma (Ver listado 5.5).

Además, si la situación legal permite guardar instantáneas se podrá modificar la variable GetData a true en el archivo Settings.sh. Estas imágenes podrán servir en un futuro para aumentar el *dataset*.

**Listado 5.5:** Código fuente del *script* para la detección de humo en Raspberry Pi

```

1 # Carga del modelo
2 model = load_model('/root/Smoke/Models/model.h5', compile=False)
3 listImages = []
4 # Captura la imagen en cada cámara
5 for i in range(4):
6     listImages.append(VideoCapture(i).read()[1])
7     VideoCapture(i).release()
8 for image in listImages:
9     # Comprueba si la imagen contiene humo
10    if sum(sum(threshold((model.predict(array([Sequential(
11        [CenterPadToAspectRatio(1., pad_mode='constant', ←
12            pad_cval=0), Resize((512, 512), ←
13            interpolation=3]).augment_image(cvtColor(image, ←
14            4))/255.]).astype('float'), batch_size=1)[0][..., ←
15            0]*255.).astype('uint8'), 127, 255, ←
16            0)[1].astype('uint8')))) > 0:
17        # Send signal
18        if getData:
19            # Guarda la imagen
20            imwrite('/root/Smoke/Images/smoke/' +
21                    datetime.now().strftime("%Y %m %d %H %M %S")+'.jpg', ←
22                    image)
23    else:
24        if getData:
25            # Guarda la imagen
26            imwrite('/root/Smoke/Images/neutral/' +
27                    datetime.now().strftime("%Y %m %d %H %M %S")+'.jpg', ←
28                    image)

```

El programa de mantenimiento necesitará una computadora extra con un sistema operativo o un programa capaz de leer y escribir los archivos de la microSD de Raspberry Pi. Para parar el apagado automático del *script* se debe cambiar el valor de la variable Loop a false en el archivo Settings.sh en la computadora extra. Primero se deberá introducir un pen drive USB vacío o con espacio considerable en la Raspberry Pi. A continuación, se volverá a introducir la tarjeta microSD y se ejecuta el archivo Test.sh a través de comandos:

*bash Test.sh*

La prueba realizará una predicción de dos imágenes ya guardadas en la Raspberry Pi, una con evidencias humo y otra sin humo. Estas imágenes han sido previamente probadas para que el modelo las prediga de forma correcta. El resultado de la prueba imprimirá por pantalla si ha realizado bien o no la predicción. Además, se guardará en el pen drive los resultados en caso de segmentación y además, en caso de haber tenido la opción de guardar las imágenes, se copiarán en el pen drive eliminado las de la Raspberry Pi. Se puede examinar el código fuente en el listado 5.6.

**Listado 5.6:** Código fuente del script para comprobar el correcto funcionamiento de la Raspberry Pi

```

1 # Carga del modelo
2 model = load_model('/root/Smoke/Models/model.h5', compile=False)
3 # Creación de la imagen con humo redimensionada
4 imageSmokeAug = Sequential([CenterPadToAspectRatio(1., ←
5     ↪ pad_mode='constant', pad_cval=0), Resize(
6         (512, 512), interpolation=3)]).augment_image(cvtColor(
7             imread('/root/Smoke/Test/smoke.jpg'), 4))
8 imageSmoke = array([imageSmokeAug/255.]).astype('float')
9
10 # Comprueba el tiempo que tarda en predecir la imagen con humo (Siempre tardará más en la
11 # primera predicción)
12 before = datetime.now()
13
14 # Predice la imagen con humo
15 predictionSmoke = model.predict(imageSmoke, batch_size=1)
16 after = datetime.now()
17 print('Prediction_time_smoke_image:' + str((after - ←
18     ↪ before).total_seconds()))
19
20 # Creación de la imagen sin humo redimensionada
21 imageNeutralAug = Sequential([CenterPadToAspectRatio(1., ←
22     ↪ pad_mode='constant', pad_cval=0), Resize(
23         (512, 512), interpolation=3)])
24 .augment_image(cvtColor(imread('/root/Smoke/Test/neutral.jpg'), ←
25     ↪ 4))
26 imageNeutral = array([imageNeutralAug/255.]).astype('float')
27
28 # Comprueba el tiempo que tarda en predecir la imagen sin humo
29 before = datetime.now()
30
31 # Predice la imagen sin humo
32 predictionNeutral = model.predict(imageNeutral, batch_size=1)
33 after = datetime.now()
34 print('Prediction_time_neutral_image:' + str((after - ←
35     ↪ before).total_seconds()))
36
37 # Crea las máscaras de cada imagen
38 maskSmoke = threshold(
39     (predictionSmoke[0][..., 0]*255.).astype('uint8'), 127, ←
40     ↪ 255, 0)[1].astype('uint8')
41 maskNeutral = threshold(
42     (predictionNeutral[0][..., 0]*255.).astype('uint8'), 127, ←
43     ↪ 255, 0)[1].astype('uint8')
44
45 # Guarda las imágenes redimensionadas y las máscaras predecidas
46 imwrite('/root/Smoke/Test/smokeMask.jpg', maskSmoke)
47 imwrite('/root/Smoke/Test/smokeAug.jpg', ←
48     ↪ cvtColor(imageSmokeAug, 4))
49 imwrite('/root/Smoke/Test/neutralMask.jpg', maskNeutral)
50 imwrite('/root/Smoke/Test/neutralAug.jpg', ←
51     ↪ cvtColor(imageNeutralAug, 4))
52
53 # Imprime si la predicción ha fallado o ha sido exitosa
54 if sum(sum(maskSmoke)) <= 0 or sum(sum(maskNeutral)) > 0:
55     print('Prediction FAILED!')
56 else:
57     print('Prediction SUCCESSFUL!')
58
59 # Devuelve el tiempo total que le ha llevado la prueba
60 afterAll = datetime.now()
61 print('Time:' + str((afterAll - beforeAll).total_seconds()))

```

## 5.7. ITERACIÓN 8: DESPLIEGUE EN APLICACIÓN WEB

Para la ejecución de la aplicación web en local se necesitará Python 3.7.6 y las librerías Tensorflow, OpenCV, ImgAug y Django. Aunque se ha desarrollado un instalador multiplataforma con los únicos requisitos de tener instalado Python 3.7.\* y, en caso de querer usar aceleración por GPU, CUDA y cuDNN para NVIDIA [10].

La aplicación web almacena modelos previamente entrenados en su base de datos. Estos modelos deberán ser o de clasificación de humo binaria o de segmentación de humo. Además, podrán ser elegidos por el propio usuario del despliegue.

Además las imágenes que pasen por los modelos serán guardadas como recolecta para aumentar el *dataset*.

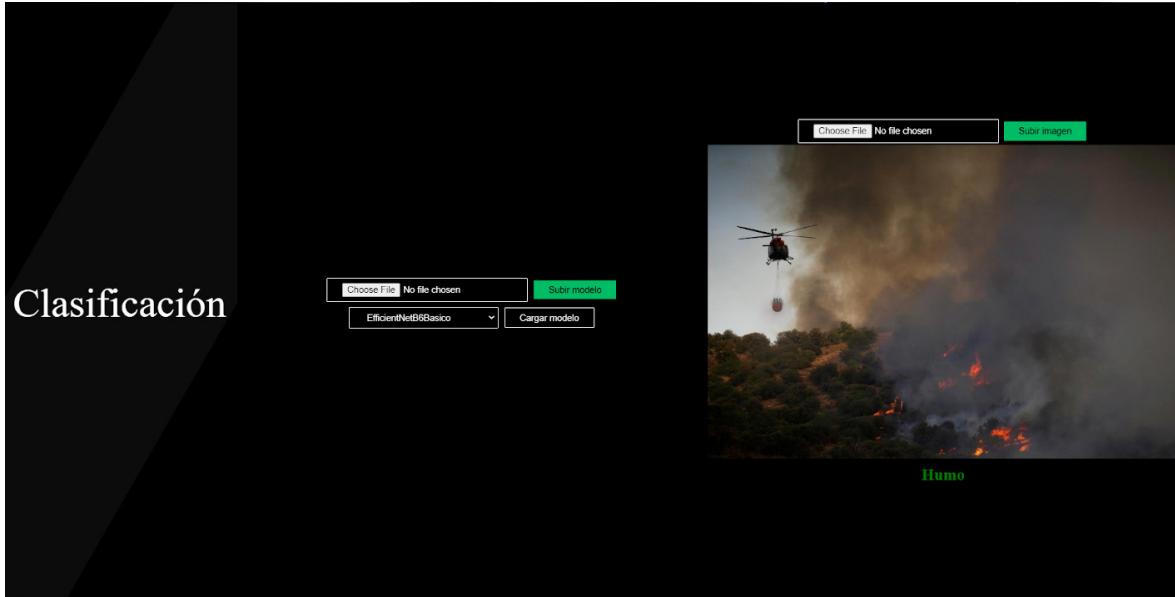
### 5.7.1. Manual de usuario

Una vez se introduzca la URL de la web la página principal mostrará 4 secciones (Ver Fig. 5.21):

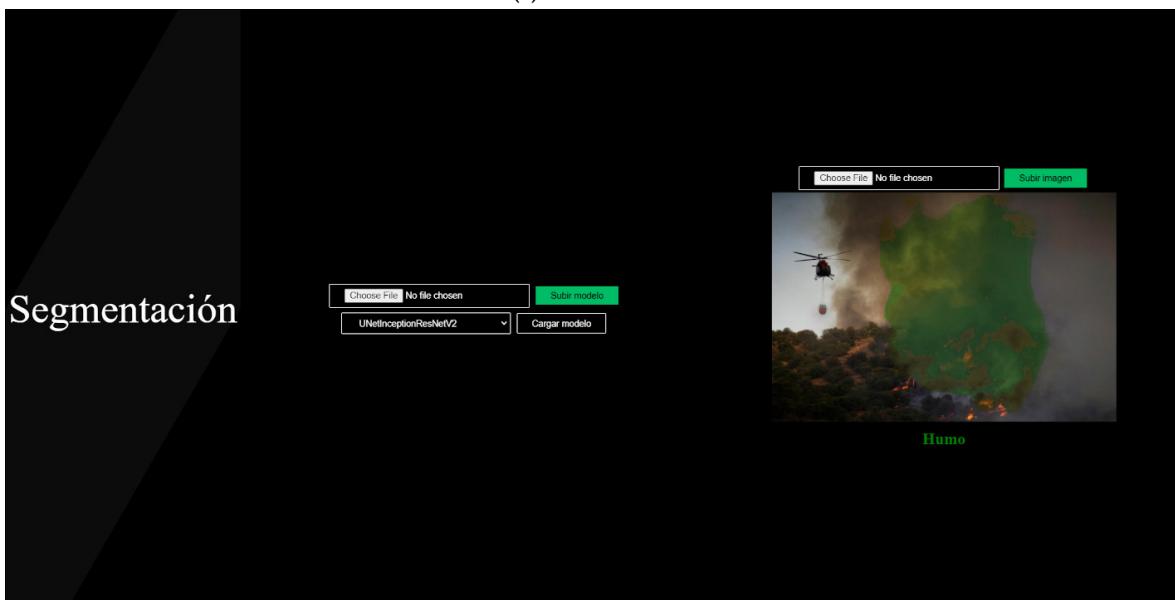


Figura 5.21: Imagen de la página principal de la aplicación web

- **Predicción**(Ver Fig. 5.22): esta sección es la aplicación de la página web. Contiene la lógica de predicción tanto de segmentación como de clasificación. Tiene dos subsecciones: Clasificación y Segmentación. Cada sección nos permite cargar modelos previamente entrenados para la detección de humo. Los modelos predeterminados son modelos elegidos por el servidor, pero cada usuario podrá subir sus propios modelos. Una vez cargado un modelo se deberá subir una imagen que posteriormente será enviada al modelo y su resultado se visualizará en la web, ya sea clasificación con un etiquetado debajo de la imagen o segmentación con la creación de una imagen con el humo señalizado en verde.  
Cabe destacar que esta sección queda bloqueada una vez un usuario la está usando. A partir de la IP del cliente si está libre solo aceptará acciones de esta IP hasta que lleve 5 o más minutos sin realizar ninguna acción (Subir modelo, subir imagen, cargar modelo, cambiar de método...). Una vez el usuario termine la sesión los modelos e imágenes que haya subido serán borrados. El listado 5.7 muestra como se estructura la Web en código y la lógica para controlar la IP del cliente.
- **Manual:** se muestra una guía de cómo utilizar la página web y sus características.
- **Instalación:** la sección proporciona un tutorial con la instalación de las dependencias necesarias para desplegar la página web en cualquier computador.
- **Documentación:** muestra sin previas descargas la documentación de este proyecto.



(a) Clasificación



(b) Segmentación

**Figura 5.22:** Imagen de la interfaz de predicción de la aplicación web

**Listado 5.7:** Código fuente de las funciones que representan cada dirección de la web

```
1 CurrentIP = ''
2 LastUsed = datetime.now()
3
4 # Página principal
5 def index(request):
6     return render(request, 'index.html')
7
8
9 # Página de elección de predicción
10 def prediction(request):
11     return render(request, 'prediccion.html')
12
13
14 # Página para la clasificación de imágenes
15 def classification(request):
16     global CurrentIP
17     ...
18     ip = request.META['REMOTE_ADDR']
19     if CurrentIP == ip:
20         context = {}
21         LastUsed = datetime.now()
22         if request.method == 'POST':
23             ...
24             return render(request, 'clasificacion.html', context)
25         else:
26             if (datetime.now() - LastUsed).total_seconds() > 300:
27                 CurrentIP = ip
28                 ...
29                 return render(request, 'clasificacion.html', context)
30             else:
31                 return render(request, 'prediccionOcupada.html')
32
33
34 # Página para la segmentación de imágenes
35 def segmentation(request):
36     global CurrentIP
37     ...
38     ip = request.META['REMOTE_ADDR']
39     if CurrentIP == ip:
40         context = {}
41         LastUsed = datetime.now()
42         ...
43         return render(request, 'segmentacion.html', context)
44     else:
45         if (datetime.now() - LastUsed).total_seconds() > 300:
46             CurrentIP = ip
47             ...
48             return render(request, 'segmentacion.html', context)
49     else:
50         return render(request, 'prediccionOcupada.html')
51
52
53 # Página donde muestra la documentación
54 def documentation(request):
55     return render(request, 'documentacion.html')
56
57 ...
```



---

## CAPÍTULO 6

# CONCLUSIONES

---

A continuación, se expondrá los objetivos alcanzados, el trabajo futuro y la conclusión de los resultados obtenidos. Además se valorará que aportación al autor a supuesto este proyecto.

### 6.1. OBJETIVOS ALCANZADOS

- **Creación del conjunto de datos.**

A partir de un nuevo caso de uso sin imágenes previas se ha podido llegar a construir un *dataset* de 7940 imágenes de las cuales 5056 han sido para clasificación y 2884 para segmentación (Ver tabla 5.1).

La variedad y el tipo de imágenes encontradas no han sido los esperados, pero se ha conseguido mantener una relación entrenamiento-validación que permita mantener un cierto nivel de coherencia aceptable en los resultados.

- **Análisis de arquitecturas de redes neuronales.**

Se ha conseguido analizar todo tipo de configuración de redes neuronales tanto para clasificación como para segmentación. Al final de la versión 1 se ha podido concluir qué modelo de red neuronal es la más eficaz para nuestro caso de uso. La versión 2, además de elegir que *backbone* era el mejor para el caso de uso, ha podido mostrar que configuración de decodificación y *data augmentation* es la más precisa y eficiente.

Aunque realmente el resultado obtenido no ha sido una precisión lo suficientemente alta como para confiar en la predicción de una instantánea a nivel de una situación real, con la ejecución secuencial del algoritmo se llevará a cabo una mejor predicción. Además, se ha configurado para que con la mejora del conjunto de datos se pueda conseguir un resultado óptimo.

- **Creación de módulo de detección de incendios.**

Gracias a que se ha tenido a disposición una Raspberry Pi se ha podido comprobar su correcto despliegue. Con el método de mantenimiento implementado se ha superado el objetivo con creces.

- **Creación de aplicación web.**

Se ha conseguido desplegar una aplicación web con una interfaz atractiva e intuitiva. También se desarrolló para su uso comercial.

- **Creación de API REST para predecir humo.**

Una API REST ha sido necesaria para la predicción masiva de imágenes. Su creación ha sido realizada durante el análisis de los modelos de redes neuronales. Aunque no había un objetivo claro sobre cual iba a ser la utilidad teniendo disponibles predicciones masivas sin API REST, su desarrollo ha sido pensando en posibles microservicios que se podría implementar en un futuro. El uso de librerías punteras ha sido planeado para la investigación de éstas.

## 6.2. TRABAJOS FUTUROS

El principal problema que ha habido en la solución del proyecto ha sido la baja precisión de detección provocada por la insuficiente robustez del conjunto de datos. Por tanto, el principal objetivo a futuro es incorporar imágenes orientadas al caso de uso, incendios forestales. Este problema se intentará solucionar con el despliegue descrito en resultados (Ver 5.6) activando la opción de recoger datos. Con el tiempo se recogerán suficientes imágenes para fortalecer el *dataset*.

También existe la incertidumbre de que señal de transmisión se utilizará para el envío de alarma de incendios. La opción más viable tanto económica como eficiente sería la transmisión por señal LoRa.

Junto al envío de señal faltaría determinar que proceso de autogestión de alimentación tendrá la Raspberry Pi. Gracias a su popularidad dispone de diferentes *hats* tanto para enviar señales LoRa como para gestionar un temporizador para encenderse cada cierto tiempo. Además de añadir una batería con una alimentación solar que permita su permanente alimentación.

Como último punto futuro una de las ideas del desarrollo del proyecto ha sido de ser universal, es decir que sirva para un cambio de objetivo en la clasificación y segmentación de imágenes. Junto a un nuevo conjunto de datos y un pequeño análisis de que modelo de redes neuronales se podría realizar nuevos resultados sobre otro objetivo, por ejemplo, detectar cierta especie animal o incluso saber diferenciar entre otras.

## 6.3. CONCLUSIÓN

Los resultados han sido aceptables aún con la limitación del conjunto de datos. Realmente no sería una opción para desplegar en situaciones de alto riesgo, pero quizás en zonas cercanas para la continua recogida de información, realización de pruebas y minimizar el coste de transporte proporcionaría una zona segura de propagación de incendios.

Aunque el resultado del proyecto ha llevado a una demo demostrativa y no a un cliente los objetivos parciales y el objetivo principal han sido alcanzados de manera satisfactoria.

## 6.4. COMPETENCIAS SATISFECHAS

Las competencias de la intensificación de Computación satisfechas en este proyecto son:

- *Capacidad para tener un conocimiento profundo de los principios fundamentales y modelos de la computación y saberlos aplicar para interpretar, seleccionar, valorar, modelar, y crear nuevos conceptos, teorías, usos y desarrollos tecnológicos relacionados con la informática.* La capacidad se satisface al comprobar que la detección de humos habitualmente es ocupada por otros sectores, ya sean química, medioambiental o telemática. En este caso, es abordada con éxito por modelos informáticos.
- *Capacidad para conocer los fundamentos teóricos de los lenguajes de programación y las técnicas de procesamiento léxico, sintáctico y semántico asociadas, y saber aplicarlas para la creación, diseño y procesamiento de lenguajes.* Este proyecto refleja patrones de construcción de código con un diseño complejo y práctico. Además, se utilizan librerías y herramientas donde se visualiza el procesamiento de lenguajes.
- *Capacidad para evaluar la complejidad computacional de un problema, conocer estrategias algorítmicas que puedan conducir a su resolución y recomendar, desarrollar e implementar aquella que garantice el mejor rendimiento de acuerdo con los requisitos establecidos.* La base del proyecto es el análisis de resultados obtenidos en base a la configuración previamente elegida. Esta configuración será elegida con la tendencia de mejorar los resultados anteriores.

- *Capacidad para conocer los fundamentos, paradigmas y técnicas propias de los sistemas inteligentes y analizar, diseñar y construir sistemas, servicios y aplicaciones informáticas que utilicen dichas técnicas en cualquier ámbito de aplicación.* El despliegue de resultados como la API REST, la aplicación web o la Raspberry Pi refleja la capacidad de analizar, diseñar y construir sistemas informáticos que utilizan técnicas de *deep learning*.
- *Capacidad para adquirir, obtener, formalizar y representar el conocimiento humano en una forma computable para la resolución de problemas mediante un sistema informático en cualquier ámbito de aplicación, particularmente los relacionados con aspectos de computación, percepción y actuación en ambientes entornos inteligentes.* El proyecto se podría resumir en la captura de datos provenientes de sentidos como es la vista transformados en vectores de colores 2D, procesar esta información, tal que la lógica conlleve a un resultado real, ya sea clasificación o segmentación, simulado en un computador.
- *Capacidad para desarrollar y evaluar sistemas interactivos y de presentación de información compleja y su aplicación a la resolución de problemas de diseño de interacción persona computadora.* Gracias a la página web un usuario común puede interactuar con técnicas de *deep learning* con tan solo botones y desplegables.
- *Capacidad para conocer y desarrollar técnicas de aprendizaje computacional y diseñar e implementar aplicaciones y sistemas que las utilicen, incluyendo las dedicadas a extracción automática de información y conocimiento a partir de grandes volúmenes de datos.* El uso de redes neuronales complejas junto al *deep learning* supervisado dejan reflejada la competencia satisfecha.



# BIBLIOGRAFÍA

---

- [1] O. Dürr y col., «Deep Learning on a Raspberry Pi for Real Time Face Recognition.», en *Eurographics (Posters)*, 2015, págs. 11-12. dirección: [https://www.researchgate.net/publication/279537625\\_Deep\\_Learning\\_on\\_a\\_Raspberry\\_Pi\\_for\\_Real\\_Time\\_Face\\_Recognition](https://www.researchgate.net/publication/279537625_Deep_Learning_on_a_Raspberry_Pi_for_Real_Time_Face_Recognition).
- [2] I. Goodfellow, Y. Bengio y A. Courville, *Deep learning*. MIT press, 2016.
- [3] K. He y col., «Deep residual learning for image recognition», en *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, págs. 770-778. dirección: <https://arxiv.org/abs/1512.03385>.
- [4] *ImageNet Challenge*. dirección: <http://image-net.org/about-overview>.
- [5] N. Kalchbrenner, E. Grefenstette y P. Blunsom, «A convolutional neural network for modelling sentences», *arXiv preprint arXiv:1404.2188*, 2014. dirección: <https://arxiv.org/abs/1404.2188>.
- [6] T. L. Koch y col., «Accurate segmentation of dental panoramic radiographs with u-NETS», en *2019 IEEE 16th International Symposium on Biomedical Imaging (ISBI 2019)*, IEEE, 2019, págs. 15-19. dirección: [https://www.researchgate.net/publication/333610363\\_Accurate\\_Segmentation\\_of\\_Dental\\_Panoramic\\_Radiographs\\_with\\_U-NETS](https://www.researchgate.net/publication/333610363_Accurate_Segmentation_of_Dental_Panoramic_Radiographs_with_U-NETS).
- [7] Y. Kodratoff, *Introduction to machine learning*. Elsevier, 1988.
- [8] Y. LeCun, K. Kavukcuoglu y C. Farabet, «Convolutional networks and applications in vision», en *Proceedings of 2010 IEEE international symposium on circuits and systems*, IEEE, 2010, págs. 253-256. dirección: [https://www.researchgate.net/publication/221376179\\_Convolutional\\_Networks\\_and\\_Applications\\_in\\_Vision](https://www.researchgate.net/publication/221376179_Convolutional_Networks_and_Applications_in_Vision).
- [9] S. A. Magid, F. Petrini y B. Dezfouli, «Image classification on IoT edge devices: profiling and modeling», *Cluster Computing*, págs. 1-19, 2019. dirección: <https://arxiv.org/abs/1902.11119>.
- [10] M. G. Monclú, *Repositorio GitHub*. dirección: <https://github.com/MauroGarciaMonclu/DeepLearningSmokeDetection.git>.
- [11] A. Namozov e Y. Im Cho, «An efficient deep learning algorithm for fire and smoke detection with limited data», *Advances in Electrical and Computer Engineering*, vol. 18, n.º 4, págs. 121-128, 2018. dirección: [https://www.researchgate.net/publication/329411277\\_An\\_Efficient\\_Deep\\_Learning\\_Algorithm\\_for\\_Fire\\_and\\_Smoke\\_Detection\\_with\\_Limited\\_Data](https://www.researchgate.net/publication/329411277_An_Efficient_Deep_Learning_Algorithm_for_Fire_and_Smoke_Detection_with_Limited_Data).
- [12] J. Á. Olivas Varela, «Contribución al estudio experimental de la predicción basada en categorías deformables borrosas», 2000. dirección: <https://ruidera.uclm.es/xmlui/bitstream/handle/10578/18399/TESIS%20Olivas%20Varela.pdf>.
- [13] L. Perez y J. Wang, «The effectiveness of data augmentation in image classification using deep learning», *arXiv preprint arXiv:1712.04621*, 2017. dirección: <https://arxiv.org/abs/1712.04621>.
- [14] M. A. Qurishee, «Low-cost deep learning UAV and Raspberry Pi solution to real time pavement condition assessment», 2019. dirección: <https://scholar.utc.edu/theses/601/>.
- [15] O. Ronneberger, P. Fischer y T. Brox, «U-net: Convolutional networks for biomedical image segmentation», en *International Conference on Medical image computing and computer-assisted intervention*, Springer, 2015, págs. 234-241. dirección: <https://arxiv.org/abs/1505.04597>.
- [16] F. Rosenblatt, «The perceptron: a probabilistic model for information storage and organization in the brain.», *Psychological review*, vol. 65, n.º 6, pág. 386, 1958.

- [17] H. Rui y col., «Fault point detection of IOT using multi-spectral image fusion based on deep learning», *Journal of Visual Communication and Image Representation*, vol. 64, pág. 102 600, 2019. dirección: <https://doi.org/10.1016/j.jvcir.2019.102600>.
- [18] A. Stellman y J. Greene, *Learning agile: Understanding scrum, XP, lean, and kanban*. :’Reilly Media, Inc.”, 2014.
- [19] C. Szegedy y col., «Inception-v4, inception-resnet and the impact of residual connections on learning», en *Thirty-first AAAI conference on artificial intelligence*, 2017. dirección: <https://arxiv.org/abs/1602.07261>.
- [20] M. Tan y Q. V. Le, «Efficientnet: Rethinking model scaling for convolutional neural networks», *arXiv preprint arXiv:1905.11946*, 2019. dirección: <https://arxiv.org/abs/1905.11946>.
- [21] F. Vallejo Banegas, «Estudio y desarrollo de técnicas de generación de descriptores de imagen 2D.», 2019. dirección: [https://ruidera.uclm.es/xmlui/bitstream/handle/10578/23135/TFG\\_VallejoBanegasFernando.pdf](https://ruidera.uclm.es/xmlui/bitstream/handle/10578/23135/TFG_VallejoBanegasFernando.pdf).
- [22] S. M. Weiss y C. A. Kulikowski, *Computer systems that learn: classification and prediction methods from statistics, neural nets, machine learning, and expert systems*. Morgan Kaufmann Publishers Inc., 1991.
- [23] S. Wu y X. Li, «Iou-balanced loss functions for single-stage object detection», *arXiv preprint arXiv:1908.05641*, 2019. dirección: <https://arxiv.org/abs/1908.05641>.
- [24] C. Ye y col., «Network deconvolution», *arXiv preprint arXiv:1905.11926*, 2019. dirección: <https://arxiv.org/abs/1905.11926>.

## **ANEXOS**



---

## ANEXO A

---

# HATS PARA RASPBERRY PI

---

La parte de resultados ha abordado el despliegue de una Raspberry Pi en el sentido software, pero en este anexo se abordará en un sentido hardware.

### A.1. TRASMISIÓN

Lo primero que se necesitará abordar es la comunicación. El caso de uso busca estar alejado de convivencia humana por tanto se busca señales inalámbricas de larga distancia. Por tanto, la tecnología LoRa será elegida para este despliegue. Existe un *hat* para Raspberry Pi (Ver Fig. A.1<sup>1</sup>) que añade un transmisor a una frecuencia de 868 Mhz (Permitida en Europa).



Figura A.1: Imagen del *hat* transmisor de señal LoRa

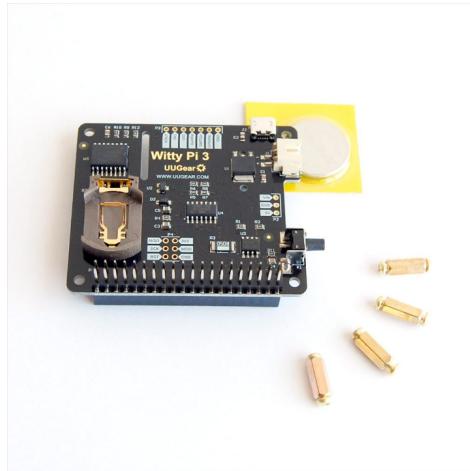
### A.2. ALIMENTACIÓN

A continuación, se debería gestionar un temporizador para que la Raspberry Pi se encienda cada cierto tiempo. Además, se deberá gestionar la alimentación por baterías y su recarga por paneles solares. Para todo ello existe un *hat* llamado Witty Pi (Ver Fig. A.2<sup>2</sup>) que se puede instalar junto a otros *hats*.

Aunque no se ha hecho un estudio exhaustivo de la autonomía de la Raspberry Pi, si se hace una

<sup>1</sup>[urlhttps://www.robotshop.com/media/catalog/product/cache/1350x/9df78eab33525d08d6e5fb8d27136e95/r/a/raspberry-pi-lora-gps-hat-3\\_1.jpg](https://www.robotshop.com/media/catalog/product/cache/1350x/9df78eab33525d08d6e5fb8d27136e95/r/a/raspberry-pi-lora-gps-hat-3_1.jpg)

<sup>2</sup>[urlhttps://www.robotshop.com/media/catalog/product/cache/1350x/9df78eab33525d08d6e5fb8d27136e95/w/i/witty-pi-3-rtc-power-management-raspberry-pi-boards-2.jpg](https://www.robotshop.com/media/catalog/product/cache/1350x/9df78eab33525d08d6e5fb8d27136e95/w/i/witty-pi-3-rtc-power-management-raspberry-pi-boards-2.jpg)



**Figura A.2:** Imagen del *hat* para gestionar la alimentación

ejecución cada 10 minutos, una batería de 12000 mAh (Ver Fig. A.3<sup>3</sup>) y un panel solar (Ver Fig. A.4<sup>4</sup>) de 12 V será más que suficiente para garantizar la autonomía del producto.



**Figura A.3:** Imagen de la batería de 12000 mAh



**Figura A.4:** Imagen del panel solar de 12 V

<sup>3</sup>url[https://cdn.shopify.com/s/files/1/2187/3161/products/PiJuice-12000mAh-Angle-Left\\_1024x.jpg](https://cdn.shopify.com/s/files/1/2187/3161/products/PiJuice-12000mAh-Angle-Left_1024x.jpg)

<sup>4</sup>url<https://www.robotshop.com/media/catalog/product/cache/1350x/9df78eab33525d08d6e5fb8d27136e95/p/i/pijuice-solar-panel-12-watt.jpg>

### A.3. CARCASA

Todo esto junto a las cámaras y la propia Raspberry Pi será protegido contra fenómenos atmosféricos.

### A.4. COSTE

El coste de la Raspberry Pi, una cámara y todos los productos expuestos en este anexo se refleja en la tabla A.1<sup>5</sup>.

**Tabla A.1:** Coste

Producto	Precio/unidad	Cantidad	Precio
Raspberry Pi 3 Model B+	43,72€	1	43,72€
Cámara	29,10€	1	29,10€
Witty Pi 3	26,37€	1	26,37€
LoRa/GPS HAT	38,01€	1	38,01€
Batería 12000 mAh	32,88€	1	32,88€
Panel solar 12 Vatios	87,73€	1	87,73€
<b>Total</b>			257,81€

---

<sup>5</sup>Precios extraídos de la tienda: <https://www.robotshop.com/>

