UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dpto. de Ingeniería de Sistemas y Automática

Desarrollo de una red neuronal convolucional (CNN) para la distinción de variedades de pólenes en muestras de miel

Trabajo Fin de Máster

Máster Universitario en Automática e Informática Industrial

AUTOR/A: Martín Osuna, Juan José

Tutor/a: Valiente González, José Miguel

Cotutor/a: López García, Fernando

CURSO ACADÉMICO: 2023/2024

Instituto Universitario de Automática e Informática Industrial
Universitat Politècnica de València

Desarrollo de una red neuronal convolucional (CNN) para la distinción de variedades de pólenes en muestras de miel

TRABAJO DE FINAL DE MÁSTER

Máster Universitario en Automática e Informática Industrial

Autor: Juan José Martín Osuna

Tutores: José Miguel Valiente Gonzalez y Fernado López García

Curso 2023-2024

Agradecimientos

No hace tanto que escribía unas palabras muy similares en un texto muy similar a este, lleno de ilusión y ganas, pero también de esfuerzo y sacrificio. Pero si algo tengo claro, son las personas que me han llevado a poder enfrentarme a las dificultades y por eso quiero dedicarles estas palabras, porque cada día hay miles de momentos para tirar la toalla y ellos siempre han estado ahí para ayudarme a recogerla.

Gracias a toda mi familia, por estar siempre ahí, pero en particular, a mis abuelos que siempre me han apoyado, pese a que aún no tengan muy claro a qué me dedico y a mis padres por escucharme y aconsejarme cada día y darme las facilidades sin las duda no estaría aquí ahora mismo.

Gracias a mi pareja, la que a día de hoy es, y siempre ha sido mi más fiel compañera de investigación, interesándose y dándolo todo para que, por muy duro que se presente, nunca pierda mi norte. Sin ti, quién sabe cómo acabaría, porque sin ti ninguna de estas palabras serían las que son.

Gracias a mis compañeros de clase, que pese a los años que llevamos juntos, aún siguen a mi lado y que, en retrospectiva, para lo malos que fuimos, lo lejos que hemos llegado.

Gracias a mi equipo, a mis hermanos mayores, que me han enseñado muchas más cosas que a jugar a las cartas y los que siempre me demuestran que a veces se gana y otras se aprende, aunque yo aprenda más veces de la cuenta.

Gracias a mis compañeros de laboratorio, no solo por acogerme y permitirme tener un ambiente laboral increíble, sino por todo lo que me han enseñado, en un mundo que hasta hace bien poco era completamente ajeno a mí.

Por ultimo, gracias a mis profesor y a todos los profesionales de los proyectos Polenet y Agromel, primero y antes que nada por darme la oportunidad profesional que me lleva hoy a poder escribir este proyecto, pero, sobre todo, por despertar en mi ese espíritu investigador que ni yo mismo sabia que tenía. Gracias a su dedicación y guía han dado forma a estas páginas

Gracias por todo, hoy y siempre...

Resum

El present projecte té com a objectiu el desenvolupament d'una nova arquitectura de Xarxa neuronal convolucional (CNN) per al projecte "Polenet", així com el seu posterior estudi de comportament amb diferents conjunts de dades i la seua comparació front altres arquitectures CNN estàndard.

La funció de la xarxa serà distingir entre diferents varietats de pòl·lens en la mel per a la determinació de la seua varietat. Per a això, el punt de partida serà una primera versió de la Xarxa "Polenet". Una xarxa amb una estructura lineal similar a les VGG.

L'objectiu principal d'aquesta nova arquitectura serà millorar la taxa d'encerts de la xarxa, així com, en cas de ser possible, disminuir la seua grandària per a la seua futura implementació. Per a això es provarà la incorporació i substitució de diferents tipus de capes, amb l'objectiu d'alinear cada vegada més els resultats de la xarxa amb la resta d'arquitectures estàndard.

A més de l'optimització de l'arquitectura, es plantejaran estratègies addicionals per a millorar el rendiment del model s'exploraran tècniques d'assemblat de múltiples xarxes i tècniques d'ajust de hiperparàmetres de manera iterativa.

Paraules clau: Xarxes neuronals convolucionals, CNN, Keras, Tensorflow, Python, Octuna, Pol·len, Visió per computador, Arquitectura de xarxes neuronals, Hiperparàmetres, Conjunts de mostres de pòl·lens, Agrupació de xarxes neuronals

Resumen

El presente proyecto tiene como objetivo el desarrollo de una nueva arquitectura de Red neuronal convolucional (CNN) para el proyecto “Polenet”, así como su posterior estudio de comportamiento con distintos conjuntos de datos y su comparación frente otras arquitecturas CNN estándar.

La función de la red será distinguir entre distintas variedades de pólenes en la miel para la determinación de su variedad. Para ello, el punto de partida será una primera versión de la Red “Polenet”. Una red con una estructura lineal similar a las VGG.

El objetivo principal de esta nueva arquitectura será mejorar la tasa de aciertos de la red, así como, en caso de ser posible disminuir su tamaño para su futura implementación. Para ello se probarán la incorporación y sustitución de distintos tipos de capas, con el objetivo de alinear cada vez más los resultados de la red con el resto de arquitecturas estándar.

Además de la optimización de la arquitectura, se plantearán estrategias adicionales para mejorar el rendimiento del modelo se explorarán técnicas de ensamblado de múltiples redes y técnicas de ajuste de hiperparámetros de manera iterativa.

Palabras clave: Redes neuronales convolucionales, CNN, Keras, Tensorflow, Python, Optuna, Polen, Visión por computador, Arquitectura de redes neuronales, Hiperparámetros, Conjuntos de muestras de pólenes, Agrupación de redes neuronales

Abstract

This project aims to develop a new Convolutional Neural Network (CNN) architecture for the "Polenet" project, its subsequent behavior study with different datasets, and its comparison against other standard CNN architectures.

The function of the network will be to distinguish between different varieties of pollen in honey to determine its variety. To do this, the starting point will be the first version of the "Polenet" Network. A network with a linear structure similar to the VGG.

The main objective of this new architecture will be to improve the network's success rate and, if possible, reduce its size for future implementation. To do this, the incorporation and replacement of different types of layers will be tested, to increasingly align the results of the network with the rest of the standard architectures.

In addition to the optimization of the architecture, additional strategies will be proposed to improve the performance of the model. Multiple network assembly techniques and hyperparameter tuning techniques will be explored iteratively.

Key words: Convolutional Neural Netwoks, CNN, Keras, Tensorflow, Python, Octuna, Pollen, Computer Vision, Neural Netwok Architectures, Hyperparameters, Pollen Dataset, Ensamble

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

1.1 Contexto

La miel es un elemento natural producido por la actividad de las abejas. Para obtenerla, es necesario un proceso de deshidratación y transformación del néctar de las flores, que tiene como resultado la obtención de un producto alimenticio de color dorado y textura densa, considerado esencial en la gastronomía global.

Su uso, se atribuye principalmente a la acción edulcorante, sin embargo, también puede aplicarse como alimento funcional, dadas sus grandes propiedades terapéuticas que pueden beneficiar al consumidor. Este hecho, se sustenta bajo estudios como el de *"**Medicinal uses and health benefits of Honey: An Overview**"* de *Kumar, K. S., Bhowmik, D., Biswajit, C., y Chandira, M. R.*, publicado en 2010 [1].

A nivel global, España está situado como uno de los 5 mayores exportadores de miel, con un volumen de importación de 117,53 millones de dólares, convirtiéndose así en el segundo mayor exportador de la Unión Europea (UE), por detrás de Alemania. Todo ello, queda reflejado en la Figura 1.1.

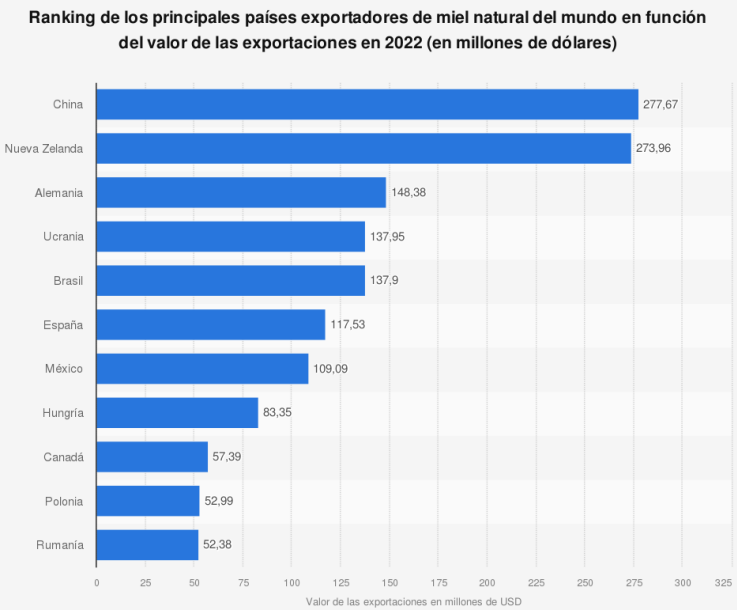


Figura 1.1: *Gráfica comparativa del Ranking de los principales países exportadores de miel natural del mundo en función del valor de las exportaciones en 2022(en millones de dólares)[2]*

Es por esto que, la apicultura es un sector muy importante en el sector primario del país en el país, con 33.833 explotaciones apícolas en todo el territorio según indica el censo de 2022 [4]. Esto sitúa a Andalucía como la principal provincia productora dentro del territorio, con el 15,8 % de las explotaciones. Sumando, entre todos esos centros, un total de 3.033.589 de colmenas, que permiten, no solo la producción de la miel, sino también llegar a obtener 1.903,8 toneladas de cera. Este material, es también resultante de la actividad de las colmenas y cuenta con diversas utilidades, entre las que destaca su importancia en la industria cosmética.

A la vista estos datos, es notoria la importancia de la apicultura dentro de la agroindustria Española. Esto prueba, por tanto, que su preservación y mejora progresiva es una prioridad vital para mantener la posición actual de este país a nivel global como referente en este sector.

1.2 Problemática

Una vez analizada la importancia de la apicultura en España, así como sus ventajas nutricionales, funcionales y económicas, es importante profundizar en la idea de que no todas las mieles son iguales. Estas, presentan una gran diversidad, caracterizada por propiedades como: la α-glucosidasa, la conductividad, el color, o el espectro polínico, siendo esta última la característica que da origen a este proyecto.

Gracias a estas propiedades, puede establecerse una clasificación que abarca dos grandes tipos de mieles:

- **Mieles monoflorales**: aquellas que destacan sobre todo por contar con una variedad de polen que destaca sobre el resto.
- **Mieles multiflorales o de mil flores**: aquellas en las que ningún polen destaca del resto.

Desde el punto de vista legislativo, la miel se clasifica en función de los parámetros anteriormente nombrados y siguiendo las especificaciones recogidas en el BOE 301 del 17/12/2002[3]. Por ejemplo, la variedad de miel monofloral de romero («Rosmarinus officinalis») se define como aquella miel que cuente con las siguientes propiedades físicoquímicas y melitopalinológica:

- "α-glucosidasa >40 US."
- "Conductividad <2,5 S/cm^{-1}."
- "Color <35 Pfund."
- "El polen de «Rosmarinus officinalis» superará el 15 por 100 del espectro polínico o bien superará el 10 por 100 siempre y cuando venga acompañado de un 5 por 100 de formas polínicas de la familia «Lamiaceae»".

Sin embargo, para que una miel sea determinada como perteneciente a la variedad milfloral, sólo es necesario cumplir con lo descrito en el apartado del BOE 301 del 17/12/2002[3]:

- "El polen de los representantes de la familia «Lamiaceae» superará el 5 por 100 del espectro polínico".

Como se puede apreciar en estos extractos del BOE, existe una gran diferencia por lo que respecta a las exigencias que ha de cumplir una determinada miel para ser considerada monofloral, frente a las que debe obedecer para ser considerada multifloral. Esto afecta directamente a su valor, dando como resultado una diferenciación de precios entre ambas variedades, siendo la monofloral la más valorada económicamente. Esta diferencia se advierte en la figura 1.2, donde se comparan los precios de las distintas mieles en la campaña de 2022-2023.

	COTIZACIONES MENSUALES DE MIEL Y POLEN DE PRODUCCION NACIONAL (euros/kg)											
A)- MIEL	Campaña 2022/2023											
MIEL A GRANEL	ABRIL	MAYO	JUNIO	JULIO	AGOSTO	SEPTIEMBRE	OCTUBRE	NOVIEMBRE	DICIEMBRE	ENERO	FEBRERO	MARZO
MULTIFLORAL	3,54	3,54	3,71	3,78	3,81	3,91	3,93	3,72	3,74	3,50	3,45	3,41
MIEL DE MIELADA	4,33	4,33	4,33	4,33	4,33	4,42	4,42	4,17	4,44	4,22	4,14	4,18
MONOFLORAL												
azahar	4,13	4,30	4,44	4,42	4,47	4,79	4,73	4,56	4,42	4,42	4,55	4,59
brezo	5,00	4,74	4,78	4,75	4,75	4,75	4,75	5,03	5,03	5,03	5,12	4,83
eucalipto	3,95	4,11	4,11	4,11	4,03	4,12	4,12	4,47	4,05	4,18	4,11	4,01
girasol	3,44	3,21	3,31	3,31	3,66	3,66	3,66	3,64	3,66	3,24	3,39	3,10
naranjo	4,50	4,30	4,30	4,60	4,60	4,60	4,60	4,35	4,35	4,35	4,35	4,35
romero	4,19	4,66	4,69	4,82	4,98	4,94	4,88	4,52	4,57	5,06	5,33	5,16
tomillo	4,22	4,22	4,42	4,50	4,50	4,50	4,49	4,53	4,37	4,83	5,19	4,87
espliego	4,17	4,72	4,72	4,72	4,72	4,72	4,72	4,72	4,72	4,94	4,56	4,56
castaño	4,04	4,03	4,03	4,14	4,14	4,14	4,13	4,13	4,15	4,15	4,24	4,22
retama	4,06	3,84	3,53	3,53	3,80	3,82	3,80	3,58	3,58	3,88	3,88	3,84
MIEL ENVASADA												
MIEL MULTIFLORAL	5,73	5,61	5,92	5,97	5,80	6,22	6,40	5,82	6,36	6,13	6,11	6,01
MIEL DE MIELADA	6,93	6,93	6,93	6,93	6,93	6,99	6,99	6,99	6,99	6,99	6,99	6,99
MIEL MONOFLORAL												
azahar	7,32	7,32	7,60	7,54	7,50	8,19	8,11	7,47	7,47	7,38	7,79	7,10
espliego	5,38	5,38	5,84	6,61	6,61	6,61	6,61	6,61	6,61	6,61	5,91	6,61
eucalipto	7,00	7,00	7,22	7,25	7,24	7,63	7,63	7,25	7,31	7,24	7,09	7,10
romero	7,08	7,20	7,15	7,30	7,26	8,00	7,84	7,39	7,56	7,58	7,21	7,09
brezo	7,43	7,90	7,90	7,90	7,52	7,90	7,90	7,90	7,90	7,90	7,53	7,93
B)- POLEN												
POLEN A GRANEL	8,24	8,03	7,68	7,64	7,68	7,68	7,96	6,88	6,75	7,24	7,98	6,31
POLEN ENVASADO	11,70	11,38	11,43	10,79	10,68	11,08	11,62	10,79	10,79	11,24	12,21	11,16

Figura 1.2: *Imagen comparativa del precio (€/kg) de la miel según su variedad en España en la campaña 2022/2023[5]*

A la vista de estos los datos expuestos en la figura anterior, se puede deducir la importancia de conocer y comprobar las características que determinan la variedad de una miel. Así pues, el presente proyecto se centrará en la parte relativa al análisis del espectro polínico.

El método más extendido para realizar este análisis consiste en la metodología de *Louveaux, et al. (1978)*[6]. Esta metodología plasma un primer proceso de preparación de las muestras, tal como define *Navarrete et al. (2016)*[7] en el artículo titulado *Espectro polínico y análisis fisicoquímico de mieles de la Región del Biobío, Chile. Gayana Bot.*.

"Para la preparación de las muestras acetolizadas, 30 g de miel fueron disueltos en 50 ml de agua destilada temperada (40 ºC) y centrifugados a 2500 rpm por 10 min en una centrífuga Eppendorf 5702R; el sedimento conteniendo los pólenes fue tratado con mezcla de acetólisis a 70 °C por 10 min, según la técnica de Erdtman (1952)[8]. Para la preparación de las muestras no acetolizadas, 10 g de miel fueron disueltos en 20 ml de agua destilada temperada (40 ºC), centrifugados a 2500 rpm por 70 ºC, 10 min; el sedimento se montó en gelatina de Kisser previamente teñida con fucsina."

Tras esto, una vez preparadas las muestras, se procederá al análisis visual de las mismas con el uso de un microscopio óptico. De esta manera, un técnico especializado deberá identificar entre 500 y 600 pólenes en la muestra, clasificándolo por su variedad. Con ello, se puede obtener el porcentaje de cada tipo de polen presente en el espectro polínico de cada muestra y, como consecuencia, determinar el tipo de miel.

Este proceso no solo es tedioso, sino que presenta dos problemas fundamentales. Por un lado, implica la necesidad de contar con un técnico cualificado con amplia experiencia en el reconocimiento e identificación de la morfología y otros aspectos del polen, cuya identificación puede estar sujeta a un criterio relativamente subjetivo. Y, por otro lado, muestra el inconveniente de que no es posible realizar una revisión parcial del trabajo a posteriori, dado que se trata únicamente de un recuento visual y la única forma de verificar la correcta ejecución de la identificación sería repetir el proceso.

A continuación, se mostrara un registro real del proceso de identificación de pólenes en una muestra (ver Figura 1.3), así como una muestra ya preparada de la miel vista por el microscopio óptico (ver Figura 1.4). Estas imágenes fueron cedidas por parte del laboratorio LABMIEL de la Universidad Politécnica de Valencia.

23062 Bipea					Pólenes	%
Brassicaceas	44	45	51	60	200	38
Rosaceas					0	0
Robinia pseudoacacia					0	0
					0	0
Otros:	79	77	72	95	323	62
Presencia:				Total pólenes	523	

23062 Bipea		Pólenes	%
Brassicaceas		0	0
Rosaceas	100	100	23
Robinia pseudoacacia		0	0
		0	0
Otros:	335	335	77
Presencia:	Total pólenes	435	

23062 Bipea		Pólenes	%
Brassicaceas		0	0
Robinia pseudoacacia		0	0
Robinia pseudoacacia	11	11	4
		0	0
Otros:	247	247	96
Presencia:	Total pólenes	258	

Figura 1.3: *Registro real, realizado en Excel del análisis polínico de una muestra de miel.*

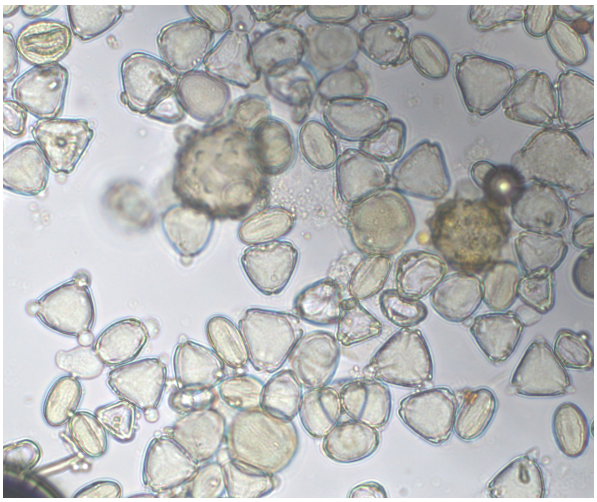


Figura 1.4: *Ejemplo de imagen de una muestra de miel preparada para realizar el análisis del espectro polínico, tomada por un microscopio óptico.*

Adicionalmente, en el apéndice D del proyecto se pueden encontrar más imágenes de muestras de miel, las cuales se emplearán posteriormente para crear el conjunto de datos.

1.3 Objetivos del proyecto

A la vista de la información presentada, se plantean una serie de objetivos para el presente proyecto con el fin de solucionar la problemática del proceso de identificación de los pólenes en la miel.

- El desarrollo de una herramienta para el etiquetado de los pólenes sobre las muestras tomadas por el microscopio óptico.
- La formación de un conjunto de datos (*dataset*) propio y específico, con muestras de múltiples mieles, para el posterior entrenamiento de las redes neuronales.
- El desarrollo de distintas arquitecturas de redes, para lograr obtener una topología que presente un rendimiento similar a las estándares, pero de menor tamaño.
- El estudio del rendimiento de redes neuronales estándar con múltiples conjuntos de datos (*dataset*), para su posterior comparación con las arquitecturas diseñadas.

- El planteamiento, desarrollo y comparación, de un sistema de predicción basado en técnicas de agrupación de redes neuronales (*ensamble*), para mejorar el rendimiento frente al uso de redes neuronales de manera individual.

Adicionalmente, se considerará, dentro del alcance del proyecto, el desarrollo de todas las herramientas y códigos que faciliten el cumplimiento de los puntos presentados anteriormente.

1.4 Relevancia del proyecto dentro del contexto de los objetivos de desarrollo sostenibles

Los los objetivos de desarrollo sostenibles (ODS) son un total de 17 objetivos planteados para la agenda 2030 por parte de Naciones Unidas, con la finalidad principal de poner fin a la pobreza, proteger el planeta y mejorar las vidas y las perspectivas de las personas en todo el mundo.

Dada su importancia, se analizará el impacto que puede tener el presente proyecto sobre los objetivos con los que se encuentra más estrechamente relacionado:

- **ODS 2: Hambre cero**: El desarrollo del proyecto tiene como objetivo ayudar en la correcta identificación de las variedades de pólenes. Esto favorece de manera directa a una mejor calidad y seguridad alimentaria de la miel. Favoreciéndose no solo el producto de manera directa, sino todos los productos que la utilizan como uno de sus ingredientes.
- **ODS 9: Industria, Innovación e Infraestructura**: El posible desarrollo de una nueva arquitectura de CNN con una mayor precisión ya constituiría, por sí solo, una innovación en el campo de la inteligencia artificial. Adicionalmente, una mejora significativa en el peso de la red podría suponer un avance a la hora de implementarse en futuras infraestructuras.
- **ODS 12: Producción y Consumo Responsables**: De manera muy similar a los motivos descritos en el ODS 2, para lograr un consumo responsable es fundamental poder identificar y rastrear las mejores variedades de miel. De esta manera, podrá limitarse el consumo a aquellos productos que hayan demostrado ser seguros y que han seguido una producción adecuada.
- **ODS 15: Vida de Ecosistemas Terrestres**: El aporte en este punto del trabajo se basa en analizar la relación entre el análisis de pólenes de manera eficiente y el análisis de la flora de un ecosistema. La mayor o menor presencia de una determinada variedad de polen tiene una relación directa con la presencia o no de una determinada variedad de plantas en ese ecosistema.

CAPÍTULO 2
Estado del arte

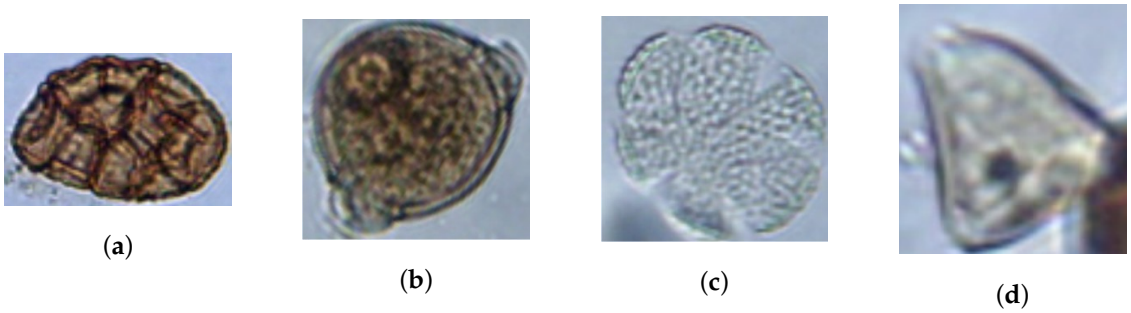
El punto de partida del proyecto será analizar, por un lado, los distintos conjuntos de datos (*datasets*) de las muestras de pólenes que hay presente en la literatura y que ya han sido utilizados anteriormente; y, por otro lado, las distintas metodologías existentes en en campo de la visión artificial para la clasificación de imágenes, profundizando especialmente en aquellas relacionadas con las CNN.

2.1 Conjuntos de datos presentes en la literatura

En la literatura, se pueden encontrar una gran diversidad de conjuntos de datos (*datasets*), que recogen muestras sobre distintas variedades de pólenes, con remarcables diferencias entre ellos. A continuación, se presentarán algunos de los más citados y, por lo tanto, los que se usarán más adelante como elementos comparativos dada su relevancia.

De ahora en adelante, en este trabajo y con el objetivo de emplear la terminología más habitual dentro de este campo, se empleará el término inglés *dataset* para referirse al término *conjuntos de datos*.

- **POLLEN23E**: Se trata de un *dataset* constituido por 805 muestras distintas de 23 variedades diferentes. Estas muestras se han obtenido de variedades de mieles propias de zonas tropicales, ya que han sido muestras recogidas de la sabana brasileña. [9]

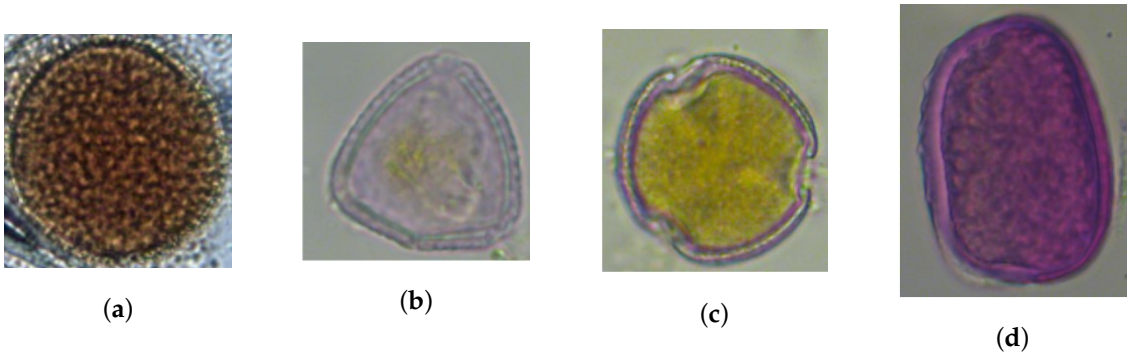


(**a**) (**b**) (**c**) (**d**)

Figura 2.1: *Imágenes del dataset POLLEN23E. Representan las variedades (a) anadenanthera, (b) dipteryx, (c) hyptis y (d) matayba*

Este *dataset* tiene un gran inconveniente, que es el bajo número de muestras que ofrece, lo cual puede afectar negativamente al entrenamiento de las redes. Por otro lado, tal como se indica en el propio artículo que lo presenta, muchas de las muestras son de variedades autóctonas de zonas tropicales, dificultando la implementación de las redes entrenadas con este *dataset* para el análisis de las muestras Españolas.

- **POLLEN73S**: Se trata de un *dataset* constituido por 2523 muestras de 73 variedades diferentes. A diferencia del *dataset* anterior, estas muestras tiene origen en el polen ambiental. [10]

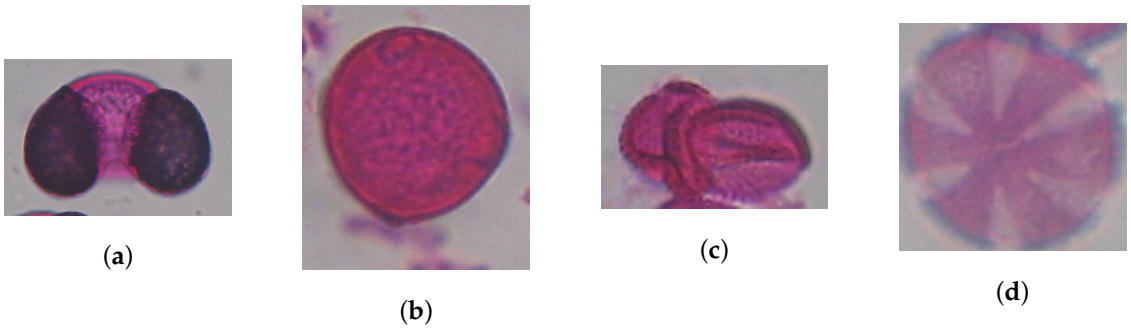


(**a**) (**b**) (**c**) (**d**)

Figura 2.2: *Imágenes del dataset POLLEN73S. Representan las variedades (a) croton, (b) erythrina mulungu, (c) genipa auniricana y (d) palmaira*

Este *dataset* es algo más extenso, sin embargo, las muestras no están tomadas del mismo medio en el que se usarán las redes y algunas de las imágenes que contienen han sido procesadas artificialmente, añadiendo pigmentaciones rosadas

- **Cretan Pollen**: Se trata de un *dataset* constituido por 4025 muestras de 20 variedades diferentes. Estas son variedades de polen de origen ambiental, al igual que el dataset anterior, pero obtenidas de Creta, Grecia. En este caso, los granos de polen fueron teñidos con una solución de fucsina. [11]

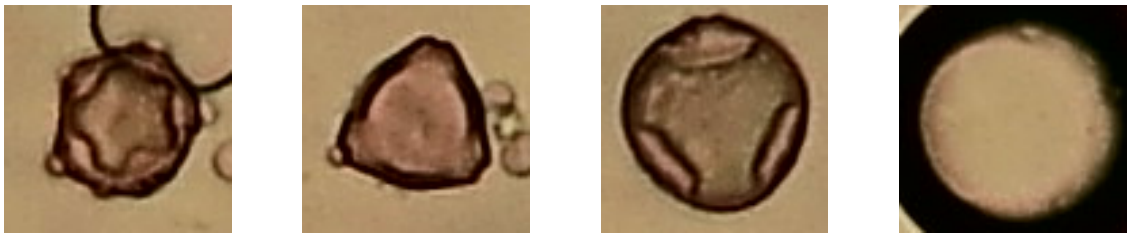


(**a**) (**b**) (**c**) (**d**)

Figura 2.3: *Imágenes del dataset Cretan Pollen Grains. Representan las variedades (a) pinus, (b) pistacia, (c) sinapis y (d) tymbra*

Este *dataset* ya empieza a presentar un número de muestras por tipo de polen apropiado para realizar el entrenamiento de una red. Sin embargo, de la misma manera que el anterior, son pólenes de origen ambiental y además han sido fuertemente teñidos. Todo esto hace que la red resultante del entrenamiento efectuado con este dataset no pueda ser empleada para la identificación de mieles no procesadas.

- **POLLEN13K**: Este *dataset* tiene su origen en le "Pollen Grain Classification Challenge 2020", un concurso que tenía como objetivo desarrollar la red con el mayor porcentaje de aciertos bajo el mismo *dataset*. El POLLEN13K está compuesto por 13.000 imágenes obtenidas de muestras aerobiológicas, clasificados en cuatro categorías. Además, es importante mencionar que una de las cuatro categorías es la de "escombros", que considera la presencia de objetos que no son granos de polen (como ramas, o burbujas, entre otros).[12]



(**a**) (**b**) (**c**) (**d**)

Figura 2.4: *Imágenes del dataset POLLEN13K. Representan las variedades (a) "Normal Pollen", (b) "Anomalous Pollen", (c) "Alnus" y (d) "Cuprissaceae"*

Este *dataset* es el el más apropiado por el número de imágenes, pero al estar pensado en el contexto del concurso, los pólenes solo están divididos en 4 grupos. Esto hace que las redes entrenadas con este *dataset* no sirvan para la identificación que se necesita en el contexto del presente trabajo.

Aunque existen muchos otros *dataset* en la literatura, en lo que respecta a pólenes de origen apícola, la cifra se reduce considerablemente. Los pocos que existen presentan un número de muestras por tipo están lejos de ser las ideales para el correcto entrenamiento de una red neuronal. Por lo tanto, es necesario comenzar este proyecto por desarrollar un *dataset* propio que cuente con la cantidad, variedad y origen de muestras ideal para la función para la que se desea diseñar la red neuronal. Pese a ello, si se emplearán estos *dataset* como elementos comparativos.

2.2 Metodología

Desde el nacimiento de las cámaras de fotos y vídeo electrónicas en las década de los setenta, ha aparecido prácticamente en paralelo a ellas el campo de estudio de la visión artificial. Su objetivo principal es el procesado y análisis de imágenes para su posterior interpretación por parte de un ordenador, guardando similitud en concepto con el sentido de la vista humano.

Hoy en día, la visión artificial se ha convertido en una tecnología muy importante a nivel industrial en muchos campos. Algunos de los más destacables incluyen su implementación en robótica, específicamente en tareas de *'pick and place'* para la manipulación de objetos. Debido a sus grandes avances, la visión artificial está cada vez más presente otros campos como la automoción, para la conducción de vehículos de manera autónoma, así como en la agricultura, para la monitorización del estado de las plantaciones. Como se puede apreciar, sus usos son muy diversos, sin embargo, en este trabajo, el foco se mantendrá en torno a aquellos planteamientos asociados a la clasificación de objetos, en particular, variedades de polen de la miel.

2.2.1. Planteamiento clásico basado en extracción de características

La extracción de características es uno de los planteamientos y campos de estudio más relevantes dentro de la inteligencia artificial.

El planteamiento de la técnica es relativamente sencillo: consiste en extraer de cada imagen el mayor número de características posibles, pertenecientes a los campos de color del objeto, textura o forma. Algunas de las características más utilizadas son: color, área, perímetro, centroide, entre muchos otros.

Para calificar este tipo de técnicas, no todas las características pueden ser igual de relevantes a la hora de identificar un elemento en concreto, por lo que es importante buscar aquellas que cumplan las siguientes propiedades:

- **Invarianza:** Se han de emplear aquellas características que no varíen la naturaleza del objeto representado ante deformaciones o transformaciones.
- **Capacidad discriminante:** Se deben de usar las características que resalten lo máximo posible los aspectos diferenciales entre objetos de clases distintas.
- **Precisión:** Se deben emplear las características de los objetos de tal manera que, dentro de una misma clase, representen la menor dispersión.
- **Incorrelación:** Se debe tener la máxima información con el mínimo número de características, por lo que se deben eliminar las características que dependen fuertemente entre sí, ya que no añaden información.
- **Concisión:** Las características deben poder representarse con pocas primitivas.

Algunas características empleadas en proyectos para la identificación de pólenes son el área, longitud, ancho, perímetro, MaxFeret90 (que indica la longitud de proyección perpendicular a la proyección máxima de Feret) y la relación entre el ancho y el largo. Estas características son las empleadas por *Zdeňka Javůrková et al. (2021)* [16] en su artículo titulado *"NUMERICAL METHODS AND IMAGE PROCESSING TECHNIQUES FOR MELISSOPALYNOLOGICAL HONEY ANALYSIS"*. En la siguiente figura (ver Figura 2.5), se puede apreciar las características aplicadas sobre una muestra de polen tipo Myositis sp.

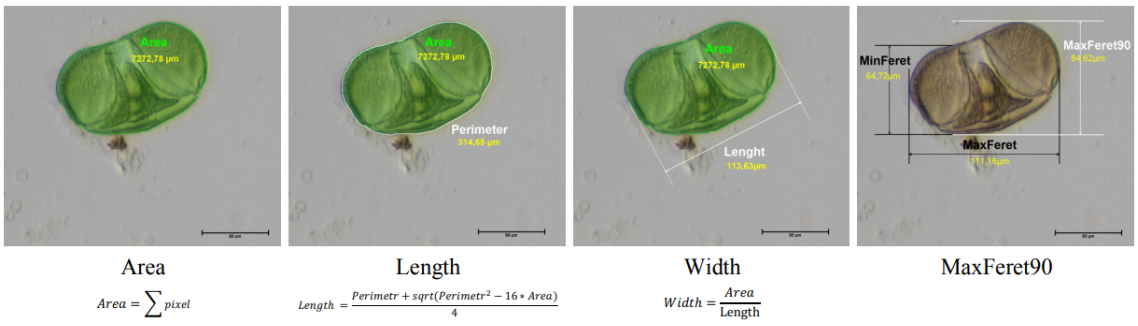


Figura 2.5: *Imagen que muestra la obtención de de características geométricas sobre un polen del tipo Myositis sp. obtenido del artículo de Zdeňka Javůrková et al. (2021) [16]*

Una vez seleccionadas las características, es necesario realizar un proceso de etiquetado, es decir, relacionar cada imagen con un tipo posible y, por lo tanto, asociar cada imagen con las características propias de su tipo.

A continuación, para resolver la parte de la identificación, es posible el planteamiento de distintas técnicas. En el artículo titulado *Automated pollen identification using microscopic imaging and texture analysis* de *J. Victor Marcos et al. (2015)* [18], se utilizan dos sistemas de identificación basados en algoritmos de aprendizaje supervisados.

Otro planteamiento para el proceso de identificación ya empleando redes neuronales es el descrito por *J. Arroyo Hernández et al. (2013)* en su artículo titulado *SYSTEM DETECTION AND AUTOMATIC CLASSIFICATION OF POLLEN GRAIN APPLIES TECHNICAL DIGITAL IMAGING PROCESS* [50]. Este artículo propone la clasificación de pólenes mediante una red neuronal de tipo feed-forward. Esta red empleará como función de activación la función hiperbólica tangencial sigmoidal y, tal como describe el artículo, el mejor número de capas ocultas para la identificación de las 11 clases con las que trabajan oscila entre 20 y 80. En la siguiente figura (ver Figura 2.6), se describe de manera visual la topología de la red neuronal feed-forward.

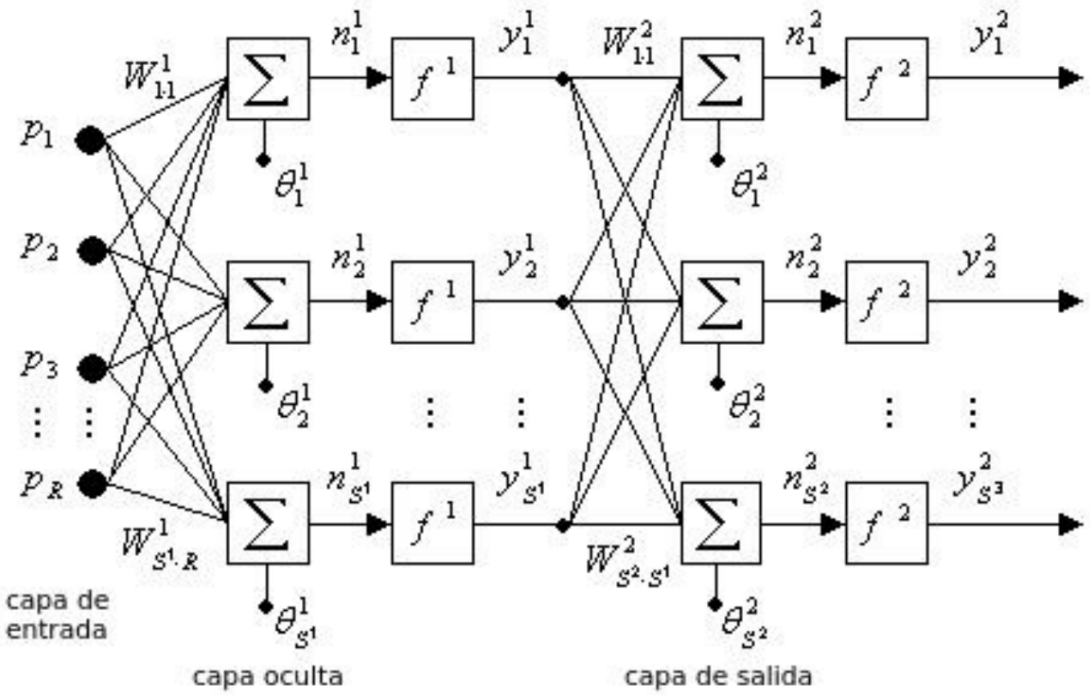


Figura 2.6: *Esquema de bloques de una red neuronal feed-forward obtenida del artículo Automated pollen identification using microscopic imaging and texture analysis de J. Victor Marcos et al. (2015) [17]*

2.2.2. Planteamiento con redes convolucionales

Las redes neuronales convolucionales (CNN) han sido uno de los campos dentro de la visión artificial que más han crecido en los últimos años. Este crecimiento se debe, en gran medida, a su capacidad para la identificación de características y patrones visuales.

Su funcionamiento se basa principalmente en realizar de manera reiterada procesos de convolución. Esto consiste en aplicar un filtro o *Kernel* de valores variables, en el proceso de entrenamiento sobre una imagen. Con la sucesión de la aplicación de este filtrado de la imagen, se permite generar un mapa de características en el que se resaltan los elementos más diferenciadores de las imágenes. Estos mapas son alterados y ajustados a través de la aplicación de otro tipo de capas según las necesidades para las que se desea emplear la red, ya sea detección de objetos, segmentación o clasificación de objetos, como es el caso.

Dentro del uso de las CNN, para las distintas aplicaciones existen distintas arquitecturas basadas principalmente en tres factores: el número de capas que la definen, la distribución y secuenciación de esas capas a lo largo de la red y, por último, la función de dichas capas. Por lo tanto, antes de presentar las principales arquitecturas de la literatura, es importante analizar qué función tienen las capas que las componen:

- **Capas convolucionales:** Estas capas son el tipo principal de las redes CNN y están compuestas por el filtro o *Kernel* descrito anteriormente. En general, las capas convolucionales se encuentran a lo largo de todas las arquitecturas. De manera general, se puede apreciar que las primeras capas son las encargadas de identificar los elementos más generales de la imagen, mientras que las últimas extraen los detalles y son las más importantes en las tareas de identificación. Tal es así, que una estrategia usada en el entrenamiento de una red preentrenada con el menor número de épocas es congelar los valores por defecto de las primeras capas y solo entrenar las últimas. La siguiente figura (ver Figura 2.7) describe de manera visual el funcionamiento de este tipo de capas.

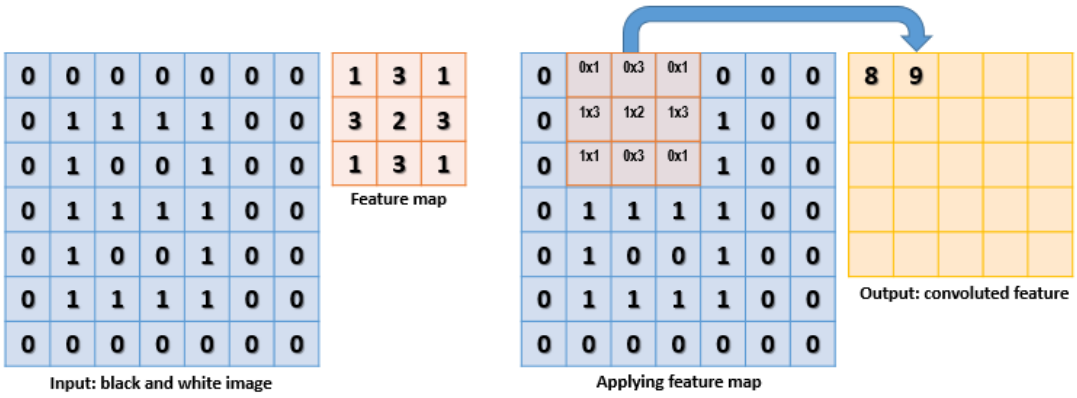


Figura 2.7: *Imagen representativa del funcionamiento de una capa convolucional. [20]*

- **Capas de pooling:** Estas capas tienen como objetivo principal reducir la dimensionalidad de las matrices de características. Esto se usa principalmente para disminuir el número de parámetros de la red y, como consecuencia, hacerla más eficiente. El *pooling* se realiza sobre una determinada área de la imagen y se obtiene un valor representativo de esa área, con el cual se construirá una nueva matriz de características con esos valores.

  Existen principalmente dos técnicas para obtener el valor representativo. Por un lado, las capas de *Max Pooling* son aquellas que toman el valor más alto entre los valores de la sección. Por otro lado, se pueden emplear las capas de *Average Pooling*, que toman la media entre los valores de la sección. A continuación, se adjunta una figura para explicar de manera más visual su funcionamiento (ver Figura 2.8).

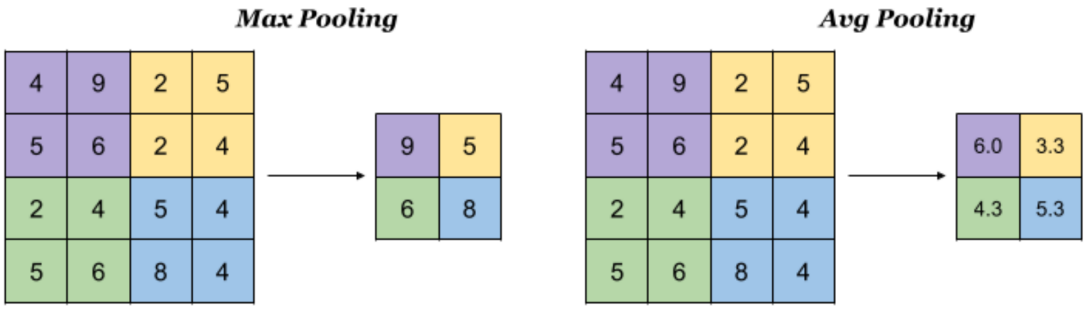


Figura 2.8: *Imagen representativa del funcionamiento de una capa Max pooling y Average pooling. [21]*

- **Capa de activación:** Estas capas se suelen emplear tras las etapas de convolución y consisten en la aplicación de una función matemática que permite al modelo capturar relaciones no lineales en los datos, extrayendo así características relevantes.

  Una de las funciones de activación más usada es la ReLU (*Rectified Linear Unit*). Esta función se basa en aplicar la operación max(0, x) a cada valor en el mapa de características.

- **Capa de aplanamiento (*flatten*):** Estas capas se aplican al final de las convoluciones y antes de las capas *fully-connected*. Su función es convertir la matriz de características en un vector unidimensional. Para poder relacionar la matriz de características con las capas *fully-connected*. Se puede apreciar el funcionamiento de este tipo de capas en la figura 2.9.

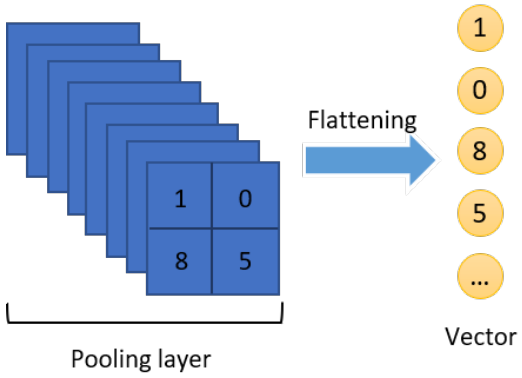


Figura 2.9: *Imagen representativa del funcionamiento de una capa de flatten. [22]*

- **Capa *fully-connected*:** Las capas *fully-connected* son capas que representan un vector de dimensión definida. Estas capas son similares a las de una red neuronal tradicional, donde cada neurona está conectada a todas las neuronas de la capa anterior. Tradicionalmente, se suelen encontrar a la salida de las redes neuronales y pueden aparecer más de una capa de este tipo conectadas en paralelo.

  También es muy habitual encontrar en las redes empleadas para clasificación una última capa de la red de tipo *fully-connected* cuya dimensión es igual al número de tipos entre los que puede distinguir la red.

  A continuación, se muestra una figura en la que se aprecia un esquema de salida de una red neuronal, en la que no solo se observa el uso de capas *fully-connected* sino también el uso de una capa de *flatten* (ver Figura 2.10).

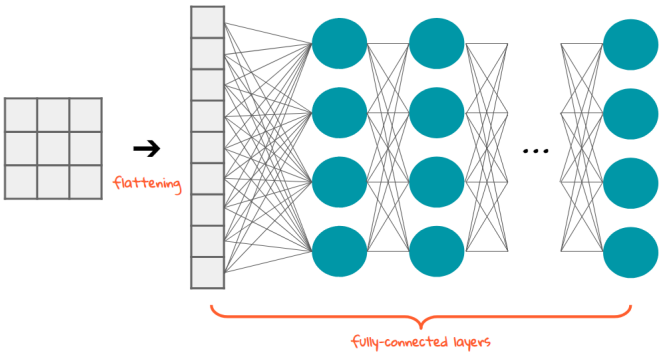


Figura 2.10: *Imagen representativa del funcionamiento de una capa de fully-connected tras la aplicacion de una capa de flatten. [23]*

- **Capa de *Batch Normalization*:** El funcionamiento de las capas de *Batch Normalization* es, como indica su nombre, normalizar las activaciones en las capas, logrando que estas tengan una media cercana a cero y una desviación estándar cercana a uno. Esto se hace con el objetivo de acelerar el entrenamiento y reducir el riesgo de sobreentrenamiento.

- **Capa de *dropout*:** Este tipo de capas se emplea normalmente para evitar el sobreentrenamiento. El funcionamiento se basaen añadir la posibilidad de desactivar algunas de las neuronas de la red de manera aleatoria, permitiendo así que el resto actualice sus pesos de manera natural en el proceso de *backpropagation*. Esto evita que una red dependa demasiado de un determinado número de neuronas, lo cual podría provocar el sobreentrenamiento.

- **Capa de concatenado:** Las capas de concatenado se usan principalmente en las arquitecturas de redes en paralelo, situándose al final de las etapas en paralelo para unir los mapas de características extraídas por cada rama en un mapa de características conjunto.

Una vez analizadas las principales capas empleadas para la definición de las topologías de las redes neuronales, se presentarán múltiples topologías relevantes en la literatura de las redes neuronales convolucionales. Además, estas redes son las que posteriormente serán empleadas durante el presente trabajo como elementos comparativos del desempeño de las redes diseñadas.

- **VGG16 y VGG19:** Estas dos redes se recogen dentro del grupo de redes VGG (*Visual Geometry Group*), presentadas por *Karen Simonyan* y *Andrew Zisserman* en el artículo *VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION* publicando en 2014. [24]

  La lógica principal tras este tipo de redes es plantear una red muy profunda y con unos filtros de convolución muy pequeños de (3x3). Esta red está diseñada para lograr extraer los detalles más locales y específicos de las imágenes. Cabe destacar que pese a la simpleza de la arquitectura, se trata de una familia de redes que ha presentado muy buenos resultados en distintas aplicaciones. La diferencia principal entre las arquitecturas VGG16 y VGG19 es el número de capas convolucionales, teniendo 16 y 19 respectivamente.

  A continuación, se mostrarán dos figuras representativas de la arquitectura VGG16 y VGG19 para facilitar su entendimiento y comparación (ver Figuras 2.11 y 2.12).

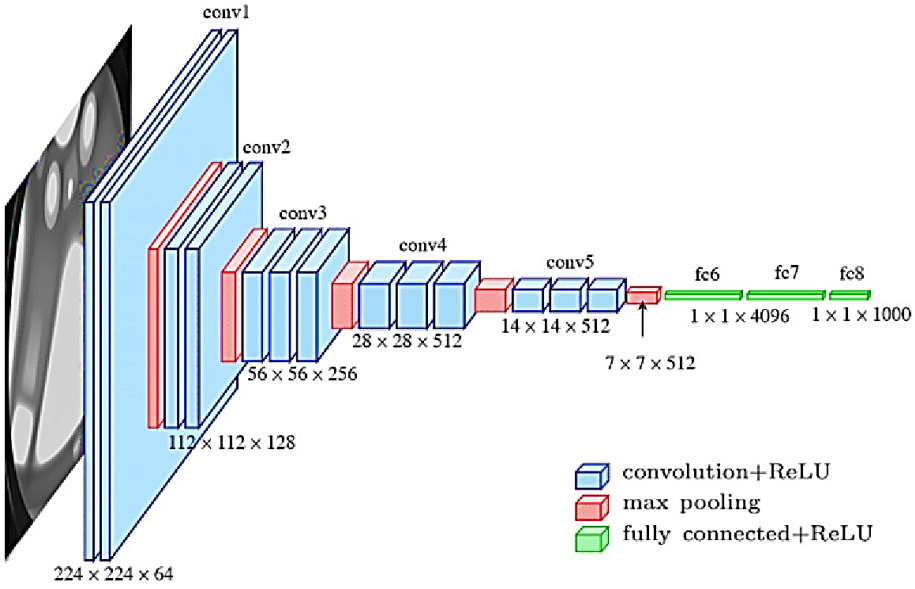


Figura 2.11: *Imagen esquemática de la arquitectura VGG16 . [25]*

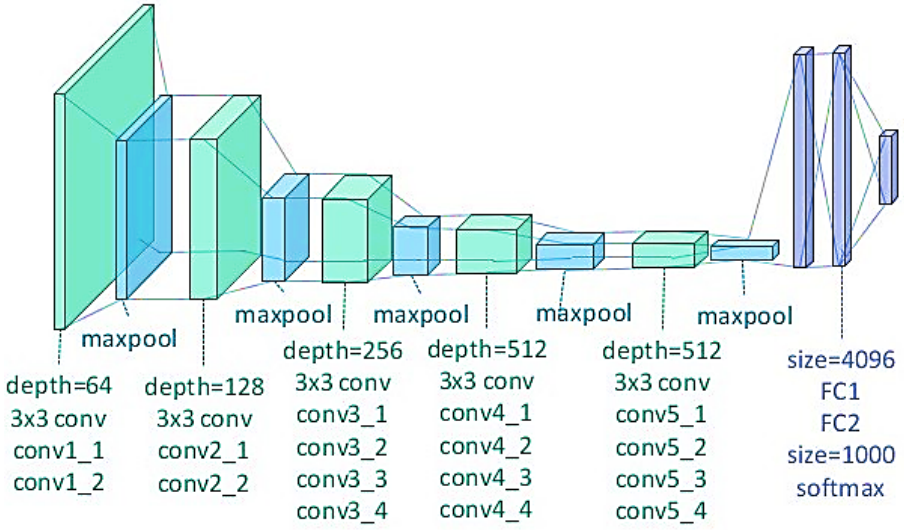


Figura 2.12: *Imagen esquemática de la arquitectura VGG19. [25]*

- **ResNet50:** La red ResNet50 presentó un cambio significativo, diferente al resto de redes, al añadir un marco de aprendizaje residual para facilitar el entrenamiento de redes mucho más profundas, siendo esta una red con 50 capas convolucionales. El aprendizaje residual consiste en trazar conexiones directas entre distintas capas sin seguir la linealidad del modelo. Esto evita el efecto de desvanecimiento por gradientes, por el cual, a medida que los datos fluyen hacia atrás durante la retropropagación, los gradientes se pueden volver demasiado pequeños a medida que se llega a las capas iniciales, provocando en las redes muy densas que las capas iniciales apenas se actualicen durante el entrenamiento.

  Esta arquitectura fue publicada por *Kaiming He, et al. (2015)* en su artículo titulado *Deep Residual Learning for Image Recognition* [26]. En la siguiente figura se puede apreciar una imagen esquemática de la arquitectura de la red, en la que se pueden apreciar las conexiones para el aprendizaje residual (ver Figura 2.13).

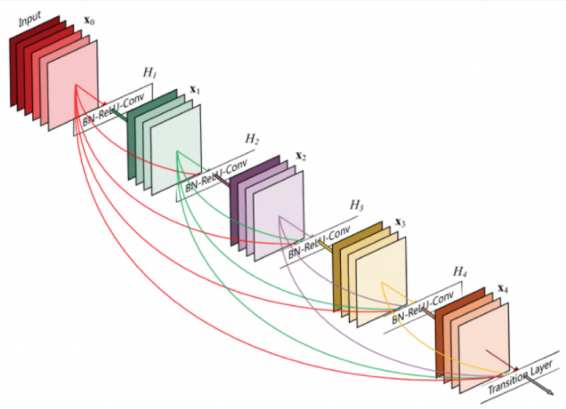


Figura 2.13: *Imagen esquemática de la arquitectura ResNet50. [27]*

- **InceptionV3:** Esta arquitectura fue presentada por *Szegedy c, et al. (2016)* en su artículo *Rethinking the inception architecture for computer vision* [28] y se basa principalmente en etapas en paralelo en las que se aplican convoluciones con *Kernels* de distintos tamaños. Esto persigue que la extracción se realice tanto para las características más generales como para las más particulares. Además, aplica muchas capas de *Batch Normalization* para poder realizar entrenamientos de manera más eficaz. A continuación, se adjunta una figura que describe la topología de la red (ver Figura 2.14).

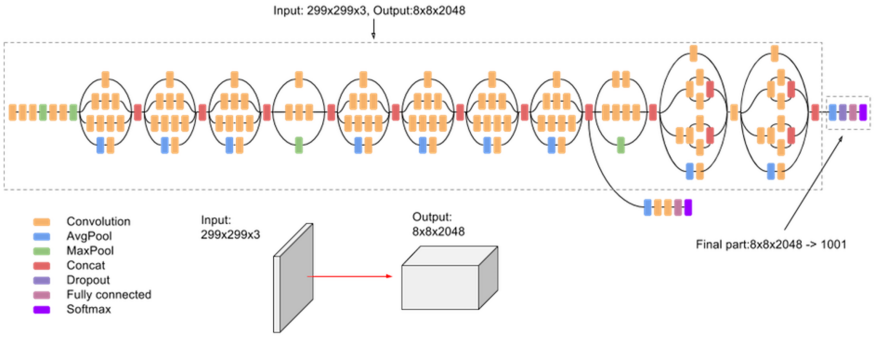


Figura 2.14: *Imagen esquemática de la arquitectura InceptionV3. [29]*

- **Xception:** La red Xceptión se diseña como un planteamiento mixto de las redes Inception. Mezclando, por un lado, estructuras convolucionales en paralelo como las de las redes Inception con etapas de convolución lineales. Esta estructura se presentó por primera vez en el artículo *Xception: Deep Learning With Depthwise Separable Convolutions* de *Francois Chollet (2017)* [30]. Esta arquitectura de red destacó por encima de la arquitectura Inception sobre todo a la hora de clasificar *datasets* grandes, probándose en el artículo con *datasets* de 350 millones de imágenes. En la figura 2.15 se puede apreciar la arquitectura Xception.

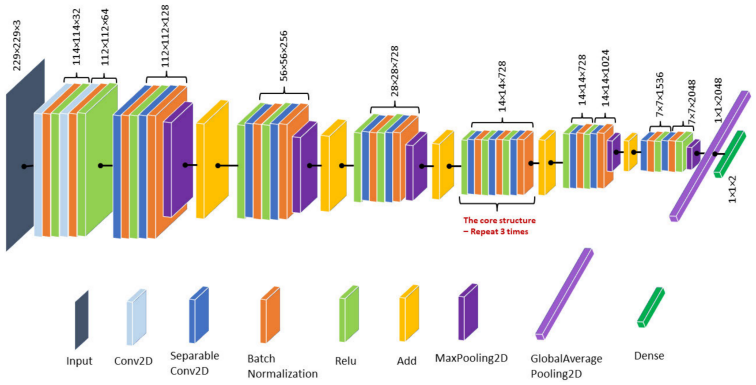


Figura 2.15: *Imagen esquemática de la arquitectura Xception. [31]*

- **DenseNet201:** La red DenseNet201 presenta un elemento diferenciador frente a otras redes basado en la conexión que realiza entre las capas. Mientras que las redes convolucionales tradicionales con X capas tienen X conexiones (una entre cada capa y su capa posterior), el sistema de conexión entre capas de la DenseNet tiene X(X+1)/2 conexiones directas. Las principales ventajas que presenta este sistema de conexión, según indica *Gao Huang, et al. (2017)* en su artículo titulado *Densely Connected Convolutional Networks* [32], son la mejora del problema del gradiente de fuga, la mejora en la propagación de características y la reducción sustancial de la cantidad de parámetros. A continuación, se adjunta una figura que describe la arquitectura de la red DenseNet201 (ver Figura 2.16).

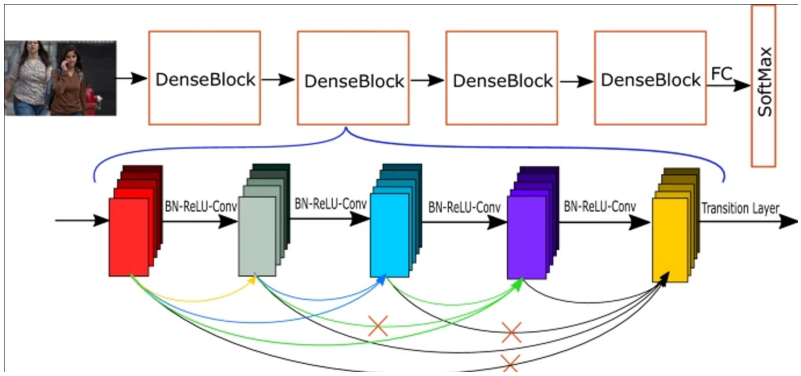


Figura 2.16: *Imagen esquemática de la arquitectura DenseNet201. [33]*

- **EfficientNetV2M:** La arquitectura de la EfficientNet destaca frente al resto de arquitecturas por proponer un nuevo método de escalado que modifica uniformemente todas las dimensiones de profundidad, ancho y resolución utilizando un coeficiente compuesto simple pero altamente efectivo. Fue presentada por Mingxing Tan, et al. (2019) en el artículo *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks* [34] y en la siguiente figura se puede apreciar su topología (ver Figura 2.17).

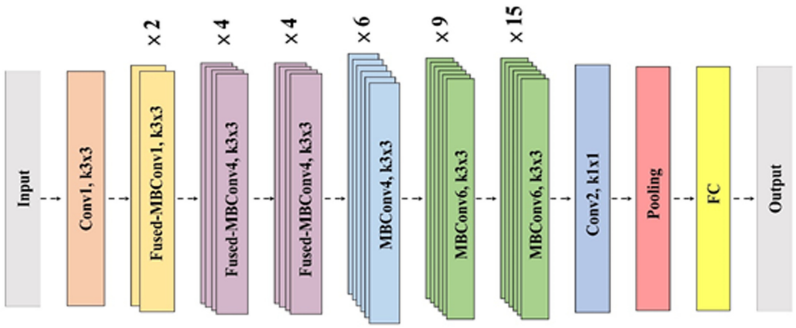


Figura 2.17: *Imagen esquemática de la arquitectura EfficientNetV2. [35]*

- **MobileNetV2:** La red MobileNetV2 es una red con una arquitectura centrada en su implementación en el mercado móvil, destacando, sobre todo, por su eficiencia computacional y su reducido número de parámetros. La arquitectura fue presentada por *Andrew Howard, et al. (2018)* en el artículo *Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation* [36] y se basa en en una estructura residual invertida donde la entrada y la salida del bloque residual son capas delgadas de cuello de botella, mientras que la capa intermedia es una representación expandida que utiliza convoluciones profundas livianas para filtrar características. Esto queda representado de manera esquemática en la figura 2.18.

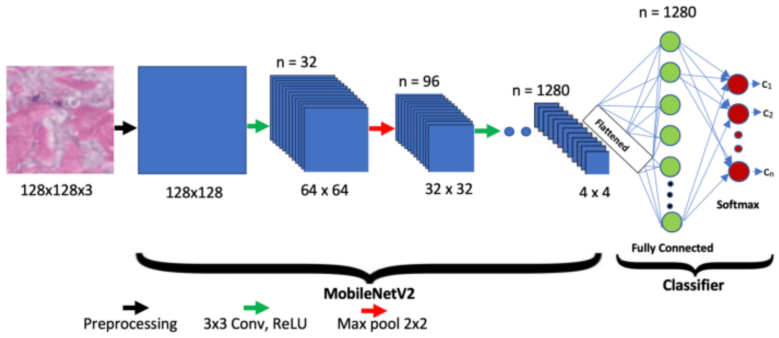


Figura 2.18: *Imagen esquemática de la arquitectura MobileNetV2. [37]*

- **NASNet:** La arquitectura de red NasNet, fue desarrollado por Google con un planteamiento distinto al resto de redes. Para lograr una arquitectura concreta, se realizó un proceso iterativo entre la cantidad y la distribución de las capas que la conforman. Tras este proceso, se llegó a los modelos que hoy se pueden encontrar implementados en herramientas como TensorFlow. En la siguiente figura se puede apreciar de manera simplificada la topología de la red (ver Figura 2.19).

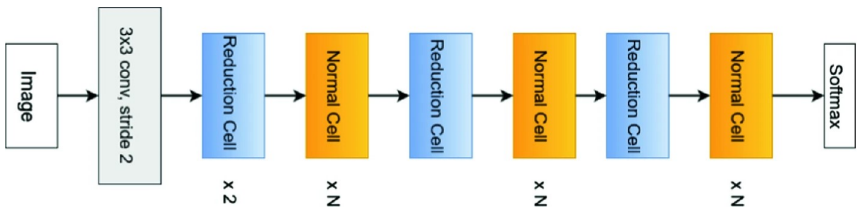


Figura 2.19: *Imagen esquemática de la arquitectura NASNet. [38]*

- **APFA_Net:** La arquitectura de Red APFA_Net, a diferencia de las arquitecturas anteriormente presentadas, no es una arquitectura pensada con un propósito general. Se trata de una estructura de capas mucho menos profunda, desarrollada específicamente para el ámbito de la clasificación de variedades de polen. Fue presentada por *Tahir Mahmood, et al (2023)* en el artículo titulado *Artificial intelligence-based classification of pollen grains using attention-guided pollen features aggregation network* [39].

  Esta red, de manera muy similar a la inceptionV3, destaca por un planteamiento con 8 etapas en paralelo, dentro de las cuales, solo se aplica una convolución, tratándose así de una red poco profunda. Además, plantea una salida en la red utilizando capas de *average pooling*, evitando así las capas de *flatten* y *fully-connected* que aumentaría drásticamente le número de hiperparámetros de la red. A continuación, se adjunta una figura representativa de la arquitectura (ver Figura 2.20).

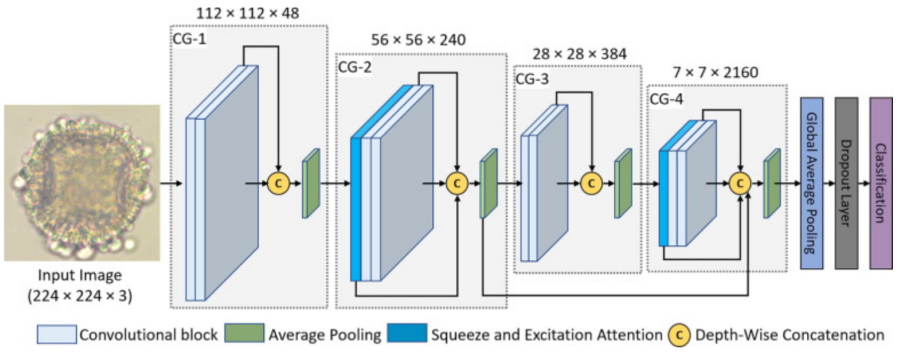


Figura 2.20: *Imagen esquemática de la arquitectura APFA_Net. [39]*

- **ViT_model:** El modelo de Vit tiene su origen en las redes *Transformers*, un tipo de redes desarrolladas originalmente para el procesado de imágenes y texto. Sin embargo, de manera muy reciente, se han empezado a emplear este tipo de redes para tareas de clasificación y detección de objetos. En el artículo titulado *Do Vision Transformers See Like Convolutional Neural Networks?*, de *Maithra Raghu, et al (2021)* [40], se que compara el uso de estas topologías con la red ResNet, y se aprecian unos resultados a nivel de rendimiento muy similares. Para entender mejor el funcionamiento de estas redes para la clasificación de imágenes, se ha añadido la siguiente figura explicativa (ver Figura 2.21).

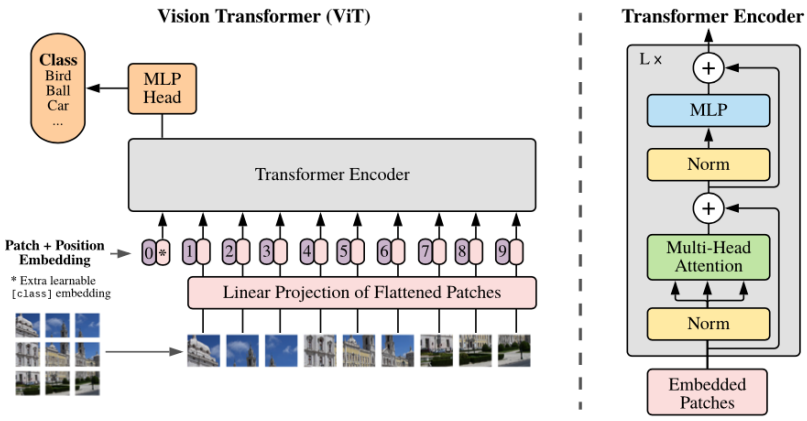


Figura 2.21: *Imagen esquemática de la arquitectura ViT_model. [41]*

Tras conocer las principales arquitecturas, es importante profundizar también en dos de las técnicas más importantes empleadas en el campo de las redes convoluciones, como son el *Transfer Learning* y el *Data Augmentation*. Estas técnicas son empleadas para mejorar el rendimiento de las CNN, especialmente cuando los conjuntos de datos son limitados y serán empleados en distintos puntos a lo largo del proyecto.

- ***Transfer Learning*:** Esta técnica tiene como función principal agilizar el proceso de entrenamiento de una red neuronal. El planteamiento es usar los pesos iniciales de una red ya entrenada con un conjunto de datos grande como pesos de partida para entrenar otra red dedicada a una tarea más especifica.

  Existen muchos artículos que defienden las ventajas de esta técnica y prueban su eficacia, como es el caso del artículo *A Comprehensive Survey on Transfer Learning* de *FUZHEN ZHUANG, et al* (2021) [54]. Tales son las ventajas del *Transfer Learning* que todas las redes dentro del paquete Keras aplican esta técnica de manera automática.

  Esta técnica presenta, a su vez, grandes resultados combinados con estrategias de congelación de capas. El funcionamiento de esta técnica consiste en fijar los pesos de las primeras capas, las cuales se centran en detectar las características generales y heredarlos de la red ya entrenada. Las épocas de entrenamiento se centran en ajustar los pesos de las últimas capas, que se especializan en identificar los detalles. Pese a ello, este trabajo se orientará en otro tipo de técnicas.

- ***Data Augmentation*:** Esta técnica consiste en crear nuevas muestras de datos a partir de otras existentes mediante la aplicación de transformaciones o modificaciones, dando como resultado un aumento en la cantidad y variedad de las imágenes empleadas en el entrenamiento.

  Son muchos los artículos dentro del campo de las CNN que hablan de las ventajas en rendimiento del uso de técnicas de *data augmentation*, como es el caso del artículo titulado *Further Advantages of Data Augmentation on Convolutional Neural Networks* de *Alex Hernández García, et al. (2018)* [49]

  Pese a las ventajas de esta técnica, cabe destacar que el uso de *data Augmentation* no elimina la necesidad de disponer de un *dataset* con un número considerable de muestras, ya que, la red puede seguir viéndose afectada por problemas de sobreentrenamiento.

2.2.3. Planteamiento mixto

Hasta ahora se han planteado estrategias para identificar imágenes delimitadas, pero dentro del campo de la visión artificial, se plantea la posibilidad de no solo clasificar las imágenes, sino ser capaz de detectar un determinado tipo de objetos dentro de una imagen general. Este enfoque se divide en dos grandes etapas: una primera de delimitación del objeto a partir de sus características y una segunda de clasificación. En la figura 2.22, se puede apreciar el planteamiento teórico del uso de este tipo de redes.

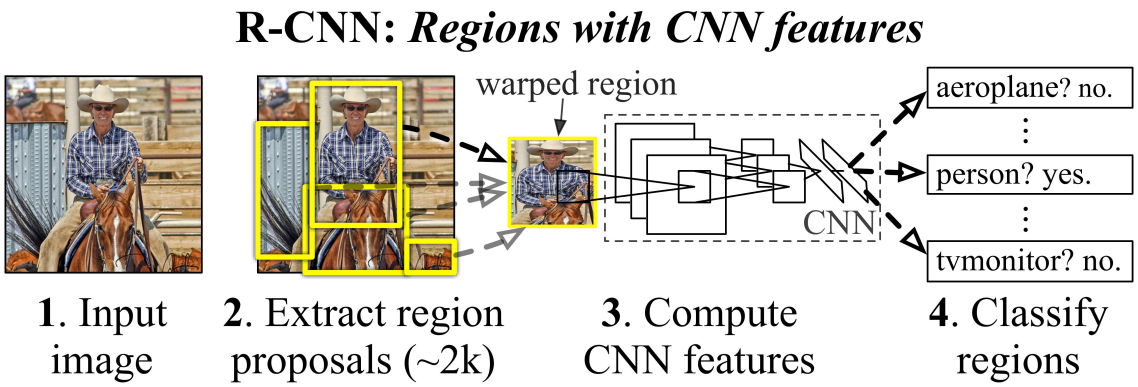


Figura 2.22: *Esquema de funcionamiento teórico de una red R-CNN. [42]*

Existen una gran variedad de artículos y proyectos con este objetivo, los cuales emplean un tipo de redes convoluciones directamente enfocadas a esta tarea y que reciben el nombre de R-CNN (Region Based Convolutional Neural Networks). Cabe destacarque este tipo de redes han ganado especial relevancia, no solo por su uso en imágenes estáticas, sino sobre todo en vídeo, permitiendo en algunos casos la identificación de elementos en tiempo real.

Un ejemplo del uso de R-CNN aplicada a la identificación en vídeo es el presentado por *Mira Park, et al. (2019)* en su artículo *Marine Vertebrate Predator Detection and Recognition in Underwater Videos by Region Convolutional Neural Network* [43], donde emplean este tipo de redes para la identificación de especies marinas, como se puede apreciar en la siguiente figura (ver Figura 2.23).

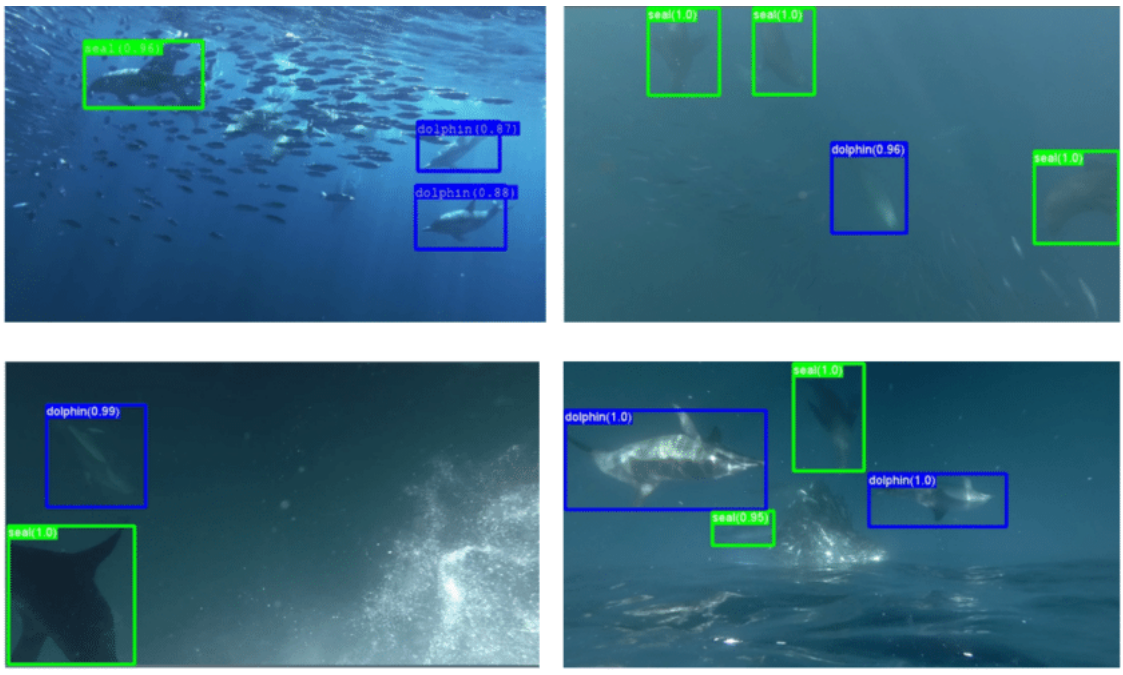


Figura 2.23: *Ejemplo de uso de la R-CNN para la identificación de especies marinas, presentado en el artículo de Mira Park, et al. (2019) [43]*

CAPÍTULO 3
Descripción del *dataset*

3.1 Herramienta para el desarrollo del *dataset*

Tras el análisis de los datasets realizado en el aparatado anterior, se llega a la conclusión de que se necesita crear un *dataset* propio para poder realizar un correcto desarrollo de las redes. Esto se debe, principalmente, a que ninguno de los conjuntos de datos analizados cuenta con las características de cantidad de muestras y origen necesarias para el objetivo del proyecto.

La herramienta para la generación del *dataset* surge ya dentro dentro del marco del proyecto **POLENET**, al igual que este trabajo. Este ensayo tiene como objetivo principal el desarrollo de un sistema de inteligencia artificial para clasificar los pólenes y cuenta con el respaldo del *Ministerio de Asuntos Económicos y Transformación Digital*.

En primera instancia, se desarrolló la aplicación *HoneyApp*. Esta aplicación, en sus primeras versiones, tenía como función principal poder delimitar, etiquetar y guardar las variedades de pólenes presentes en las imágenes obtenidas con el microscopio óptico, con el objetivo de poder solucionar uno de los principales problemas que acarrea el análisis polínico, como es la revisión de las pruebas realizadas. A continuación, se muestra una figura ilustrativa del interfaz de la aplicación *HoneyApp* para el etiquetado de pólenes (ver Figura 3.1).

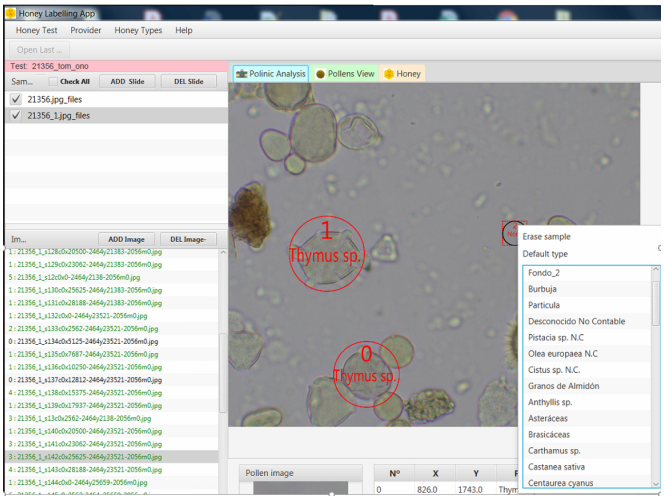


Figura 3.1: *Interfaz de la aplicación HoneyApp en la que se puede apreciar el proceso de delimitación y etiquetado de las variedades de polen.*

Esta primera versión de la aplicación se puso en funcionamiento en el laboratorio LABMIEL (Laboratorio de Control de Calidad de Miel y Productos Apícolas de la Universitat Politècnica de València, España). El laboratorio posee una acreditación para la realización de esta metodología según la norma según ISO 17025 (ISO/IEC 17025:2017,162 2017). Su papel es fundamental para el desarrollo del *dataset*, ya que gracias a las múltiples análisis polínicos que realizan a lo largo del año, han permitido ir generando un número de muestras de pólenes de diferentes tipos, suficiente para poder generar un *dataset* apropiado para el entrenamiento de las redes.

3.2 Análisis del *dataset*

En el momento de redacción de este proyecto, el *dataset* cuenta con muestras de 73 mieles distintas, dando como resultado 35626 muestras organizadas en 57 tipos distintos. A continuación, se muestra una tabla que presenta todas las muestras de de mieles analizadas, comparando el número de muestras que se ha podido identificar y el número que finalmente se ha añadido al *dataset* (ver Figura 3.2).

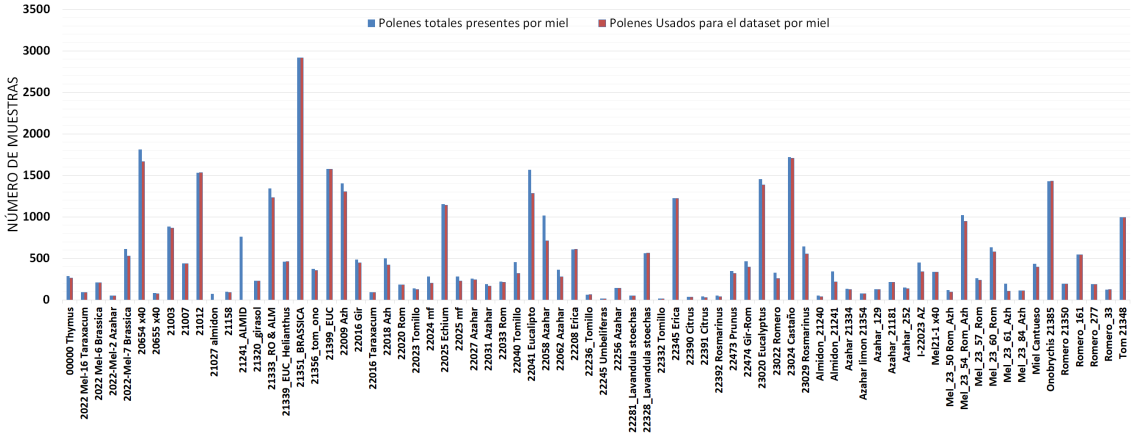


Figura 3.2: *Gráfica comparativa entre los pólenes obtenidos por tipo de miel y los usados en el dataset.*

Pese a la gran variedad de pólenes que se han etiquetado por parte del equipo del laboratorio LABMIEL, para esta primera versión del *dataset*, a causa de diversos factores, es necesario realizar un cribado de los tipos con los que contará el *dataset* que se empleará para el entrenamiento de la redes. Esto es debido a varios motivos; Por un lado, la presencia de variedades de polen con poca representatividad. Por ello, se estableció un límite mínimo de 10 muestras para cada variedad de miel y por lo que se eliminaron del estudio los tipos de *Erica sp.* o *Lotus corniculatos*, entre otras.

Por otro lado, dada la posible confusión que algunas muestras de la misma variedad pueden generar, al presentar diferencias significativas entre una y otra, también se optó por eliminar los tipos *Desconocidos contables* o *Desconocidos incontables*, ya que se tratan de familias de pólenes, las cuales recogen muchas variedades distintas en si mismas. Cabe destacar que ningún tipo eliminado en este proceso de cribado es determinante para el clasificado de la variedad de miel en el análisis polínico, sino que se tratan de elementos o tipos de pólenes tan poco representativos que no son tenido en cuenta en el análisis.

A continuación, se mostrará una gráfica en la que aparecen los tipos de pólenes empleados y la cantidad de muestras de las que se dispone de cada uno de ellos (ver Figura 3.3).

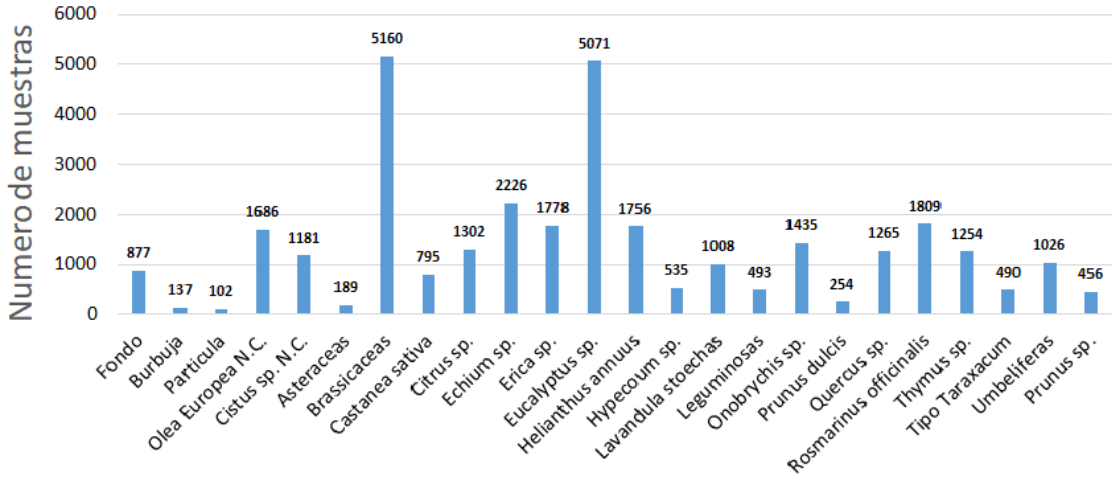


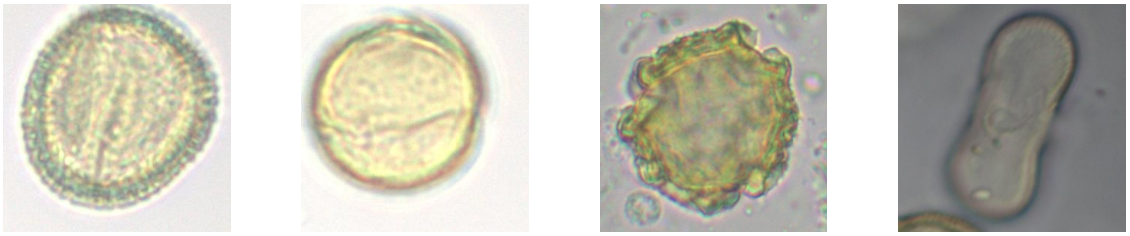
Figura 3.3: *Gráfica representativa de los pólenes empleados para el entrenamiento de las redes y el número de muestras de cada tipo.*

Es importante analizar las ventajas de este *dataset* frente a los presentados anteriormente. Este *dataset* cuenta con un total de 32297 muestras, divididas entre 29 tipos de elementos distintos. Todos los tipos de pólenes cuentan con al menos 189 muestras, llegando los tipos más comunes como el *Brassicaceas*, a las 5160. Además, todas las muestras, como ya se ha mencionado anteriormente, tienen su origen en muestras de miel. Por lo tanto, coincide de forma directa el medio de obtención de las muestras con el medio en el que se plantean utilizar las redes una vez ya hayan sido entrenadas.

3.3 Preparación del *dataset* para el entrenamiento

Una vez definido el total de imágenes que se emplearan para el entrenamiento es importante realizar una preparación previa antes de realizar el entrenamiento. Por un lado, es necesario una división en tres grupos de imágenes. El más grande, el grupo en entrenamiento, se empleará para el entrenamiento de la red y constituirá el 80 % de todas las imágenes de las que se disponga. El segundo, el grupo de validación, se empleará para validar en cada época del entrenamiento si el proceso se esta realizando de manera apropiada y la red no esta sufriendo problemas de sobreentrenamiento, y constituirá un 10 % de todas las imágenes. Por ultimo el, grupo de test corresponderá al 10 % restante de las imágenes, y su objetivo será realizar una ultima prueba para asegurar que la red está funcionando de manera apropiada y es capaz de clasificar imagines distintas a las empleadas para el entrenamiento.

A continuación, se mostrarán una serie de figuras representativas de algunas de las muestras que se pueden encontrar en el dataset desarrollado:



(**a**) (**b**) (**c**) (**d**)

Figura 3.4: *Imágenes del dataset propio. Representan las variedades (a) Olea Europea N.C., (b) Quercus sp., (c) Tipo Taraxacum y (d) Umbeliferas*

Adicionalmente, para realizar una comprobación más precisa del rendimiento de las redes, se aplicará la metodología del *k-fold*. Esta metodología es una técnica de *cross validation* muy empleada en el campo del *machine learning*, que tiene como objetivo que las imágenes empleadas en cada grupo de entrenamiento puedan aparecer también en el resto de grupos. Para ello, se generarán *k* número de carpetas (para este caso se usarán 5), en cada una de ellas, las fotos empleadas para el test serán distintas. Se ha empleando esta metodología debido, entre otros motivos, a los resultados obtenidos por *Hastie, et al* en su libro *"The elements of statistical learning"* [14]. A continuación, se muestra la figura 3.5, la cual pretende facilitar la compresión de este concepto.

Figura 3.5: *Figura representativa del funcionamiento de la metodología de k-folding aplicada para 4 fold. [13]*

En el apéndice B.1, se puede encontrar el código empleado para la división del *dataset* y la aplicación de la técnica de *5-fold*. Este código se ha desarrollado a partir del repositorio de GitHub del usuario *Aravinda*(2023) [15], pero se ha modificado para aplicar el procedimiento de *k-fold*, ya que el código presentado en el repositorio solo realiza la división entre test, validación y entrenamiento.

CAPÍTULO 4
Desarrollo de la arquitectura

Para comenzar con el desarrollo de la arquitectura de la red en necesario partir de una topología inicial. A partir de este punto, se analizará su comportamiento y se irá iterando sobre esta para lograr mejorar su rendimiento.

La topología inicial de la red será la denominada como *"Polenet V.1."* y cuenta con una estructura lineal muy similar a las VGG. Está diseñada para recibir como entrada una imagen en formato RGB, es decir, una imagen de tres canales y de 256 x 256 píxeles de tamaño, la cual, la red tratará como matriz de 256x256 con tres capas, representado en cada posición de esta matriz la información de cada píxel.

La arquitectura de la red se puede dividir en dos grandes etapas. En la primera etapa, más profunda, se intercalan capas de convolución con capas de *MaxPooling*. En esta etapa, se aplican de manera escalonada más filtros en potencias de dos, desde 64 hasta 512. Posteriormente, se reduce el tamaño de la matriz en función del valor máximo. Como resultado de la última etapa de convolución y pooling, se obtiene una matriz de 8 x 8 con 512 capas.

La segunda etapa es mucho más densa y esta enfocada a obtener la salida de la red. Parte de una capa de *Fllaten*, la cual convierte la matriz de 8 x 8 con 512 capas obtenida de la última etapa de *MaxPoolin* en un vector unidimensional de 32768 valores. Esto se hace con el objetivo de poder conectarlo posteriormente con capas densas. A continuación, se disponen simultáneamente dos capas densas, completamente conectadas y con función de activación ReLU. Estas capas permiten reducir el tamaño del vector resultante del *Fllaten* de 32768 a 1000 en la primera capa y de 1000 a 200 en la segunda. Como consecuencia de esto, estas capas aglutinan la gran mayoría de parámetros a entrenar de la red, teniendo 3276900 parámetros la primera capa densa y 200200 la segunda.

Por último, la capa de salida, es en sí otra capa densa, completamente conectada, pero la función de activación empleada en este caso es la Softmax, la cual, se usa habitualmente en las redes para la clasificación y teniendo como dimensión de salida el número de posibles tipos entre los que puede detectar la red. En el caso se usar el *dataset* propio desarrollado serán 24. A continuación, se mostrará una figura representativa de la arquitectura de la red descrita (ver Figura 4.1).

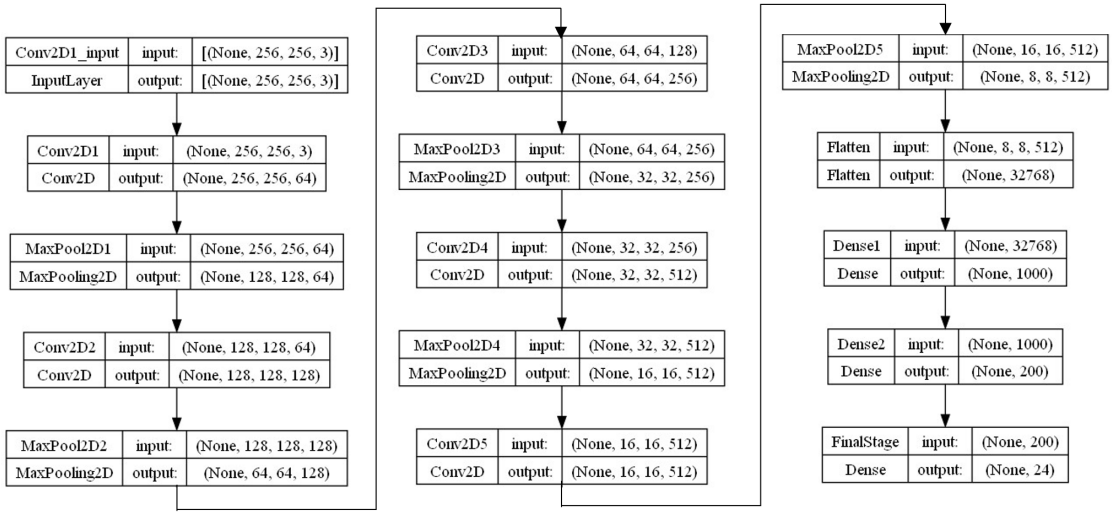


Figura 4.1: *Imagen representativa de la arquitectura de la red Polenet V.1.*

Una vez presentada la arquitectura de la red, se mostrará una tabla detallada con el número de parámetros por capa, resultante de aplicar la arquitectura para el *dataset* propio (ver Tabla 4.1).

Bloque	**Tipo de Capa**	N° **de Filtros**	**Dimensión de Salida**	N° **de Parámetros**
Input-block	Input	-	(256 x 256 x 3)	0
Conv2D1	Conv2D	64	(256 x 256 x 3)	1792
MaxPool2D1	MaxPooling2D	-	(128 x 128 x 64)	0
Conv2D2	Conv2D	128	(128 x 128 x 128)	73856
MaxPool2D2	MaxPooling2D	-	(64 x 64 x 128)	0
Conv2D3	Conv2D	256	(64 x 64 x 256)	295168
MaxPool2D3	MaxPooling2D	-	(32 x 32 x 256)	0
Conv2D4	Conv2D	512	(32 x 32 x 512)	1180160
MaxPool2D4	MaxPooling2D	-	(16 x 16 x 512)	0
Conv2D5	Conv2D	512	(16 x 16 x 512)	2359808
MaxPool2D5	MaxPooling2D	-	(8 x 8 x 512)	0
Flatten	Flatten	-	(32768)	0
Dense1	Dense	-	(1000)	3276900
Dense2	Dense	-	(200)	200200
FinalStage	Dense	-	(24)	4824

Tabla 4.1: Descripción detallada de las capas de la red Polenet V.1. con el número de parámetros

4.1 Arquitecturas propuestas

4.1.1. Red 1: Aumento de capas convolucionales

Para la primera prueba de mejora de la red, se procederá a hacer la red mucho más profunda de lo que originalmente es, intentando asemejarla aún más a las arquitecturas VGG. Esto se pretende conseguir añadiendo 3 capas convolucionales más y cambiando las dimensiones de la capa Conv2D5 del modelo anterior, ya que el tamaño de la salida es redundante en el número de capas con la anterior y se pasará a una capa con 1024 capas, siendo la salida ahora de 16 x 16 x 1024.

Tras esta etapa de aumento de las capas, que ya estaba presente en la primera versión, las tres nuevas capas comenzarán un proceso de disminución, dejando la salida de la última capa antes del *Flatten* con unas dimensiones de 2 x 2 x 128.

Esta prueba persigue obtener una mejor precisión en la red, al hacerla mucho más profunda, ya que, por lo que respecta al número de parámetros, se verá incrementado radicalmente. En la figura 4.2, se pueden ver representadas de manera visual los cambios realizados.

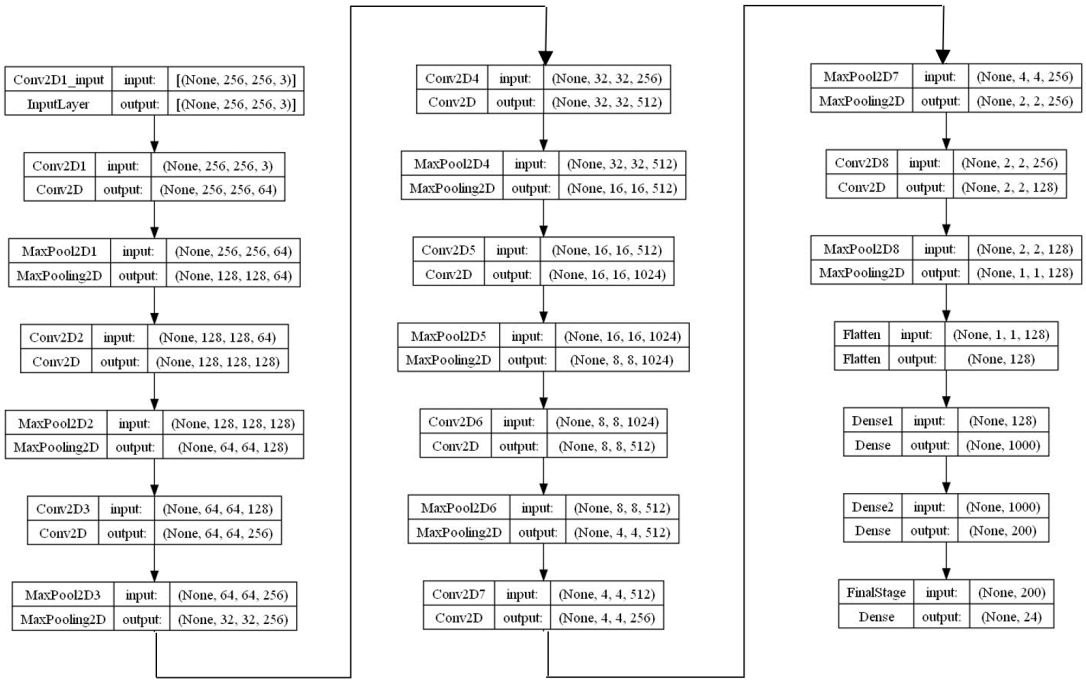


Figura 4.2: *Imagen representativa de la arquitectura propuesta para la primera red.*

Adicionalmente, se incluirá la tabla 4.2 por un lado, para poder entender mejor la arquitectura y, por otro lado, para comprobar los efectos que ha tenido el hecho de aumentar considerablemente la densidad de la red, tanto en el número de parámetros como en las dimensiones de salida.

Bloque	Tipo de Capa	N° de Filtros	Dimensión de Salida	N° de Parámetros
Input-block	Input	-	(256 x 256 x 3)	0
Conv2D1	Conv2D	64	(256 x 256 x 64)	1792
MaxPool2D1	MaxPooling2D	-	(128 x 128 x 64)	0
Conv2D2	Conv2D	128	(128 x 128 x 128)	73856
MaxPool2D2	MaxPooling2D	-	(64 x 64 x 128)	0
Conv2D3	Conv2D	256	(64 x 64 x 256)	295168
MaxPool2D3	MaxPooling2D	-	(32 x 32 x 256)	0
Conv2D4	Conv2D	512	(32 x 32 x 512)	1180160
MaxPool2D4	MaxPooling2D	-	(16 x 16 x 512)	0
Conv2D5	Conv2D	1024	(16 x 16 x 1024)	4719616
MaxPool2D5	MaxPooling2D	-	(8 x 8 x 1024)	0
Conv2D6	Conv2D	512	(8 x 8 x 512)	4719104
MaxPool2D6	MaxPooling2D	-	(4 x 4 x 512)	0
Conv2D7	Conv2D	256	(4 x 4 x 256)	1179904
MaxPool2D7	MaxPooling2D	-	(2 x 2 x 256)	0
Conv2D8	Conv2D	128	(2 x 2 x 128)	295040
MaxPool2D8	MaxPooling2D	-	(1 x 1 x 128)	0
Flatten	Flatten	-	(128)	0
Dense1	Dense	-	(1000)	129000
Dense2	Dense	-	(200)	200200
FinalStage	Dense	-	(24)	4824

Tabla 4.2: Descripción detallada de las capas de la arquitectura propuesta para la primera red.

4.1.2. Red 2: Cambio de tamaño en los *Kernels* y *poolings*

A la vista de las arquitecturas analizadas en el estado del arte de este proyecto, se puede apreciar que la variación del tamaño de los *Kernels* y las capas de pooling es un factor muy explotado a la hora de desarrollar arquitecturas. Como en el caso de la InceptionV3, donde se combinan en paralelo distintos tipos de tamaño de *Kernel*. Esto, teóricamente, se explica porque el uso de unos tamaños de *Kernel* más grandes implican una mayor capacidad de la red para extraer patrones, pero a cambio se sacrifica su capacidad para identificar detalles en las imágenes.

Por lo tanto, en esta prueba se intentara comprobar qué efectos tiene el tamaño de los filtros en el comportamiento de la red. Para ello, se usará el número de capas y distribución de la Polenet V.1, pero cambiando el tamaño de los *Kernel* de las capas convolucionales a 5x5 y el de las capas de pooling a 3x3, tal y como muestra la figura 4.3.

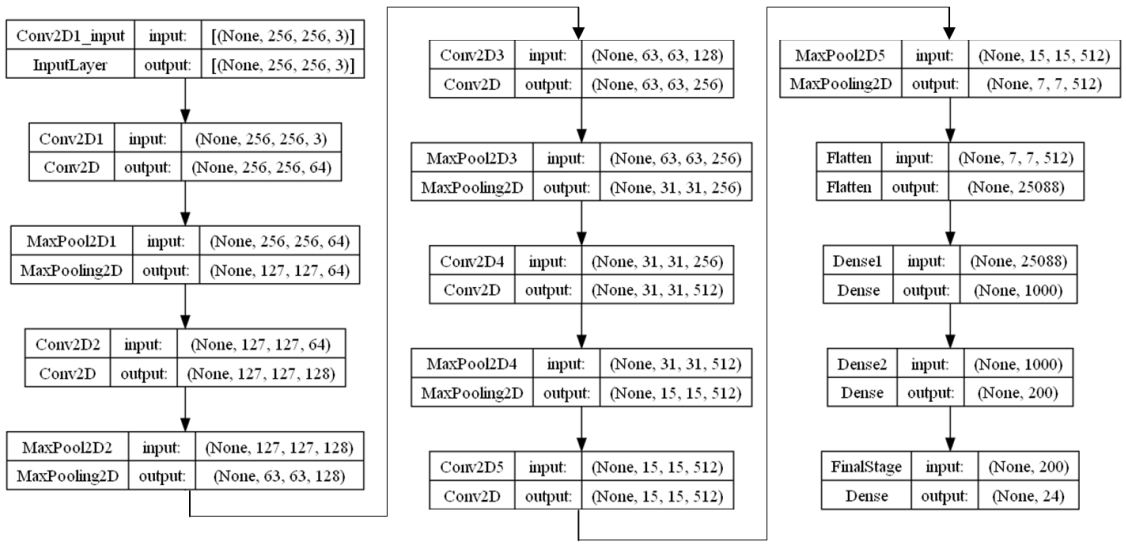


Figura 4.3: *Imagen representativa de la arquitectura propuesta para la segunda red.*

También se mostrará la tabla 4.3, la cual desglosa de la arquitectura de la segunda red.

Bloque	**Tipo de Capa**	**N° de Filtros**	**Dimensión de Salida**	**N° de Parámetros**
Input-block	Input	-	(256 x 256 x 3)	0
Conv2D1	Conv2D	64	(256 x 256 x 64)	4864
MaxPool2D1	MaxPooling2D	-	(127 x 127 x 64)	0
Conv2D2	Conv2D	128	(127 x 127 x 128)	204928
MaxPool2D2	MaxPooling2D	-	(63 x 63 x 128)	0
Conv2D3	Conv2D	256	(63 x 63 x 256)	819456
MaxPool2D3	MaxPooling2D	-	(31 x 31 x 256)	0
Conv2D4	Conv2D	512	(31 x 31 x 512)	3277312
MaxPool2D4	MaxPooling2D	-	(15 x 15 x 512)	0
Conv2D5	Conv2D	512	(15 x 15 x 512)	6554112
MaxPool2D5	MaxPooling2D	-	(7 x 7 x 512)	0
Flatten	Flatten	-	(25088)	0
Dense1	Dense	-	(1000)	25089000
Dense2	Dense	-	(200)	200200
FinalStage	Dense	-	(24)	4824

Tabla 4.3: Descripción detallada de las capas de la arquitectura propuesta para la segunda red.

4.1.3. Red 3: Aumento de capas convolucionales y cambio de tamaño en los *Kernels* y *poolings*

Esta prueba tiene como objetivo combinar las dos primeras redes. Todo esto con el objetivo de, al aumentar el tamaño de los *Kernel*, permitir a cada capa convolucionar percibir más información por cada sección, ya que agrupa la convolución de más píxeles, permitiéndole identificar patrones mucho más grandes. Como ya se ha mencionado anteriormente, el efecto adverso de esto es una perdida significativa de la capacidad de la red para identificar detalles. Por lo tanto, para mejorar esto, se optará por hacer la red más profunda, permitiendo, teóricamente, un mayor reconocimiento de patrones sin sacrificar la identificación de detalles. Por lo que la arquitectura resultante quedaría de la siguiente forma (ver Figura 4.4).

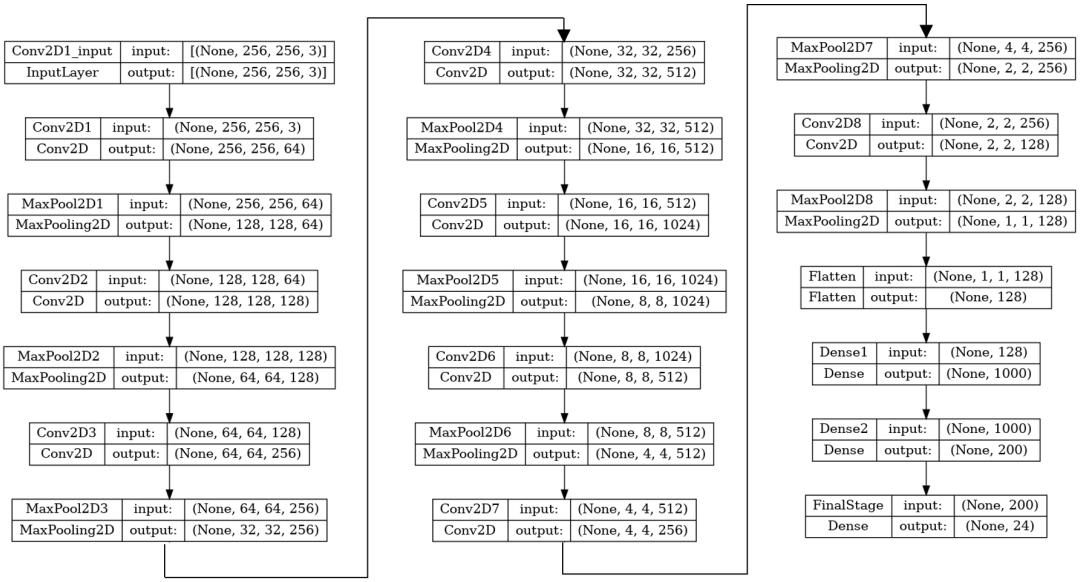


Figura 4.4: *Imagen representativa de la arquitectura propuesta para la tercera red.*

A continuación, se adjuntará la tabla 4.4 para comprobar qué efecto ha tenido sobre la salida el cambio de tamaño de los *Kernels* en esta nueva red mucho más profunda.

Bloque	**Tipo de Capa**	N° **de Filtros**	**Dimensión de Salida**	N° **de Parámetros**
Input-block	Input	-	(256 x 256 x 3)	0
Conv2D1	Conv2D	64	(256 x 256 x 64)	4864
MaxPool2D1	MaxPooling2D	-	(128 x 128 x 64)	0
Conv2D2	Conv2D	128	(128 x 128 x 128)	204928
MaxPool2D2	MaxPooling2D	-	(64 x 64 x 128)	0
Conv2D3	Conv2D	256	(64 x 64 x 256)	819456
MaxPool2D3	MaxPooling2D	-	(32 x 32 x 256)	0
Conv2D4	Conv2D	512	(32 x 32 x 512)	3277312
MaxPool2D4	MaxPooling2D	-	(16 x 16 x 512)	0
Conv2D5	Conv2D	1024	(16 x 16 x 1024)	13108224
MaxPool2D5	MaxPooling2D	-	(8 x 8 x 1024)	0
Conv2D6	Conv2D	512	(8 x 8 x 512)	13107712
MaxPool2D6	MaxPooling2D	-	(4 x 4 x 512)	0
Conv2D7	Conv2D	256	(4 x 4 x 256)	3277056
MaxPool2D7	MaxPooling2D	-	(2 x 2 x 256)	0
Conv2D8	Conv2D	128	(2 x 2 x 128)	819328
MaxPool2D8	MaxPooling2D	-	(1 x 1 x 128)	0
Flatten	Flatten	-	(128)	0
Dense1	Dense	-	(1000)	129000
Dense2	Dense	-	(200)	200200
FinalStage	Dense	-	(24)	4824

Tabla 4.4: Descripción detallada de las capas de la arquitectura propuesta para la tercera red.

4.1.4. Red 4: Cambios masivos en al arquitectura de la red

Para esta prueba se ha optado por una transformación mucho más radical en la arquitectura de la red.

En primer lugar, como ya se ha aplicado en arquitecturas anteriores, se aumentará el número de capas convolucionales para obtener una mejor precisión. Se ha pasado de 5 a 8 capas convolucionales, produciendo las cinco primeras un aumento en el número de capas de la salida y 3 disminuyéndolo, para no aumentar tan radicalmente el número de parámetro. Además, se ha probado con las capas de *Batch Normalization* entre las capas de convolución y de *MaxPooling*, ya que este tipo de capas ayudarán a estabilizar y acelerar el entrenamiento de una red neuronal.

Adicionalmente, se realizarán cambios a la salida de la red, para evitar emplear la conexión directa entre la capa de *Flatten* y la **Fully-connected**, ya que estas aumentan significativamente el número total de parámetros de la red. Por lo tanto, se sustituirá la capa de *Flatten* por una tipo *GlobalAvgPooling2D*, que en vez de convertir toda la salida en un vector unidimensional, lo convierte en un vector de longitud 128 donde cada posición representa la media de los pesos de las capas de la salida anterior. Esta capa se conectará con solo una capa **Fully-connected** y posteriormente con la salida de la red (ver Figura 4.5).

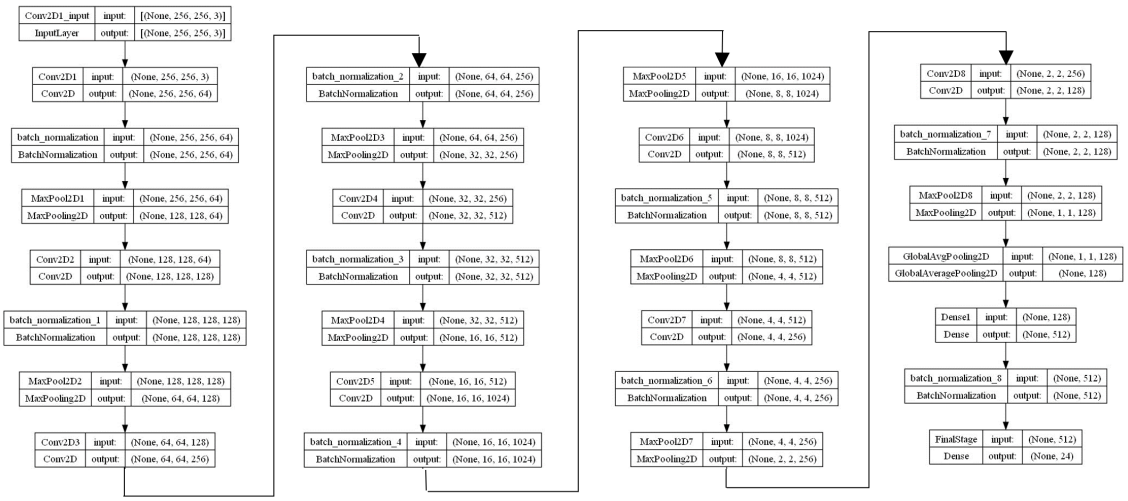


Figura 4.5: *Imagen representativa de la arquitectura propuesta para la cuarta red.*

Adicionalmente a la imagen de la arquitectura, se mostrará la tabla 4.5, para poder comprobar de manera más detallada los efectos de todos los cambios aplicados a la red.

Bloque	Tipo de Capa	N° de Filtros	Dimensión de Salida	N° de Parámetros
Input-block	Input	-	(256 x 256 x 3)	0
Conv2D1	Conv2D	64	(256 x 256 x 64)	1792
batch_norm	BatchNorm.	-	(256 x 256 x 64)	256
MaxPool2D1	MaxPooling2D	-	(128 x 128 x 64)	0
Conv2D2	Conv2D	128	(128 x 128 x 128)	73856
batch_norm_1	BatchNorm.	-	(128 x 128 x 128)	512
MaxPool2D2	MaxPooling2D	-	(64 x 64 x 128)	0
Conv2D3	Conv2D	256	(64 x 64 x 256)	295168
batch_norm_2	BatchNorm.	-	(64 x 64 x 256)	1024
MaxPool2D3	MaxPooling2D	-	(32 x 32 x 256)	0
Conv2D4	Conv2D	512	(32 x 32 x 512)	1180160
batch_norm_3	BatchNorm.	-	(32 x 32 x 512)	2048
MaxPool2D4	MaxPooling2D	-	(16 x 16 x 512)	0
Conv2D5	Conv2D	1024	(16 x 16 x 1024)	4719616
batch_normn_4	BatchNorm.	-	(16 x 16 x 1024)	4096
MaxPool2D5	MaxPooling2D	-	(8 x 8 x 1024)	0
Conv2D6	Conv2D	512	(8 x 8 x 512)	4719104
batch_norma_5	BatchNorm.	-	(8 x 8 x 512)	2048
MaxPool2D6	MaxPooling2D	-	(4 x 4 x 512)	0
Conv2D7	Conv2D	256	(4 x 4 x 256)	1179904
batch_norm_6	BatchNorm.	-	(4 x 4 x 256)	1024
MaxPool2D7	MaxPooling2D	-	(2 x 2 x 256)	0
Conv2D8	Conv2D	128	(2 x 2 x 128)	295040
batch_norm_7	BatchNorm.	-	(2 x 2 x 128)	512
MaxPool2D8	MaxPooling2D	-	(1 x 1 x 128)	0
GlobalAvgPooling2D	Glob.Av.Pooling	-	(128)	0
Dense1	Dense	-	(512)	66048
batch_norm_8	BatchNorm.	-	(512)	2048
FinalStage	Dense	-	(24)	12312

Tabla 4.5: Descripción detallada de las capas de la arquitectura propuesta para la cuarta red.

4.1.5. Red 5: Entrada con dos ramas convolucionales en paralelo

Para esta arquitectura se intentará aplicar la lógica de la topología InceptiónV3, es decir, el uso de procesos convolucionales en paralelo. En este caso, una de las ramas de convolución aplicará un filtrado ascendente de 64, 128 y 256 con un *Kernel Size* de 3x3 y la otra aplicará un filtrado también ascendente, pero de 34, 64, 128 con un *Kernel Size* de 5x5, convergiendo ambas capas en una capa de concatenado. Por lo que respecta al resto de la topología de la red, se empleará la distribución planteada en la red anterior. Esto se puede ver representado en la figura 4.6.

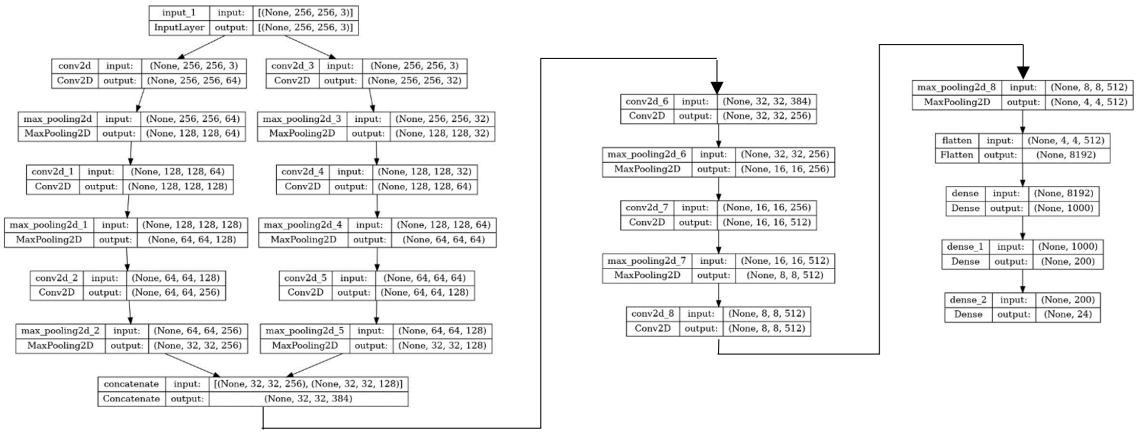


Figura 4.6: *Imagen representativa de la arquitectura propuesta para la quinta red.*

Para entender las dimensiones de cada salida, así como el número de parámetros en esta nueva arquitectura, se adjunta la tabla resumen 4.6.

Bloque	**Tipo de Capa**	**N° de Filtros**	**Dimensión de Salida**	**N° de Parámetros**
input_1	InputLayer	-	(256 x 256 x 3)	0
conv2d	Conv2D	64	(256 x 256 x 64)	1792
conv2d_3	Conv2D	32	(256 x 256 x 32)	2432
max_pooling2d	MaxPooling2D	-	(128 x 128 x 64)	0
max_pooling2d_3	MaxPooling2D	-	(128 x 128 x 32)	0
conv2d_1	Conv2D	128	(128 x 128 x 128)	73856
conv2d_4	Conv2D	64	(128 x 128 x 64)	51264
max_pooling2d_1	MaxPooling2D	-	(64 x 64 x 128)	0
max_pooling2d_4	MaxPooling2D	-	(64 x 64 x 64)	0
conv2d_2	Conv2D	256	(64 x 64 x 256)	295168
conv2d_5	Conv2D	128	(64 x 64 x 128)	204928
max_pooling2d_2	MaxPooling2D	-	(32 x 32 x 256)	0
max_pooling2d_5	MaxPooling2D	-	(32 x 32 x 128)	0
concatenate	Concatenate	-	(32 x 32 x 384)	0
conv2d_6	Conv2D	256	(32 x 32 x 256)	884992
max_pooling2d_6	MaxPooling2D	-	(16 x 16 x 256)	0
conv2d_7	Conv2D	512	(16 x 16 x 512)	1180160
max_pooling2d_7	MaxPooling2D	-	(8 x 8 x 512)	0
conv2d_8	Conv2D	512	(8 x 8 x 512)	2359808
max_pooling2d_8	MaxPooling2D	-	(4 x 4 x 512)	0
flatten	Flatten	-	(8192)	0
dense	Dense	-	(1000)	8193000
dense_1	Dense	-	(200)	200200
FinalStage	Dense	-	(24)	4824

Tabla 4.6: Descripción detallada de las capas de la arquitectura propuesta para la quinta red.

4.1.6. Red 6: Entrada con dos ramas en paralelo, una convolucional y otra lineal

Siguiendo con el planteamiento visto en la red 5, en este caso se optará por una nueva topología con las primeras capas en paralelo. Esta vez se quiere probar el unificar dos planteamientos distinto; por un lado, una de las dos ramas seguirá la misma secuencia convolucional vista en la red 5, es decir, un filtrado de ascendente de 64, 128 y 256 con un *Kernel Size* de 3x3. Por otro lado, la otra rama no presentará una estructura convolucional, sino que presentará una estructura lineal, empleando capas fully connected,

buscando asemejar su comportamiento al que tendría una red neuronal clásica como la feed-foorward. Con este enfoque, la topología de la red quedaría tal y como se representa en la figura 4.7.

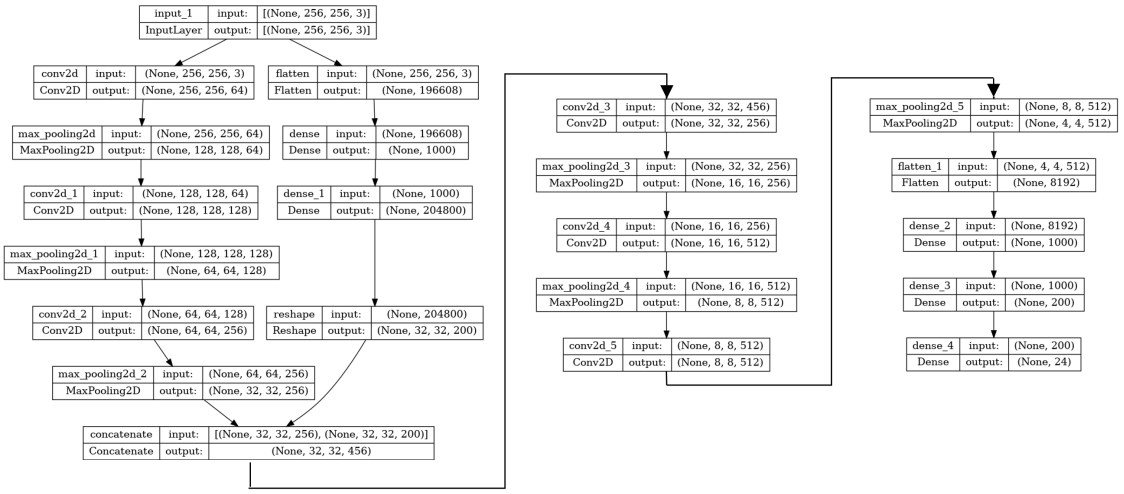


Figura 4.7: *Imagen representativa de la arquitectura propuesta para la sexta red.*

Este planteamiento diseñado dispara radicalmente el número de parámetros de la red, alcanzando más de trece millones, tal como se puede deducir al analizar la tabla 4.7, por lo que se necesitará una mejora drástica en el rendimiento de la red para optar por esta topología.

Bloque	**Tipo de Capa**	N° **de Filtros**	**Dimensión de Salida**	N° **de Parámetros**
input_1	InputLayer	-	(256 x 256 x 3)	0
conv2d	Conv2D	64	(256 x 256 x 64)	1792
max_pooling2d	MaxPooling2D	-	(128 x 128 x 64)	0
conv2d_1	Conv2D	128	(128 x 128 x 128)	73856
flatten	Flatten	-	(196608)	0
max_pooling2d_1	MaxPooling2D	-	(64 x 64 x 128)	0
dense	Dense	-	(1000)	196609000
conv2d_2	Conv2D	256	(64 x 64 x 256)	295168
dense_1	Dense	-	(204800)	205004800
max_pooling2d_2	MaxPooling2D	-	(32 x 32 x 256)	0
reshape	Reshape	-	(32 x 32 x 200)	0
concatenate	Concatenate	-	(32 x 32 x 456)	0
conv2d_3	Conv2D	256	(32 x 32 x 256)	1050880
max_pooling2d_3	MaxPooling2D	-	(16 x 16 x 256)	0
conv2d_4	Conv2D	512	(16 x 16 x 512)	1180160
max_pooling2d_4	MaxPooling2D	-	(8 x 8 x 512)	0
conv2d_5	Conv2D	512	(8 x 8 x 512)	2359808
max_pooling2d_5	MaxPooling2D	-	(4 x 4 x 512)	0
flatten_1	Flatten	-	(8192)	0
dense_2	Dense	-	(1000)	8193000
dense_3	Dense	-	(200)	200200
FinalStage	Dense	-	(24)	4824

Tabla 4.7: Descripción detallada de las capas de la arquitectura propuesta para la sexta red.

4.1.7. Red 7: Arquitectura original con capas de *Batch Normalization*:

Hasta este punto, la mayoría de pruebas de red realizadas se han centrado en modificar y hacer más profunda y densa la red, provocando, en muchos casos, un aumento significativo en el número de parámetros y, por consiguiente, en el peso de la misma.Por tanto, en esta prueba el planteamiento será volver a la arquitectura original, pero esta vez empleando capas de Batch Normalization.

Como se ha analizado anteriormente, este tipo de capas acelera el entrenamiento y reduce el riesgo de sobreentrenamiento. Dado que la arquitectura original de la Polenet V.1. tiene una buena relación entre su precisión y su peso, se plantea la idea de solo añadir estas capas, intentando mejorar la precisión sin afectar significativamente al peso (ver Figura 4.8).

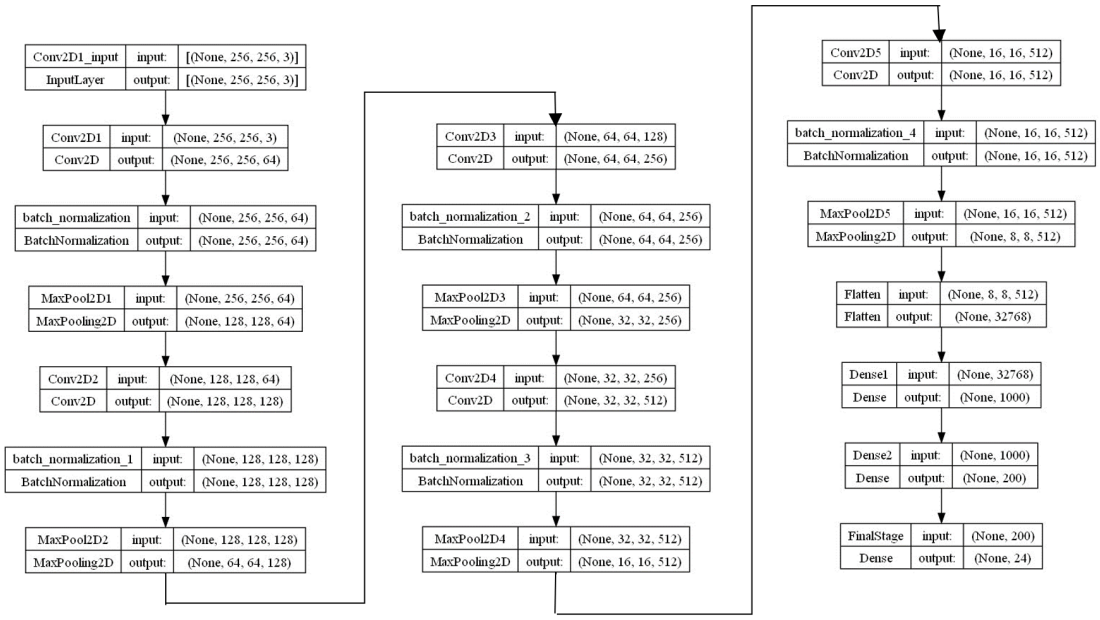


Figura 4.8: *Imagen representativa de la arquitectura propuesta para la séptima red.*

A continuación, la tabla de la arquitectura permitirá comprobar que se trata de la arquitectura original de la Polenet V.1. y también comprobar en cuánto ha crecido el número de parámetros de la red (ver Tabla 4.8).

Bloque	Tipo de Capa	N° de Filtros	Dimensión de Salida	N° de Parámetros
input_1	InputLayer	-	(256 x 256 x 3)	0
Conv2D1	Conv2D	64	(256 x 256 x 64)	1792
batch_normalization	BatchNormalization	-	(256 x 256 x 64)	256
MaxPool2D1	MaxPooling2D	-	(128 x 128 x 64)	0
Conv2D2	Conv2D	128	(128 x 128 x 128)	73856
batch_normalization_1	BatchNormalization	-	(128 x 128 x 128)	512
MaxPool2D2	MaxPooling2D	-	(64 x 64 x 128)	0
Conv2D3	Conv2D	256	(64 x 64 x 256)	295168
batch_normalization_2	BatchNormalization	-	(64 x 64 x 256)	1024
MaxPool2D3	MaxPooling2D	-	(32 x 32 x 256)	0
Conv2D4	Conv2D	512	(32 x 32 x 512)	1180160
batch_normalization_3	BatchNormalization	-	(32 x 32 x 512)	2048
MaxPool2D4	MaxPooling2D	-	(16 x 16 x 512)	0
Conv2D5	Conv2D	512	(16 x 16 x 512)	2359808
batch_normalization_4	BatchNormalization	-	(16 x 16 x 512)	2048
MaxPool2D5	MaxPooling2D	-	(8 x 8 x 512)	0
Flatten	Flatten	-	(32768)	0
Dense1	Dense	-	(1000)	32769000
Dense2	Dense	-	(200)	200200
FinalStage	Dense	-	(24)	4824

Tabla 4.8: Descripción detallada de las capas de la arquitectura propuesta para la séptima red.

4.1.8. Red 8: Arquitectura original, con capas de *Batch Normalization* y salida con *Global Average Pooling*:

Esta red busca aplicar la misma lógica que se ha descrito en el apartado anterior por lo que respecta al proceso de convolución. Sin embargo, al mismo tiempo, se intentará disminuir lo máximo posible el tamaño de la red. Para ello, se realizarán cambios en las capas de salida, ya que sigue siendo el punto que más parámetros acumula de toda la red. Por ello, se sustituirá la capa de *Flatten* por una de *Global Average Pooling*. Además, se pasará de dos capas densas a solo una. Finalmente, entre la capa densa y la capa de salida, se incluirá una capa de *Batch Normalization* para evitar el sobreentrenamiento durante el entrenamiento de estas capas (ver Figura 4.9).

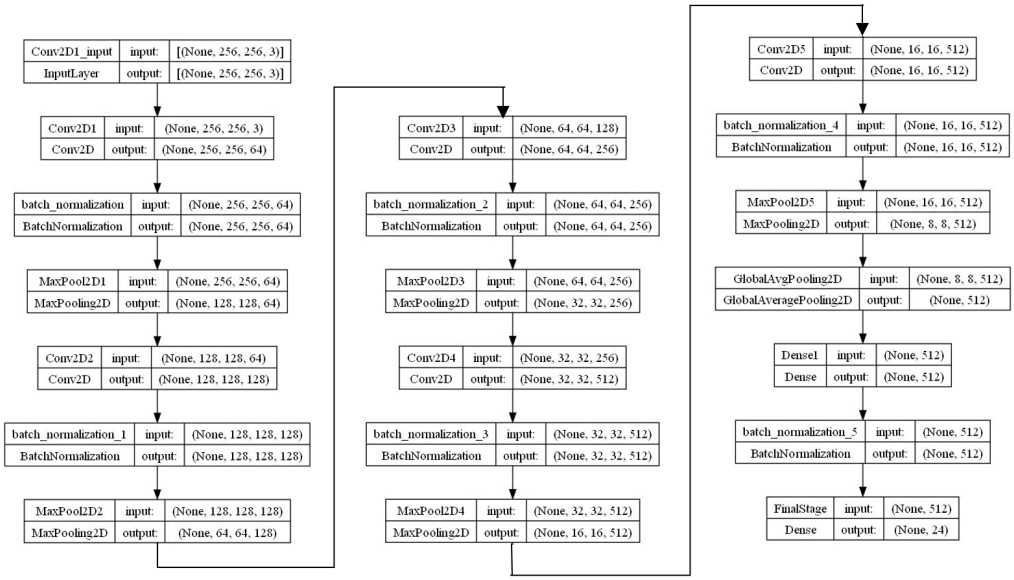


Figura 4.9: *Imagen representativa de la arquitectura propuesta para la octava red.*

Con estos cambios, esta red solo tiene 4 millones de parámetros entrenables, teniendo esta red casi tantos parámetros en total como tenía la arquitectura original solo en su capa de *Flatten* (ver Tabla 4.9).

Bloque	**Tipo de Capa**	**N° de Filtros**	**Dimensión de Salida**	**N° de Parámetros**
input_1	InputLayer	-	(256 x 256 x 3)	0
Conv2D1	Conv2D	64	(256 x 256 x 64)	1792
batch_normalization	BatchNormalization	-	(256 x 256 x 64)	256
MaxPool2D1	MaxPooling2D	-	(128 x 128 x 64)	0
Conv2D2	Conv2D	128	(128 x 128 x 128)	73856
batch_normalization_1	BatchNormalization	-	(128 x 128 x 128)	512
MaxPool2D2	MaxPooling2D	-	(64 x 64 x 128)	0
Conv2D3	Conv2D	256	(64 x 64 x 256)	295168
batch_normalization_2	BatchNormalization	-	(64 x 64 x 256)	1024
MaxPool2D3	MaxPooling2D	-	(32 x 32 x 256)	0
Conv2D4	Conv2D	512	(32 x 32 x 512)	1180160
batch_normalization_3	BatchNormalization	-	(32 x 32 x 512)	2048
MaxPool2D4	MaxPooling2D	-	(16 x 16 x 512)	0
Conv2D5	Conv2D	512	(16 x 16 x 512)	2359808
batch_normalization_4	BatchNormalization	-	(16 x 16 x 512)	2048
MaxPool2D5	MaxPooling2D	-	(8 x 8 x 512)	0
GlobalAvgPooling2D	Glob.Av.Pooling	-	(512)	0
Dense1	Dense	-	(512)	262656
batch_norm	BatchNorm.	-	(512)	2048
FinalStage	Dense	-	(24)	12312

Tabla 4.9: Descripción detallada de las capas de la arquitectura propuesta para la octava red.

4.1.9. Red 9: Arquitectura original, con salida combinada de capas de *Global Average Pooling* y *Flatten*:

Esta arquitectura tiene un interés experimental, ya que no se ha encontrado ninguna arquitectura que haga uso de la combinación de estas dos capas. En esta configuración, se buscará comprobar los efectos de combinar una capa de *Global Average Pooling* y una capa *Flatten*. Desde un punto de vista teórico, el uso de una capa de *Global Average Pooling*

debería disminuir el número de parámetros de la red y el uso de una capa *Flatten* permitiría una mejor propagación de los datos. Con esta prueba, se pretende analizar cómo afecta esto al rendimiento de la red. Únicamente se realizará este cambio en comparación con la arquitectura original para garantizar una correcta comparación (ver Figura 4.10).

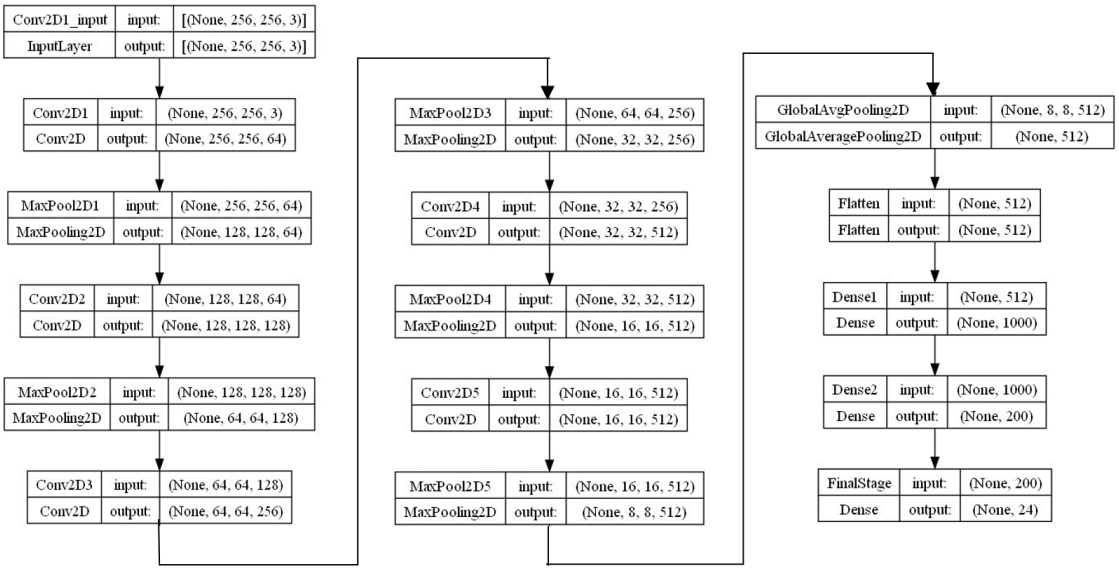


Figura 4.10: *Imagen representativa de la arquitectura propuesta para la novena Red.*

Con el uso de la capa de *Global Average Pooling*, se ha reducido el número de parámetros de los 7,392,708 originales a solo 4,628,808, quedando la topología como se muestra en la tabla 4.10

Bloque	**Tipo de Capa**	N° **de Filtros**	**Dimensión de Salida**	N° **de Parámetros**
input_1	InputLayer	-	(256 x 256 x 3)	0
Conv2D1	Conv2D	64	(256 x 256 x 64)	1792
MaxPool2D1	MaxPooling2D	-	(128 x 128 x 64)	0
Conv2D2	Conv2D	128	(128 x 128 x 128)	73856
MaxPool2D2	MaxPooling2D	-	(64 x 64 x 128)	0
Conv2D3	Conv2D	256	(64 x 64 x 256)	295168
MaxPool2D3	MaxPooling2D	-	(32 x 32 x 256)	0
Conv2D4	Conv2D	512	(32 x 32 x 512)	1180160
MaxPool2D4	MaxPooling2D	-	(16 x 16 x 512)	0
Conv2D5	Conv2D	512	(16 x 16 x 512)	2359808
MaxPool2D5	MaxPooling2D	-	(8 x 8 x 512)	0
GlobalAvgPooling2D	Glob.Av.Pooling	-	(512)	0
Flatten	Flatten	-	(512)	0
Dense1	Dense	-	(1000)	513000
Dense2	Dense	-	(200)	200200
FinalStage	Dense	-	(24)	4824

Tabla 4.10: Descripción detallada de las capas de la arquitectura propuesta para la novena Red.

4.2 Criterios de comparación del rendimiento de las redes

Una vez ya descritas todas las arquitecturas propuestas para la comparación, es importante analizar los criterios matemáticos que definirán el rendimiento de la red. Estos criterios serán cinco y se corresponden con los proporcionados por defecto por la herra-

mienta Keras para analizar el funcionamiento de las distintas redes ante unas muestras específicas ([44], [45], [46], [47], [48],):

- **Exactitud (accuracy):** Este parámetro mide la proporción de aciertos en la predicción de la red frente al total de muestras sobre las que se predice. Se calcula mediante la siguiente fórmula:

$$accuracy = \frac{Verdaderos\ positivos + Verdadesros\ negativos}{N^{\circ} demuestras} \tag{4.1}$$

- **Precisión (precision):** Este parámetro define la capacidad de una red para proporcionar aciertos de un tipo concreto frente al total de aciertos realizados. La fórmula para calcular la precisión de cada tipo es:

$$precision = \frac{Verdaderos\ positivos}{Verdaderos\ positivos + Falsos\ positivos} \tag{4.2}$$

  A diferencia del parámetro anterior, para calcular la precisión de la red, se calcula la precisión particular de cada tipo entre aquellos que la red puede predecir. Luego, se calcula una media ponderada (weighted average), donde el valor de precisión de cada tipo tiene un peso que depende del número de muestras asociadas a ese tipo. La fórmula se muestra a continuación:

$$precision\ red = \frac{\sum(precision\ tipo * numero\ muestras\ tipo)}{numero\ muestras\ totales} \tag{4.3}$$

- **Sensibilidad (recall):** Este parámetro es el opuesto a la precisión, ya que nos proporciona información sobre el rendimiento de un clasificador con respecto a falsos negativos en lugar de los falsos positivos, como lo hace la precisión. La fórmula para calcularlo es muy similar a la de la precisión:

$$precision = \frac{Verdaderos\ positivos}{Verdaderos\ positivos + Falsos\ negativos} \tag{4.4}$$

  Con este parámetro, al igual que con la precisión, se calcula para cada tipo, y se define la sensibilidad total de la red como la media ponderada de estos.

- **Puntuación-f1(f1-score):** El f1-score es un parámetro que evalúa la capacidad de una red para equilibrar la relación entre la precisión y la sensibilidad. Se calcula mediante la siguiente fórmula:

$$F1 = \frac{2 * precision * recall}{precision + recall} \tag{4.5}$$

  Para calcular el global de la red, se puede usar los parámetros de precision y recall de cada clase y luego ponderarlos, o utilizar directamente los valores ponderados de la red.

- **Coeficiente de correlación de Matthews (MCC):** Este valor es una métrica que contabiliza la relación entre una clase y el resto de clases en la matriz. Es más relevante cuando existe una diferencia significativa entre las predicciones negativas y positivas, ya que un valor muy bajo indica que la red no está prediciendo de manera significativa, sino asignando valores de manera aleatoria. La fórmula para calcular esto está descrita por la siguiente expresión:

$$MCC = \frac{VP * VN - FP * FN}{\sqrt{(VP + FP)(VP + FN)(VN + FP)(VN + FN)}} \tag{4.6}$$

Siendo VP (Verdaderos positivos), VN (Verdaderos negativos), FP (Falsos positivos) y FN (Falsos negativos), como ya se ha descrito anteriormente, ahora es necesario ponderar el resultado para cada tipo y obtener el total de la red.

En este caso, también se puede aplicar una generalización para multiclase, llamado el estadístico R_k , que obtiene directamente el valor total de la red [53] :

$$\text{MCC} = \frac{\sum_k \sum_l \sum_m C_{kk} C_{lm} - C_{kl} C_{mk}}{\sqrt{\sum_k \left(\sum_l C_{kl}\right) \left(\sum_{k' \neq k} \sum_{l'} C_{k'l'}\right)} \sqrt{\sum_k \left(\sum_l C_{lk}\right) \left(\sum_{k' \neq k} \sum_{l'} C_{l'k'}\right)}} \tag{4.7}$$

Siendo:

$s = \sum_i \sum_j C_{ij} \Rightarrow$ El número total de muestras.

$c = \sum_k C_{kk} \Rightarrow$ El número de muestras predichas correctamente por la red.

$p_k = \sum_i C_{ki} \Rightarrow$ El número de veces que la clase k a sido predicha por la red.

$t_k = \sum_i C_{ik} \Rightarrow$ El número de veces que la clase k a sido predicha correctamente.

De ahora en adelante, se usará también la nomenclatura en inglés para referirse a los términos matemáticos presentados para definir el rendimiento de la red.

Adicionalmente a estos parámetros matemáticos, se emplearán dos criterios más a la hora de seleccionar una red sobre otra: como son su **inferencia**, es decir, el tiempo que tarda en generar una predicción una vez te entrenada ,y su **peso en bytes**. Estos dos criterios son muy importantes para su posterior implementación en una herramienta informática.

4.3 Prueba para la selección de arquitectura mediante el *dataset* propio

Antes de comenzar, es necesario determinar los criterios que se seguirán a la hora de entrenar las redes con las arquitecturas propuestas en el apartado anterior, ya que deberán ser constantes en todas las pruebas para poder obtener redes comparables entre si.

Por lo que respecta a los hiperparámetros, se emplearán la siguiente valores:

- **Tamaño de la imagen de entrada:** 256 x 256
- **Batch Size:** 32
- **Epochs:** 350
- **Optimizador:** SGD
- **Learning rate:** 0,05
- **Métrica comparar para la optimización:** *accuracy*
- **Reesacalado de la imagen:** Entre valores de 0 a 1

Se emplearán, para las pruebas de selección, solo uno de los *5-fold* generados, en este caso el *output* 1, con la finalidad de agilizar el proceso, ya que en este punto del proyecto solo se persigue seleccionar la mejor arquitectura y no entrenar una red para usarse de manera definitiva.

Adicionalmente, se aplicará a las imágenes de entrada un proceso de *data augmentation* basado en el criterio descrito en el apartado 2.2.2., empleando las siguientes funciones del paquete keras:

- rotation_range=int(180 * 0.1)
- width_shift_range=0.1
- height_shift_range=0.1
- zoom_range=0.1
- horizontal_flip=True
- vertical_flip=True

Por el contrario, no se podrá aplicar la técnica de *Transfer Learning*, ya que es la primera vez que se entrenarán estas arquitecturas. Por eso, es necesario emplear un número tan elevado de épocas, ya que en ensayos anteriores se ha logrado una convergencia de la red empleando esta cantidad de épocas.

A continuación, se realizarán los entrenamientos de las distintas arquitecturas, usando un código con una estructura idéntica a la descrita en el apéndice B.2, pero aplicando en cada caso la topología de la red a entrenar. Las siguientes gráficas muestran el rendimiento de las diez arquitecturas descritas en los siete criterios comparativos presentados anteriormente (ver Figuras 4.11, 4.12 y 4.13) .

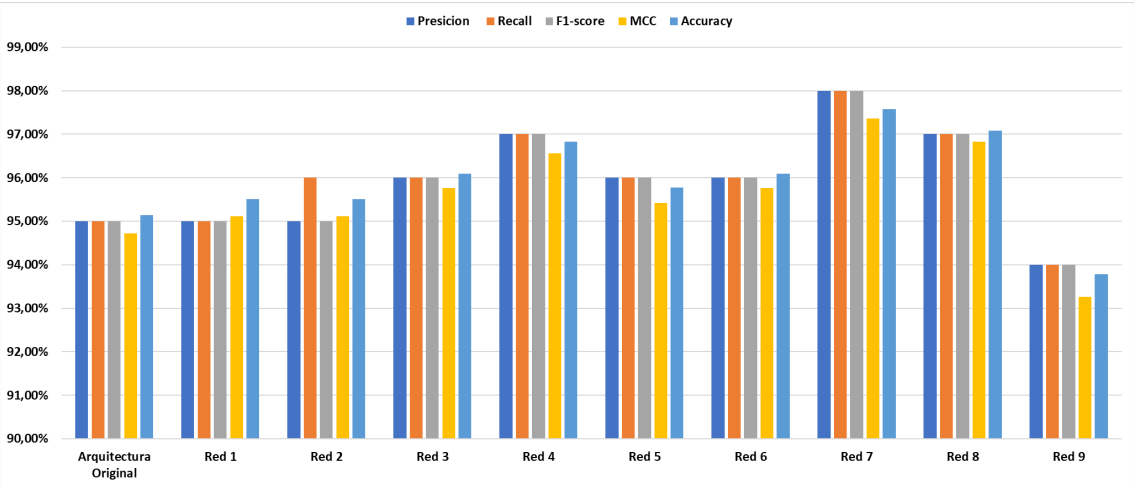


Figura 4.11: *Gráfica comparativa de los valores porcentuales de Precison, Recall, F1-score, MCC y Accuracy para el entrenamiento con el dataset propio de las arquitecturas presentadas en el aparatado 4.2.*

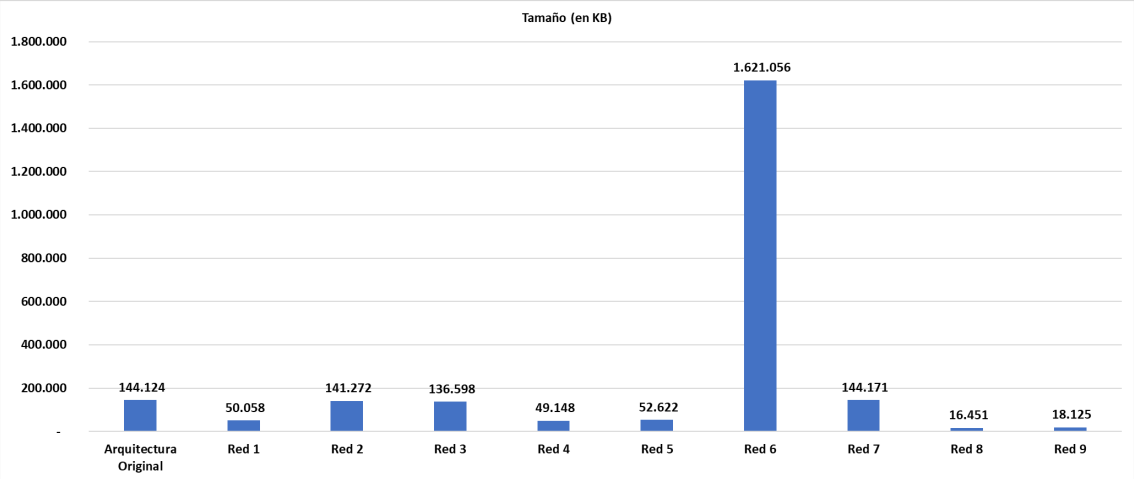


Figura 4.12: *Gráfica comparativa del peso en Kilobyte de las arquitecturas presentadas en el aparatado 4.2.*

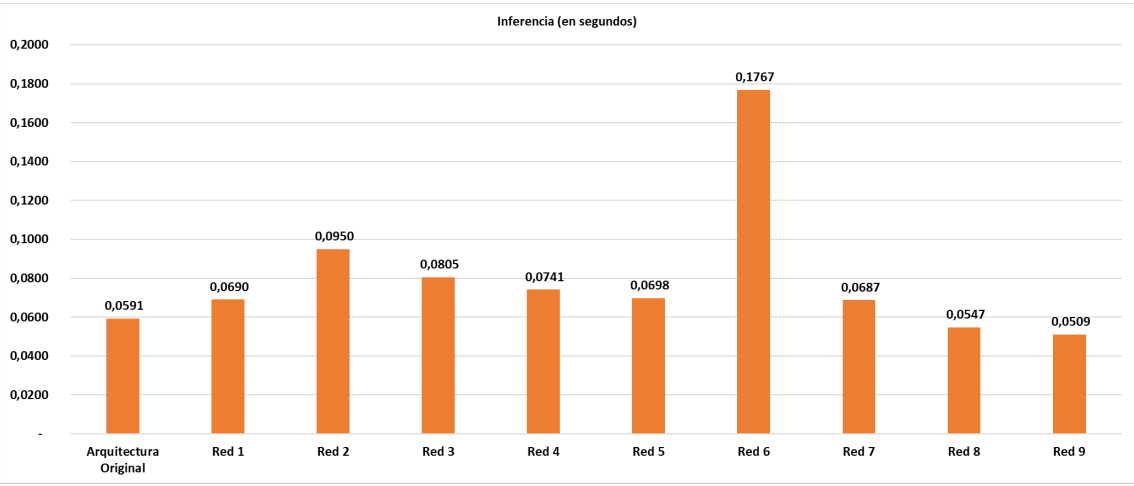


Figura 4.13: *Gráfica comparativa del tiempo de inferencia en realizar la perdición del tipo de una imagen en segundos de las arquitecturas presentadas en el aparatado 4.2.*

A la vista de los resultados, se pueden concluir diferentes hechos. En lo que respecta a la exactitud en las perdiciones de las redes, todas las arquitecturas propuestas menos la 9 mejoran el rendimiento de la arquitectura original. Destacando la arquitectura 7 como la mejor, con una exactitud del 97, 57 % . Otras arquitecturas, como la 4 o la 8, también presentan una mejora significativa en la exactitud de 96,83 % y 97,08 % respectivamente.

Por lo que respecta al tamaño en KB, salvo en una de las arquitecturas, se ha disminuido el tamaño de la red. Destacando, entre todas, la arquitectura de la red 8 como la menos pesada de todas, con un tamaño de 16451 KB.

Por último, el tiempo de inferencia no ha resultado un criterio muy significativo para realizar un cribado entre las destinas redes, ya que todas, excepto la arquitectura 6, tienen un tiempo de inferencia más que razonable a la hora de ser implementadas para una aplicación como la descrita en este proyecto.

Teniendo en cuenta los objetivos del proyecto, existen tres arquitecturas que destacan sobre el resto: la arquitectura 7, la cual presenta la mejor exactitud entre todas, pero no disminuye significativamente su tamaño frente a la arquitectura original, y las arquitectu-

ras 4 y 8, que sí disminuyen su tamaño significativamente pero a costa de un rendimiento ligeramente inferior. Para facilitar el proceso de selección, se presentarán todas las arquitecturas en una gráfica que relacione su exactitud con su tamaño (ver Figura 4.14).

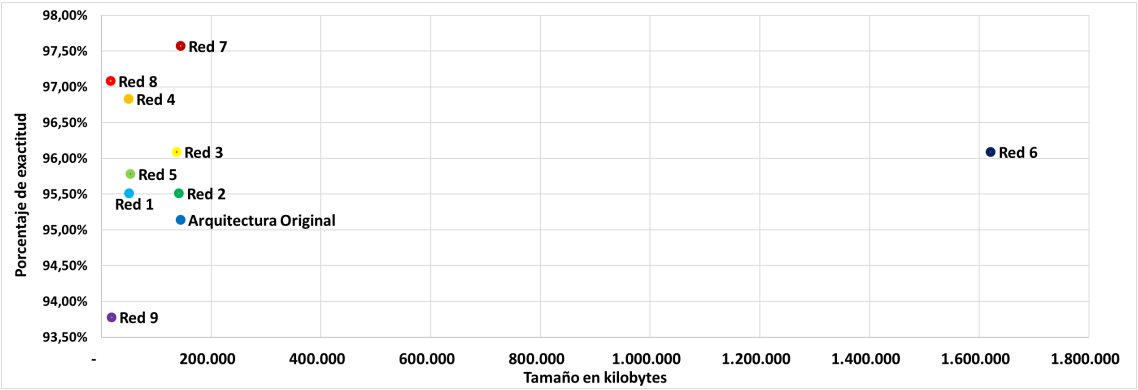


Figura 4.14: *Gráfica comparativa que relaciona la exactitud y el tamaño de las arquitecturas presentadas en el aparatado 4.2.*

Con toda la información presentada y tras analizar la figura 4.14, se puede apreciar que existen dos arquitecturas superiores al resto: la arquitectura 7, que destaca por su exactitud y la arquitectura 8, que destaca por su reducido tamaño. Dado que estos dos enfoques son los puntos de mayor interés del proyecto se ha decidido continuar no solo con una red como la nueva arquitectura para la Polenet, sino eligiendo ambas. De ahora en adelante, la arquitectura 7 será nombrada como Polenet V.2 y la arquitectura 8 como Polenet V.2 mobile.

CAPÍTULO 5
Comparación de los red propuesta con otras arquitecturas estándar

Una vez seleccionadas las mejores arquitecturas entre todas las descritas en el capitulo 4, es importante contextualizar estas dos nuevas arquitecturas dentro del marco de las CNN más relevantes. Las redes empleadas para la comparación son las descritas en apartado de metodología de este proyecto.

Antes de comenzar a realizar las pruebas, es importante definir, como se hizo en el apartado 4.3, los parámetros que se seguirán a la hora de llevar a cabo las pruebas.

En lo que respecta a los hiperparámetros de las pruebas, algunos permanecerán constantes durante todos los entrenamientos, independientemente del *dataset* o la arquitectura entrenada, como son:

- **Batch Size:** 32
- **Optimizador:** SGD
- **Learning rate:** 0,05
- **Métrica comparar para la optimización:** *accuracy*

Por lo que respecta al *data augmentation*, se aplicarán las siguientes funciones, las cuales realizan transformaciones más extensas a las imágenes que las vistas en el apartado 4.3. Esto se hace con el objetivo principal de mejorar el rendimiento de las redes.

- rotation_range=90
- width_shift_range=0.15
- height_shift_range=0.15
- zoom_range=0.15
- horizontal_flip=True
- vertical_flip=True
- brightness_range=[0.5, 1.5]

A continuación, se mostrarán dos gráficas de *'loss vs accuracy'* de las redes InceptionV3 y Polenet V.1, para demostrar la convergencia de estas redes al ser entrenadas con este número de épocas. (ver Figuras 5.1 y 5.1).

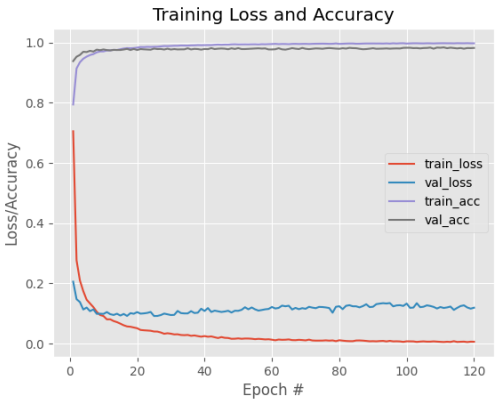


Figura 5.1: *Gráfica de 'loss vs accuracy' de ejemplo para comprobar la convergencia de la red InceptionV3 al ser entrenada con solo 120 épocas.*

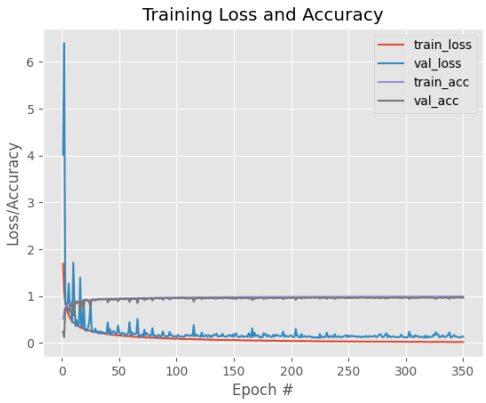


Figura 5.2: *Gráfica de 'loss vs accuracy' de ejemplo para comprobar la convergencia de la red Polenet V.1 al ser entrenada con solo 350 épocas.*

El resto de hiperparámetros, como el tamaño de la imagen de entrada, la aplicación o no de preproceso y la aplicación o no de reescalado, serán diferentes para cada red, siguiendo las instrucciones descritas en su respectiva documentación dentro del paquete Keras [51].

5.1 Comparativa con el *dataset* propio

Como se mencionó en el apartado 3.2, el número de muestras del *dataset* desarrollado es lo suficientemente grande para poder aplicar la técnica de validación cruzada *5-fold*, descrita en el apartado 3.3. Por lo tanto, a la hora de presentar los resultados se llevará a cabo una tabla para cada división (ver Figuras 5.3, 5.4, 5.5, 5.6 y 5.7) y posteriormente se añadirá la figura 5.8, en la que se habrá obtenido el resultado promedio de los 5 entrenamientos.

Es importante mencionar que los parámetros de tamaño de la red entrenada e inferencia son constantes para cada arquitectura, independientemente de la división del *dataset* empleado para entrenarla. Por lo tanto, solo se presentará una gráfica de resultados para cada parámetro (ver Figuras 5.9 y 5.10).

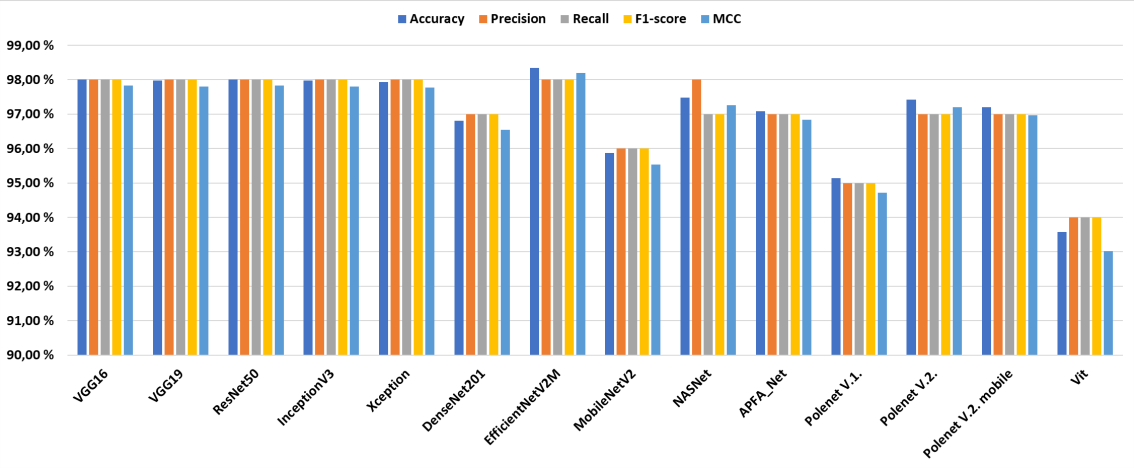


Figura 5.3: *Tabla comparativa de los resultados de Accuracy, Precision, Recall, F1-socre y MCC de las distintas arquitecturas entrenadas con la primera división del dataset.*

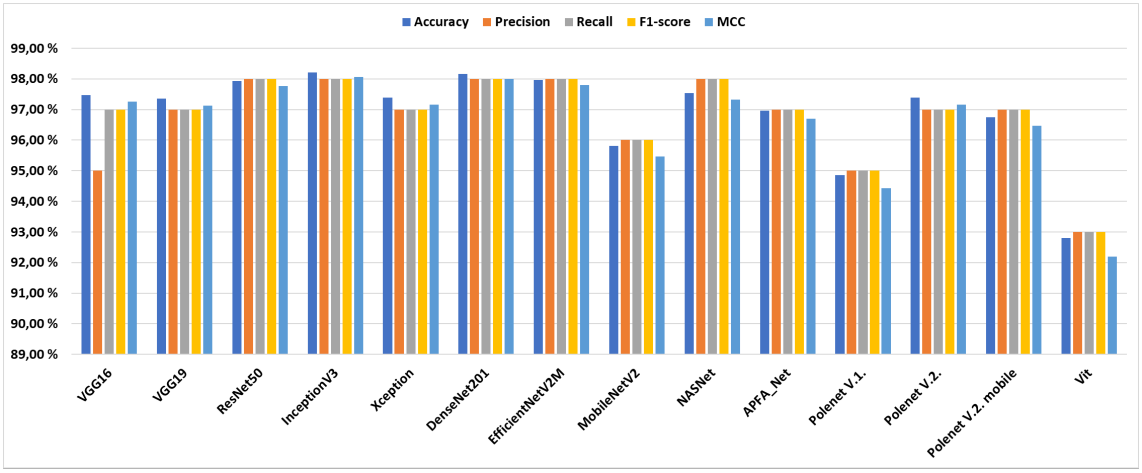


Figura 5.4: *Tabla comparativa de los resultados de Accuracy, Precision, Recall, F1-socre y MCC de las distintas arquitecturas entrenadas con la segunda división del dataset.*

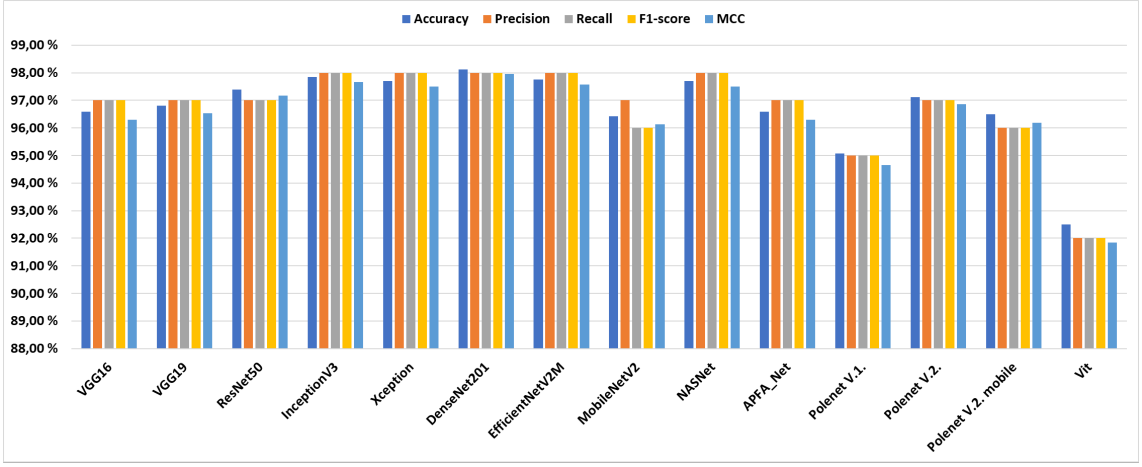


Figura 5.5: *Tabla comparativa de los resultados de Accuracy, Precision, Recall, F1-socre y MCC de las distintas arquitecturas entrenadas con la tercera división del dataset.*

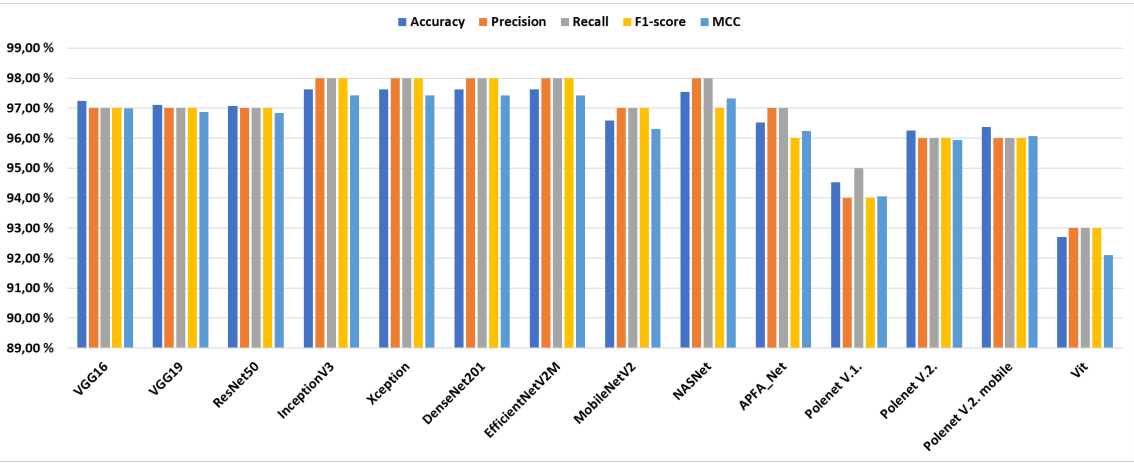


Figura 5.6: *Tabla comparativa de los resultados de Accuracy, Precision, Recall, F1-socre y MCC de las distintas arquitecturas entrenadas con la cuarta división del dataset.*

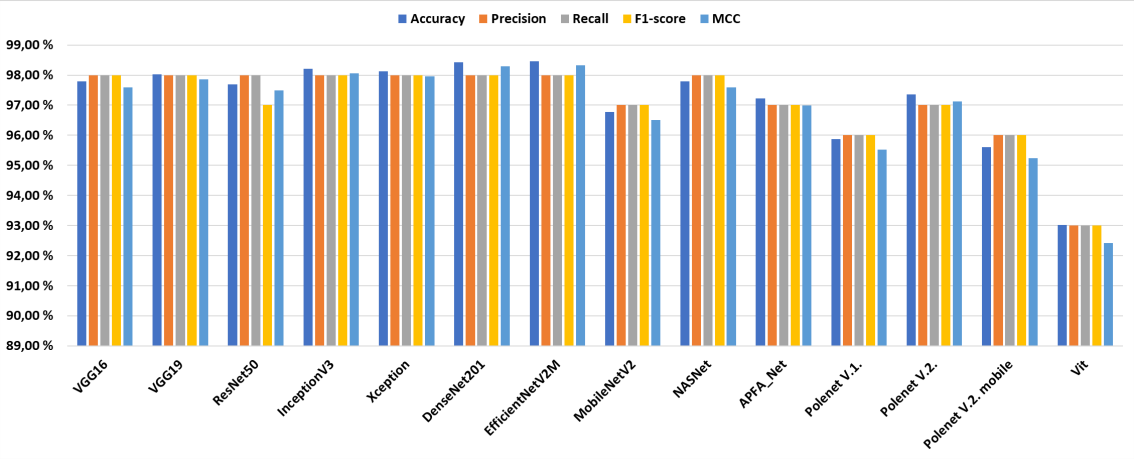


Figura 5.7: *Tabla comparativa de los resultados de Accuracy, Precision, Recall, F1-socre y MCC de las distintas arquitecturas entrenadas con la quinta división del dataset.*

Una vez presentados las cinco gráficas, se obtendrá la tabla 5.8, con los resultados promedio de las otras cinco. Estos valores serán los que se utilizarán para formular las conclusiones del experimento, ya que, como se ha podido observar, la distribución de las imágenes entre los conjuntos de test, entrenamiento y validación sí afecta al resultado general. De media, todas las redes entrenadas con el primer y el quinto *dataset* tiene un rendimiento casi del 1 % más que esas mismas redes entrenadas con los otros *dataset*.

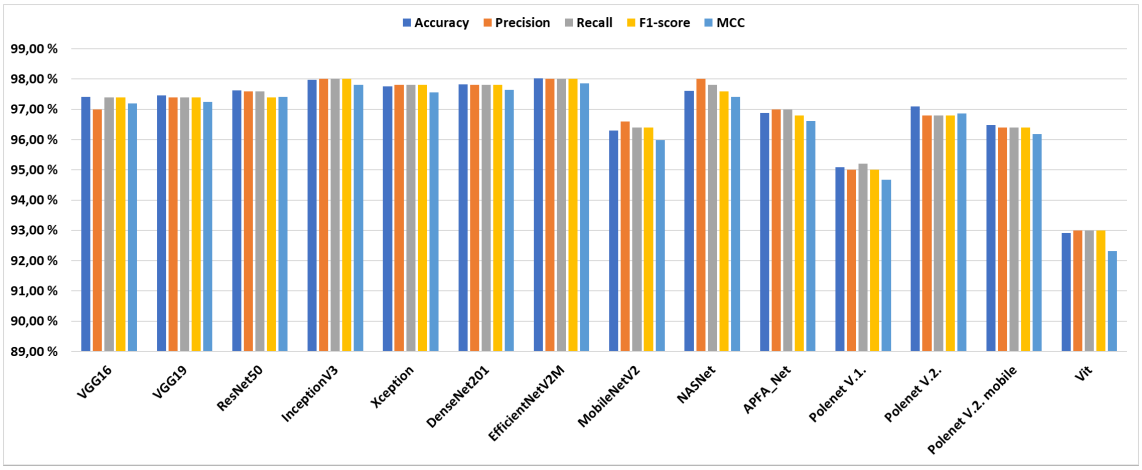


Figura 5.8: *Tabla comparativa de los resultados promedio de Accuracy, Precision, Recall, F1-socre y MCC de las distintas arquitecturas entrenadas con todas las divisiones del dataset.*

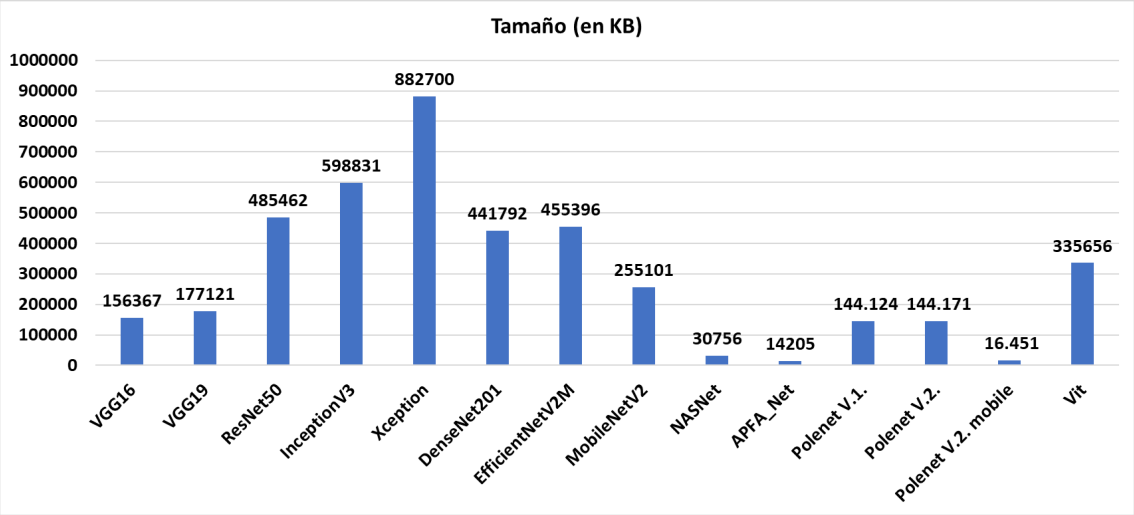


Figura 5.9: *Gráfica comparativa del peso en Kilobyte de todas las arquitecturas comparadas en este apartado*

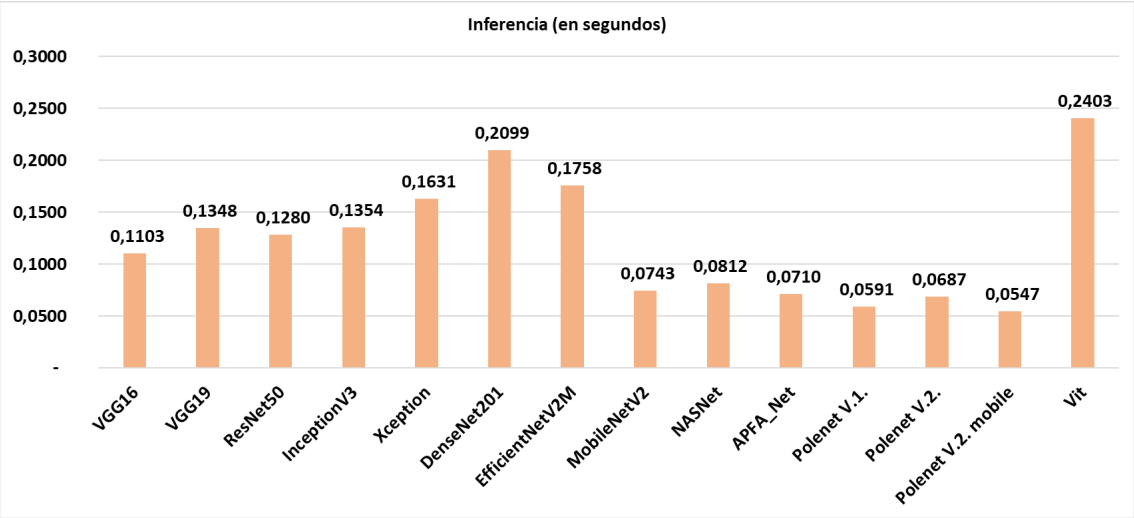


Figura 5.10: *Tabla comparativa de los resultados promedio de Accuracy, Precision, Recall, F1-socre y MCC de las distintas arquitecturas entrenadas con todas las divisiones del dataset.*

De manera similar a como se ha hecho en el apartado 4.3 con la figura 4.14, se mostrará una tabla comparativa ente el tamaño de la red y su exactitud, para que sirva de manera visual como forma rápida de comparación entre todas las arquitecturas según los parámetros más importantes a analizar para este proyecto (ver Figura 5.11)

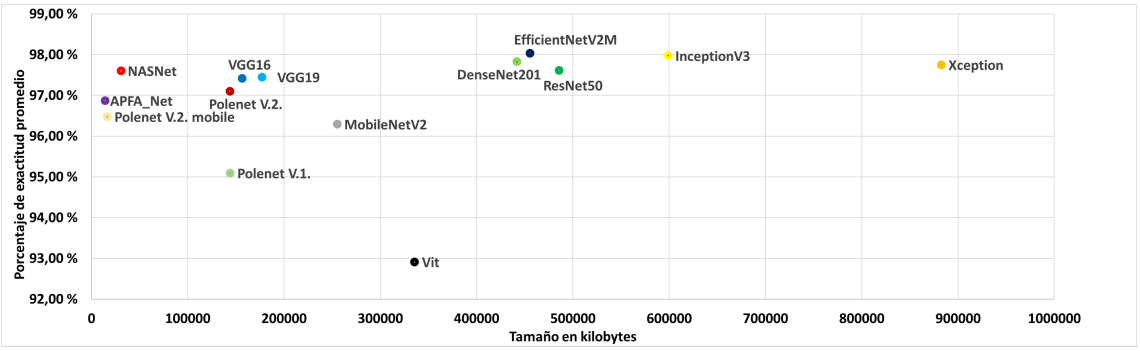


Figura 5.11: *Gráfica comparativa que relaciona la exactitud media y el tamaño, de las arquitecturas diseñadas y de la literatura, con el entrenamiento realizado usando el dataset propio.*

5.2 Comparativa con los *datasets* de la literatura

En este punto, para completar la información obtenida de las pruebas realizadas anteriormente, se repetirán los entrenamientos de las redes, pero empleando los *datasets* de variedades de polen más habituales en la literatura, siendo estos los ya descritos en el apartado 2.1.

Es importante mencionar que, debido a las bajas muestras por tipo de las que disponen de forma general estos *datasets*, no se empleará la técnica de *5-fold*; solo se dividirán las muestras entre test, validación y entrenamiento.

La forma de presentar los datos será con una figura de resultados por *dataset* (ver Figuras 5.12, 5.13, 5.14 y 5.15), la figura 5.17 que muestra el tiempo de inferencia y la figura 5.16, que muestra el peso de las redes entrenadas. Si bien es cierto que al variar el *dataset* el tamaño total de la red cambia, al tratarse de variaciones de menos de 100 KB en el mayor de los caso, se considerarán despreciables y se mostrará una única tabla con el tamaño medio de cada red.

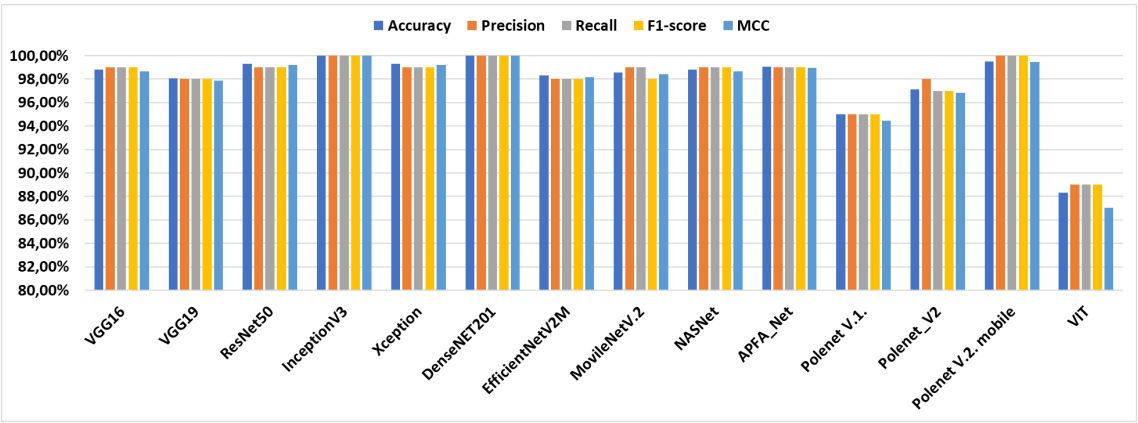


Figura 5.12: *Tabla comparativa de los resultados de Accuracy, Precision, Recall, F1-socre y MCC de las distintas arquitecturas entrenadas con el dataset Cretan Pollen.*

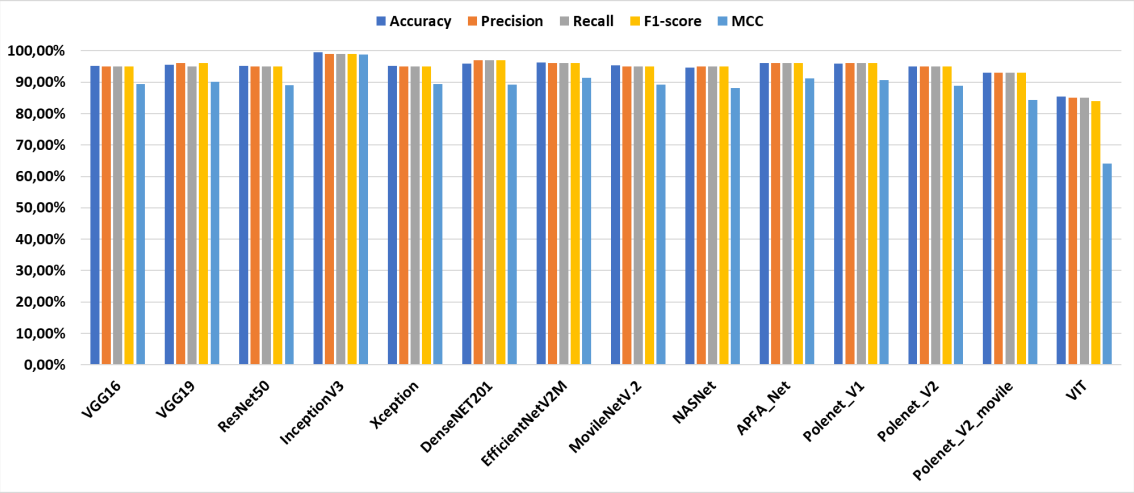


Figura 5.13: *Tabla comparativa de los resultados de Accuracy, Precision, Recall, F1-socre y MCC de las distintas arquitecturas entrenadas con el dataset POLLEN13K.*

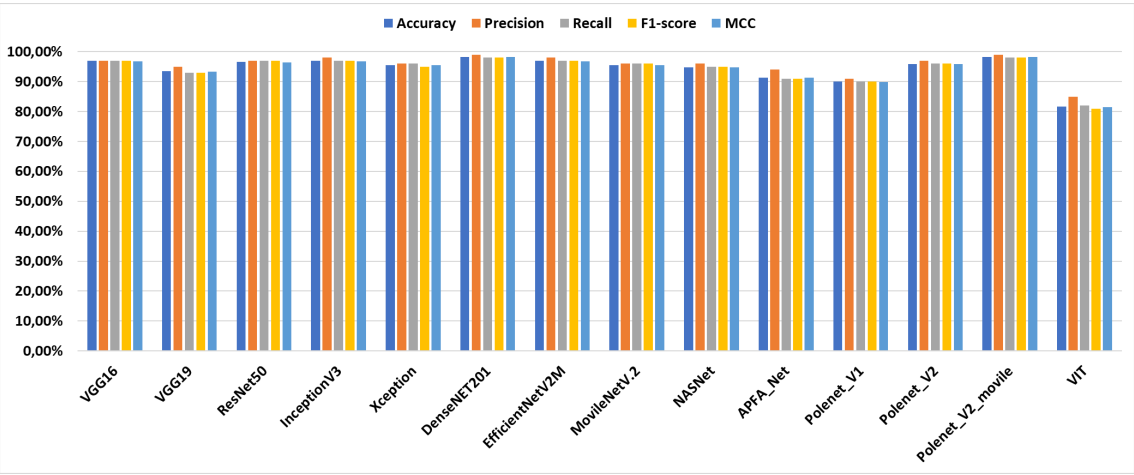


Figura 5.14: *Tabla comparativa de los resultados de Accuracy, Precision, Recall, F1-socre y MCC de las distintas arquitecturas entrenadas con el dataset POLLEN23E.*

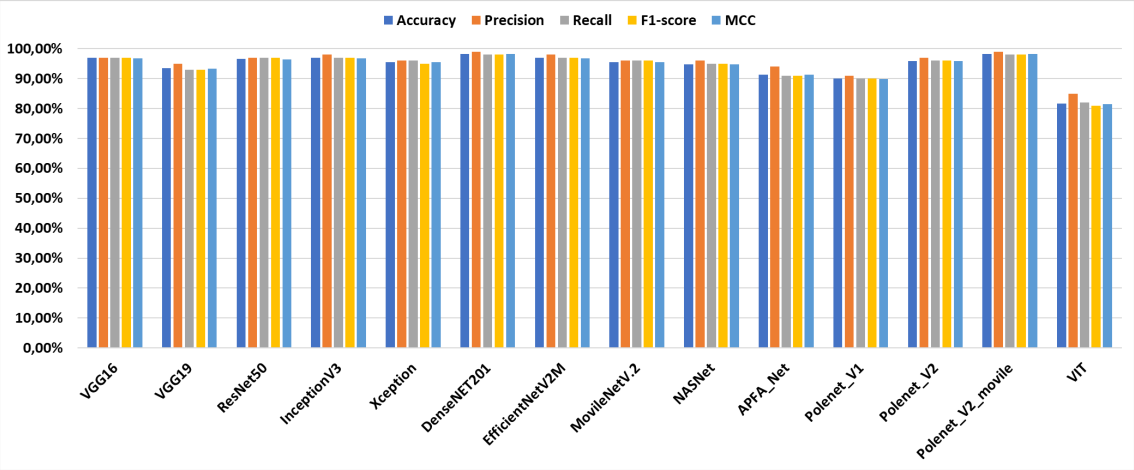


Figura 5.15: *Tabla comparativa de los resultados de Accuracy, Precision, Recall, F1-socre y MCC de las distintas arquitecturas entrenadas con el dataset POLLEN73S.*

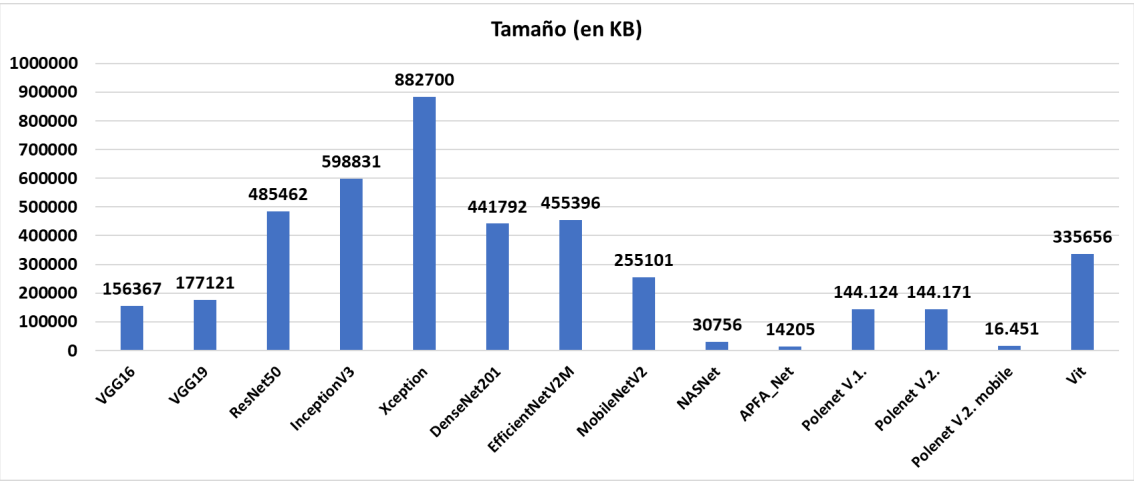


Figura 5.16: *Gráfica comparativa del peso promedio, en Kilobyte, de todas las arquitecturas entrenadas con cada dataset analizado en este apartado.*

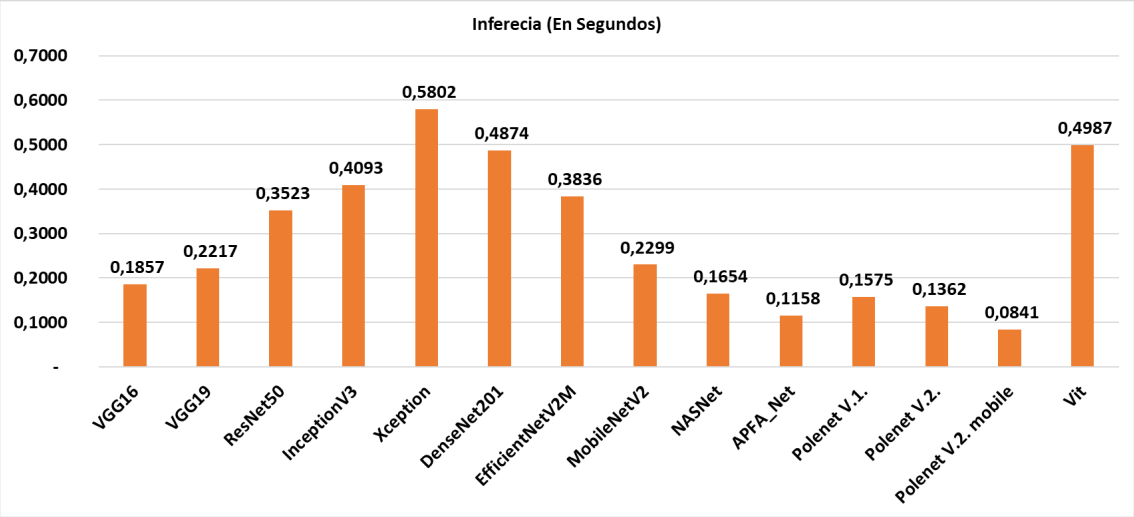


Figura 5.17: *Gráfica comparativa del tiempo de inferencia en realizar la perdición del tipo de una imagen de todas las arquitecturas entrenadas con cada dataset analizado en este apartado.*

Con los resultados obtenidos, se generarán nuevamente cuatro gráficas comparativas, una para cada *dataset*, las cuales relacionarán los valores de tamaño y exactitud de cada arquitectura (ver Figuras 5.18, 5.19, 5.20 y 5.21).

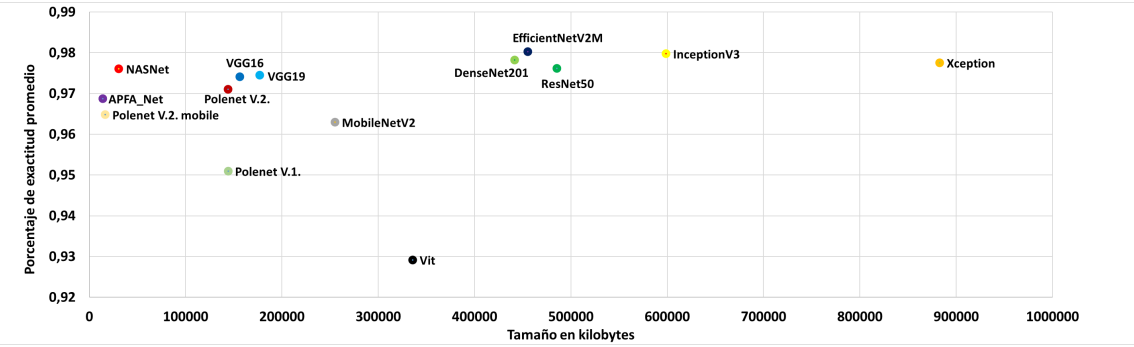


Figura 5.18: *Gráfica comparativa que relaciona la exactitud media y el tamaño, de las arquitecturas diseñadas y de la literatura, con el entrenamiento realizado usando el dataset Cretan Pollen.*

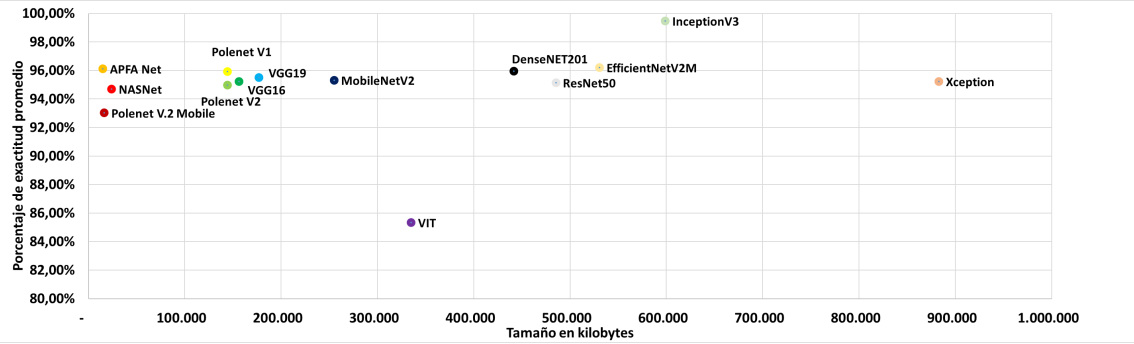


Figura 5.19: *Gráfica comparativa que relaciona la exactitud media y el tamaño, de las arquitecturas diseñadas y de la literatura, con el entrenamiento realizado usando el dataset POLLEN13K.*

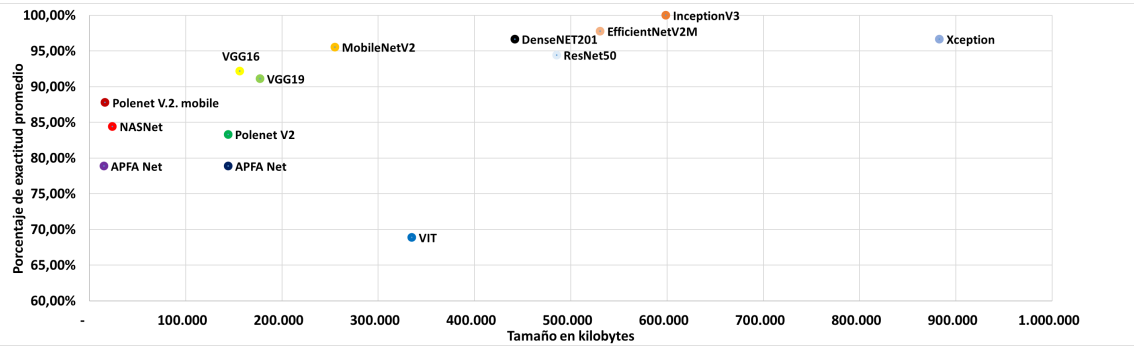


Figura 5.20: *Gráfica comparativa que relaciona la exactitud media y el tamaño, de las arquitecturas diseñadas y de la literatura, con el entrenamiento realizado usando el dataset POLLEN23E.*

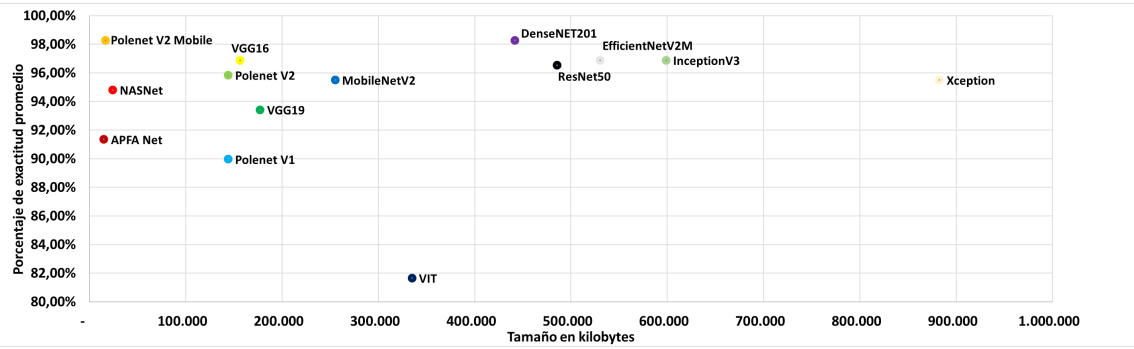


Figura 5.21: *Gráfica comparativa que relaciona la exactitud media y el tamaño, de las arquitecturas diseñadas y de la literatura, con el entrenamiento realizado usando el dataset POLLEN73S.*

5.3 Análisis de los resultados

A la vista de todos los resultados obtenidos en este apartado, se puede apreciar cómo las nuevas arquitecturas desarrolladas en el proyecto son capaces de superar en rendimiento a muchas de las arquitecturas más comunes dentro del sector, destacando sobre todo en el reducido tamaño de estas.

Las arquitecturas con mejor rendimiento, como la EfficeinetV2M o la InceptionV3, están a algo más de un 1 % de las dos arquitecturas desarrolladas en este trabajo. Sin embargo, estas últimas presentan un tamaño del orden de cinco a diez veces mayor, en comparación con la POLENET V.2 y la POLENET V.2 Mobile.

Cabe destacar, en ambas arquitecturas superan a la arquitectura original, la POLENET V.1, tanto en rendimiento como en menor tamaño.

Lo más destacable de estas pruebas es la inconsistencia en los resultados, al utilizar los *datasets* de la literatura. Como se puede apreciar en las distintas gráficas, estos *datasets* no generan redes con un comportamiento homogéneo, afectando drásticamente al rendimiento de una misma arquitectura al ser entrenado con un *dataset* frente a otro.

Al analizar el profundidad estos *dataset*, se han confirmado algunos problemas que ya han sido mencionados en el apartado 2.1. La mayoría de estos *datasets*, como POLLEN23E, POLLEN73S y Cretan Pollen, no cuentan con muestras suficientes para hacer un correcto entrenamiento. Como se puede ver en sus gráficas resultantes del entrenamiento (ver Figuras 5.22 y 5.23), algunas de las redes que han obtenido mejores resultados, como la Efficienent o la Inception, están sufriendo una tendencia al sobreentrenamiento. Esto permite afirmar que su buen rendimiento con estos *datasets* está causado por el hecho de ser redes preentrenadas y no realmente por la calidad de su arquitectura.

Figura 5.22: *Gráfica de 'loss vs accuracy' resultante del entrenamiento de la red EfficientNetV2M con el dataset POLLEN13K.*

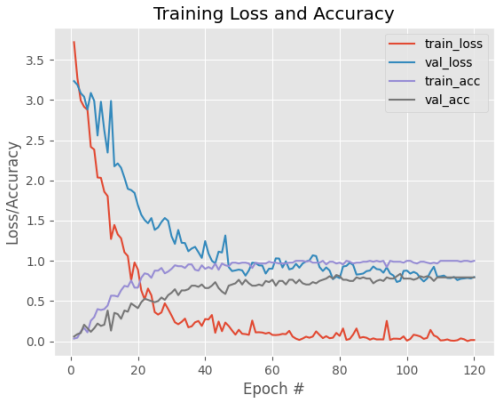


Figura 5.23: *Gráfica de 'loss vs accuracy' resultante del entrenamiento de la red InceptionV3 con el dataset POLLEN23E.*

Esto no solo justifica la inconsistencia de los resultados obtenidos en el apartado 5.2, sino que también pone en valor la calidad del *dataset* desarrollado en este proyecto, especialmente para el entrenamiento redes que buscan posteriormente ser implementadas en proyectos de este ámbito.

CAPÍTULO 6

Ajuste de hipeparámetros mediante la biblioteca "Optuna"

La elección de unos buenos valores de hipeparámetros es uno de los criterios más importantes a la hora de entrenar una red neuronal convolucional. Como se ha visto en los apartados 4.3 y 5.0, a la hora de realizar los entrenamientos, se han elegido unos valores arbitrarios basados en la experiencia personal y en ejemplos presentes en la literatura.

Con estos antecedentes, este apartado busca desarrollar un criterio más empírico para determinar el valor más óptimo para los hiperparámetros. Para ello, se emplea la biblioteca Optuna [52], que realiza un proceso de ajuste de hiperparámetros mediante iteraciones.

El proceso consiste en hacer entrenamientos parciales de las redes, modificando el valor de los hipeparámetros, permitiendo así que los valores varíen en función de los entrenamientos con mejor resultado.

El planteamiento de estas pruebas consistirá en entrenar las redes Polenet V.2 y Polenet V.2 Mobile con esta biblioteca, para lograr teóricamente obtener un mejor funcionamiento de la red sin afectara a su arquitectura. Los hipeparámetros a ajustar serán, por un lado, el *learning rate* y el *dropout*. Los criterios para realizar las pureabas serán 50 entrenamientos parciales de 150 épocas y un entrenamiento final con los mejores parámetros obtenidos de 350 épocas. El código para lograr esto está descrito en el apéndice B.3 del proyecto.

Para presentar los resultados, se utilizarán dos tipos de gráficas. Por un lado, las gráficas comparativas de los resultados ya empleadas anteriormente, una para cada arquitectura (ver Figuras 6.1 y 6.2), y , por otro lado, se mostrarán las gráficas que representarán la relación entre los valores de los hiperparámetros finales obtenidos y la exactitud de la red completamente entrenada (ver Figuras 6.3, 6.4, 6.5 y 6.6).

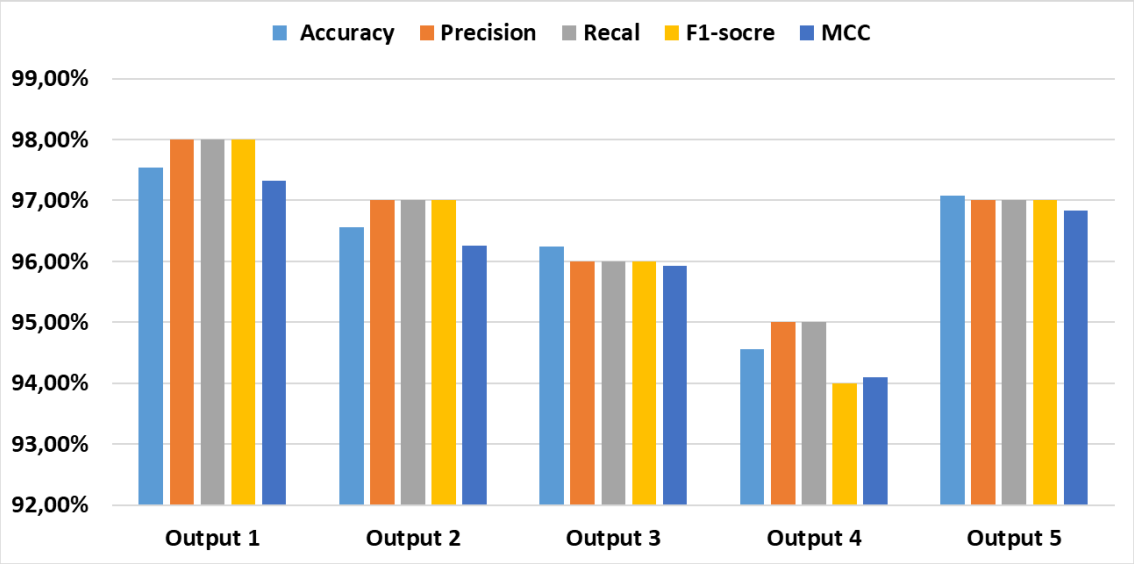


Figura 6.1: *Tabla comparativa de los resultados de Accuracy, Precision, Recall, F1-socre y MCC de las arquitectura POLENET V.2, aplicando la biblioteca Optuna.*

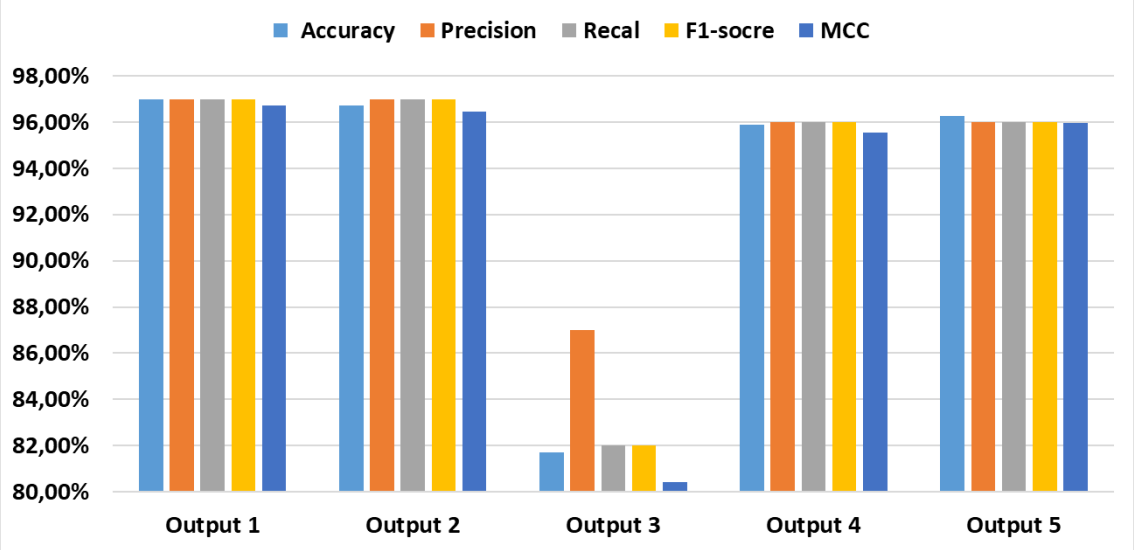


Figura 6.2: *Tabla comparativa de los resultados de Accuracy, Precision, Recall, F1-socre y MCC de las arquitectura POLENET V.2 Mobile, aplicando la biblioteca Optuna.*

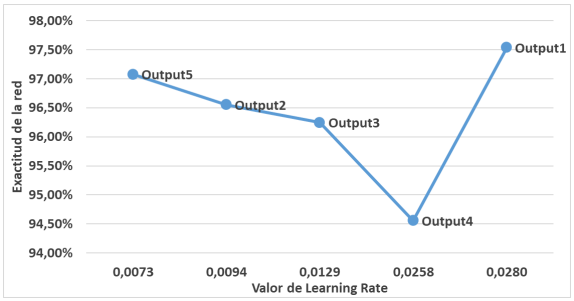


Figura 6.3: *Gráfica comparativa de los valores de "learning rate" y exactitud, para la arquitectura POLENET V.2, aplicando la biblioteca Optuna.*

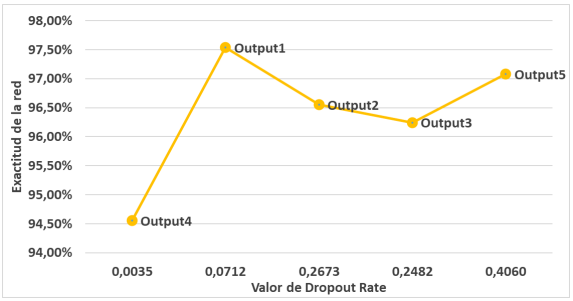


Figura 6.4: *Gráfica comparativa de los valores de "dropout" y exactitud, para la arquitectura POLENET V.2, aplicando la biblioteca Optuna.*

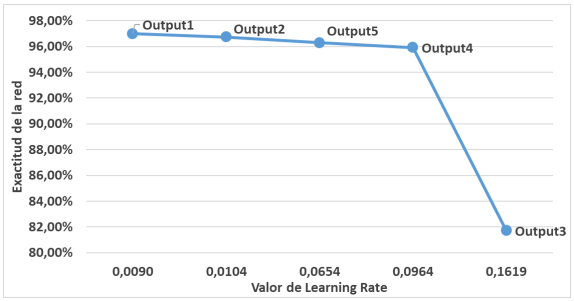


Figura 6.5: *Gráfica comparativa de los valores de "learning rate" y exactitud, para la arquitectura POLENET V.2 Mobile, aplicando la biblioteca Optuna.*

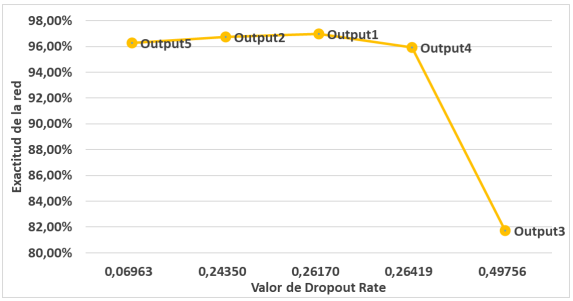


Figura 6.6: *Gráfica comparativa de los valores de "dropout" y exactitud, para la arquitectura POLENET V.2 Mobile, aplicando la biblioteca Optuna.*

6.1 Análisis de los resultados

Los resultados de estas pruebas no han conducido a una conclusión esclarecedora. En todos los casos, las diez redes entrenadas han mostrado un rendimiento peor que sus contrapartes entrenadas con unos hiperparámetros estandarizados.

Además, no se ha podido obtener ningún resultado concluyente que relacione un mayor o menor valor de *learning rate* y *dropout* con el rendimiento general de la red. Por el contrario, se observa más claramente cómo la división de los datos del *dataset* influyen más en la exactitud de la red que la variación de los valores de los hiperparámetros.

Por lo tanto, debido al tiempo excesivo que conlleva realizar un entrenamiento implementando el sistema de Optuna y el bajo rendimiento que se ha obtenido de las pruebas realizadas, se opta por dejar este planteamiento de lado y optar por otras estrategias.

CAPÍTULO 7
Aplicación de la técnica de agrupación de redes neuronales

Las técnica de agrupación de redes neuronales (*ensamble*), es un planteamiento dentro de las CNN, que busca mejorar el rendimiento de un sistema de clasificación, mediante la suma de varias redes convolucionales.

Esta técnica se basa en disponer de varias redes ya entrenadas, las cuales generarán conjuntamente una predicción de una imagen de manera simultánea y se definirá el tipo de la imagen combinándolas. En este punto, existen muchos criterios para determinar cómo relacionar los predicciones, pero para este proyecto se estudiarán dos de los planteamientos más sencillos: por mayor porcentaje de exactitud y por votación.

Cabe mencionar que, para estas pruebas, se emplearan grupos de tres redes, ya que de lo contrario, si se usaran un mayor número, el tamaño del sistema en conjunto sería demasiado grande. Además, siempre se usarán las redes entrenadas con el mismo *output* del *dataset* para que, a la hora de utilizar las imágenes de test, se pueda asegurar que no se han empleado para el entrenamiento, ya que esto provocaría un resultado alterado.

La elección de los grupos de redes para realizar las pruebas responderán a los siguientes criterios: se realizará una prueba con las tres redes con mayor exactitud, luego se probarán las dos redes con más exactitud y las dos redes desarrolladas en este proyecto, y por último, se probarán con las tres redes más pequeñas.

Por lo que refiere a los resultados, ya que no distan mucho entre los distintos *datasets*, se mostrarán los resultados como un promedio de las cinco pruebas.

De ahora en adelante, en este trabajo, con el objetivo de emplear la terminología más habitual dentro de este campo, se utilizará el termino ingles *ensamble* para referirse al término agrupación de redes neuronales.

7.1 Resultados mediante planteamiento de mayor porcentaje

Este planteamiento se basa en, una vez generadas las predicciones por todas las redes, tomar como la predicción definitiva de la imagen aquella que cuente con el mayor porcentaje de exactitud. En el apéndice B.4 se muestra el código empleado para lograr esto.

Para comprobar el resultado de este planteamiento, se mostrará en cada figura relativa a un *ensamble* el número de aciertos del conjunto, el número de fallos y, para este caso, el número de veces que se ha elegido cada predicción de la red, como la predicción del conjunto (ver Figuras 7.1, 7.2, 7.3 y 7.4).

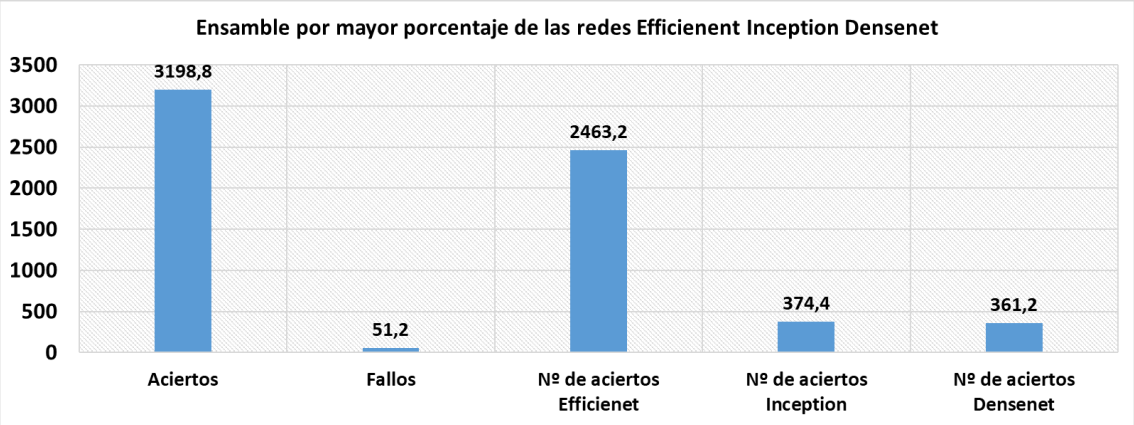


Figura 7.1: *Gráfica de resultados promedio al aplicar la técnica de ensamble de mayor porcentaje con las redes Efficienent, Inception y Densenet.*

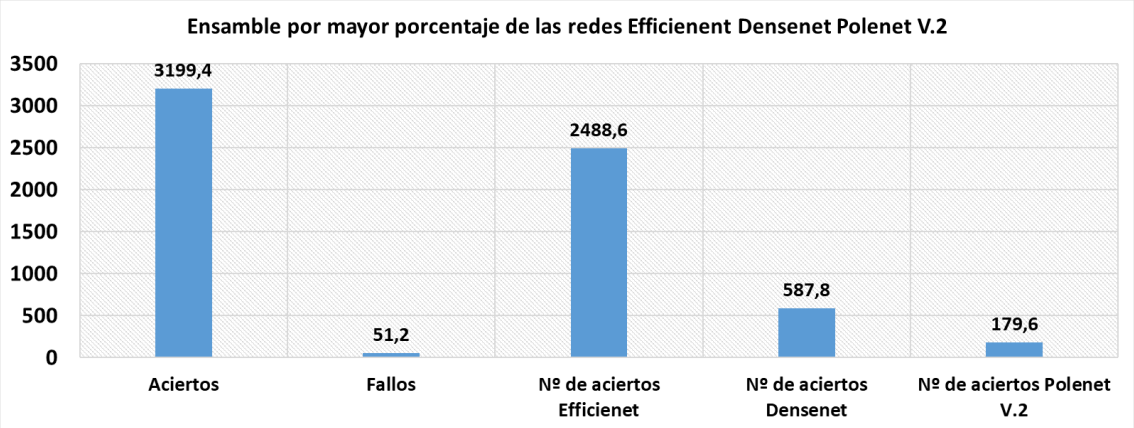


Figura 7.2: *Gráfica de resultados promedio al aplicar la técnica de ensamble de mayor porcentaje con las redes Efficienent, Densenet y Polenet V.2.*

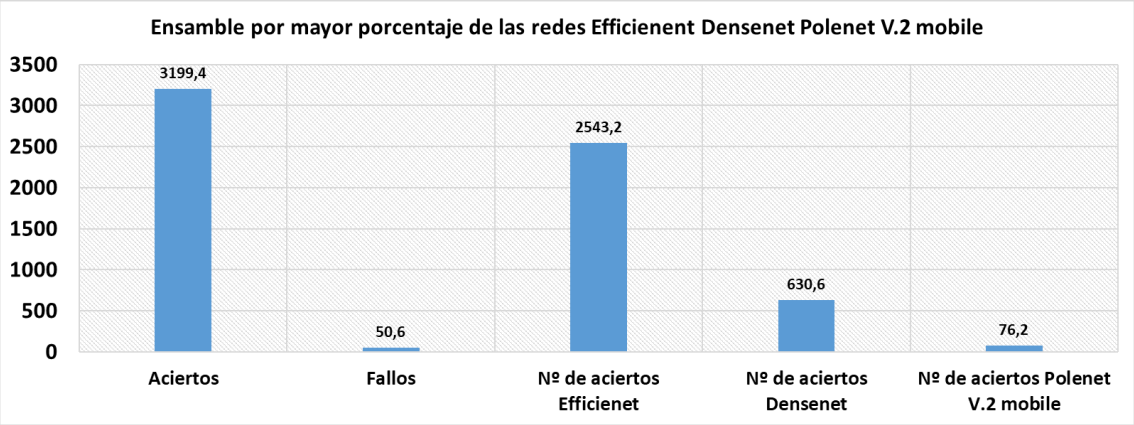


Figura 7.3: *Gráfica de resultados promedio al aplicar la técnica de ensamble de mayor porcentaje con las redes Efficienent, Densenet y Polenet V.2 mobile.*

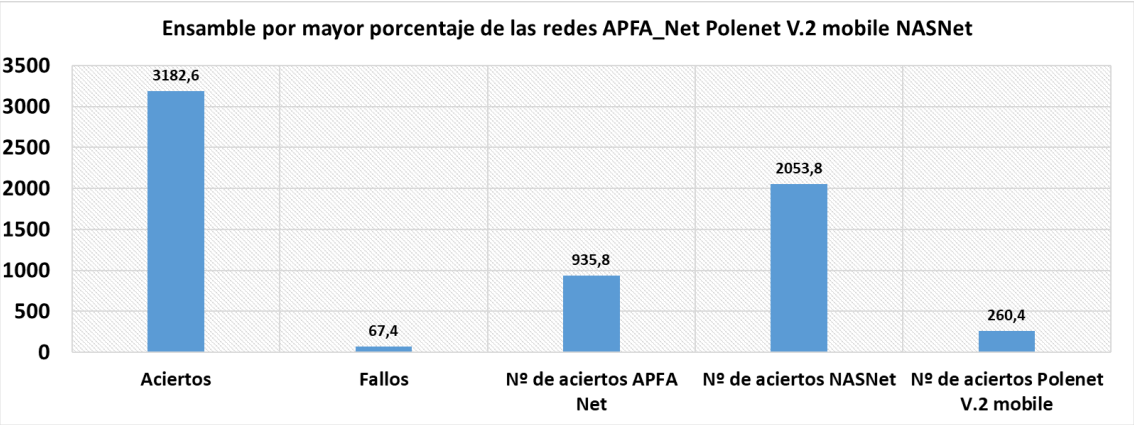


Figura 7.4: *Gráfica de resultados promedio al aplicar la técnica de ensamble de mayor porcentaje con las redes APFA Net, Polenet V.2 mobile y NASNet.*

7.2 Resultados mediante planteamiento de votación

Este planteamiento se basa en, una vez generadas las predicciones por todas las redes, se tomara como el tipo de la imagen aquella que haya salido más veces entre todas las redes. En el apéndice B.5 se muestra el código empleado para lograr esto.

Para comprobar el resultado de este planteamiento se mostrara en cada figura relativa a un *ensamble* los mismo parámetros que en el apartado anterior, salvo que al no ser por porcentaje mayor, se mostrará para cada red cuándo la predicción ha sido correcta y esa red ha acertado en la predicción (ver Figuras 7.5, 7.6, 7.7 y 7.8).

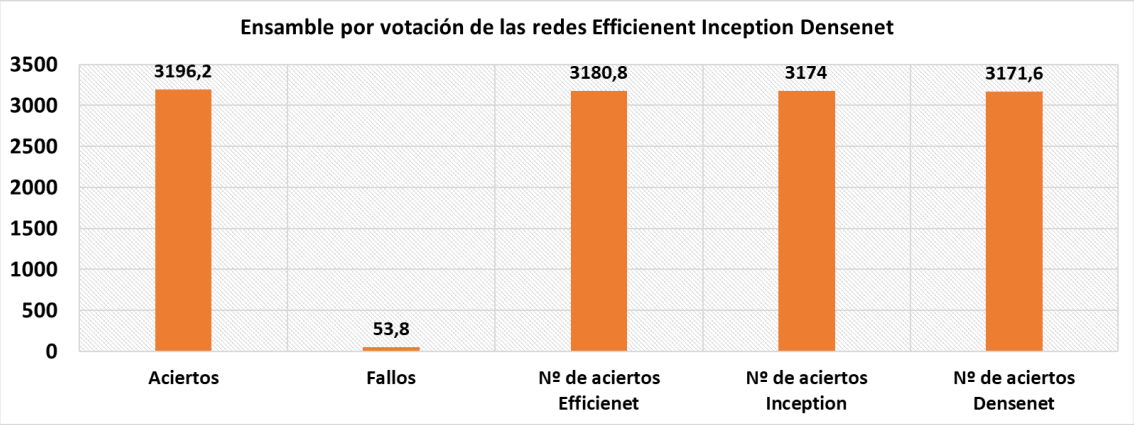


Figura 7.5: *Gráfica de resultados promedio al aplicar la técnica de ensamble de votación con las redes Efficienent, Inception y Densenet.*

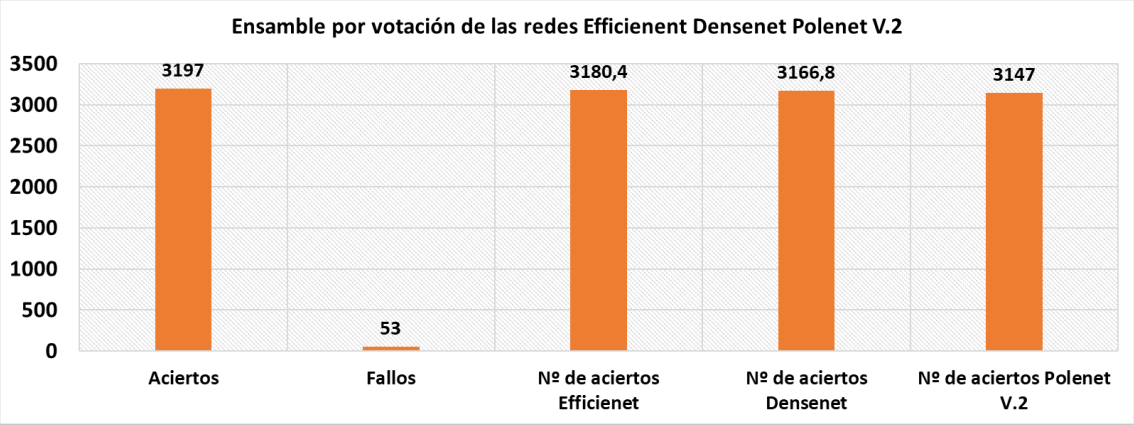


Figura 7.6: *Gráfica de resultados promedio al aplicar la técnica de ensamble de votación con las redes Efficienent, Densenet y Polenet V.2.*

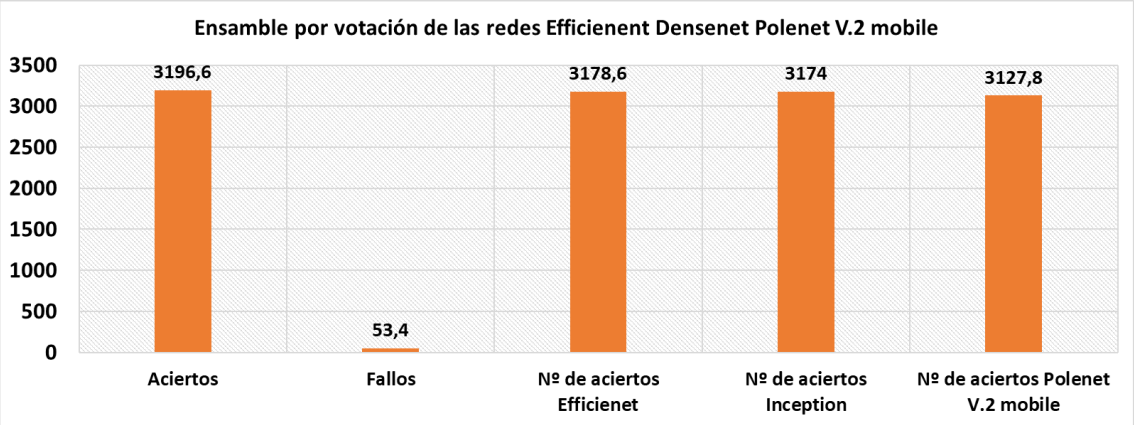


Figura 7.7: *Gráfica de resultados promedio al aplicar la técnica de ensamble de votación con las redes Efficienent, Densenet y Polenet V.2 mobile.*

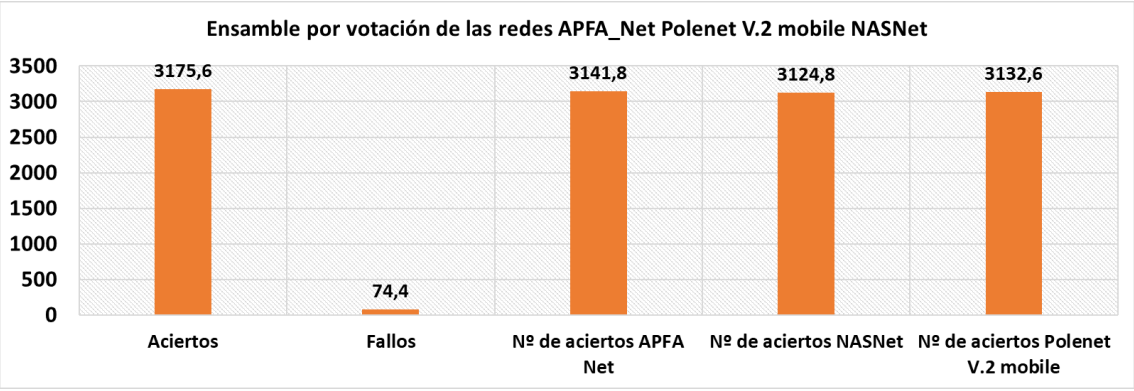


Figura 7.8: *Gráfica de resultados promedio al aplicar la técnica de ensamble de votación con las redes APFA Net, Polenet V.2 mobile y NASNet.*

7.3 Análisis de los resultados

Una vez completadas todas las pruebas y con el objetivo de presentar los resultados de la manera más clara posible, se mostrará la tabla 7.9 con el porcentaje de exactitud de los cuatro grupos de redes, para ambos criterios. Por otro lado, se mostrará una figura con el tamaño de cada grupo de redes para poder contextualizar la relación entre el aumento de exactitud a cambio del aumento de peso del conjunto (ver Figura 7.10).

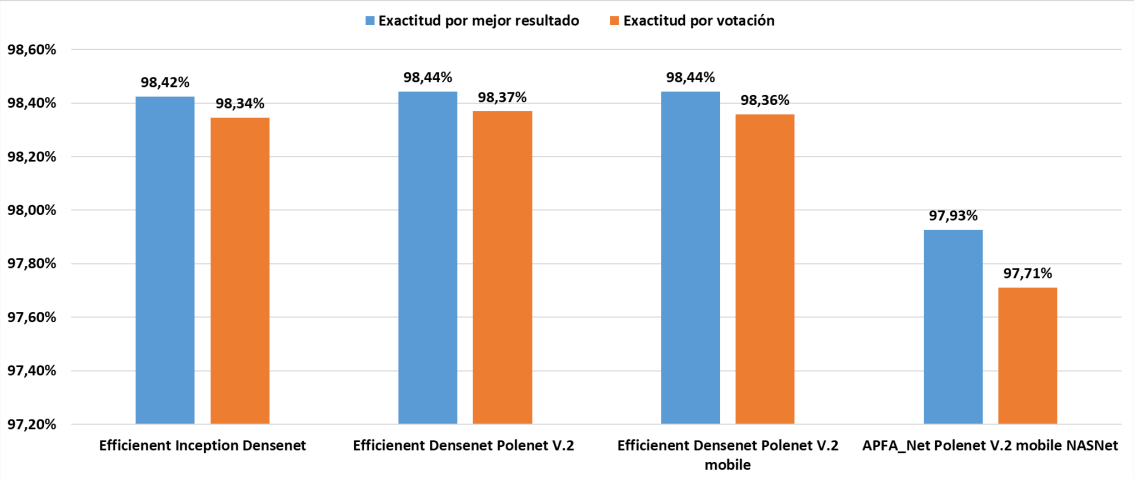


Figura 7.9: *Gráfica comparativa de los resultados promedio de la exactitud, para los cuatro conjuntos de redes y aplicando ambos criterios.*

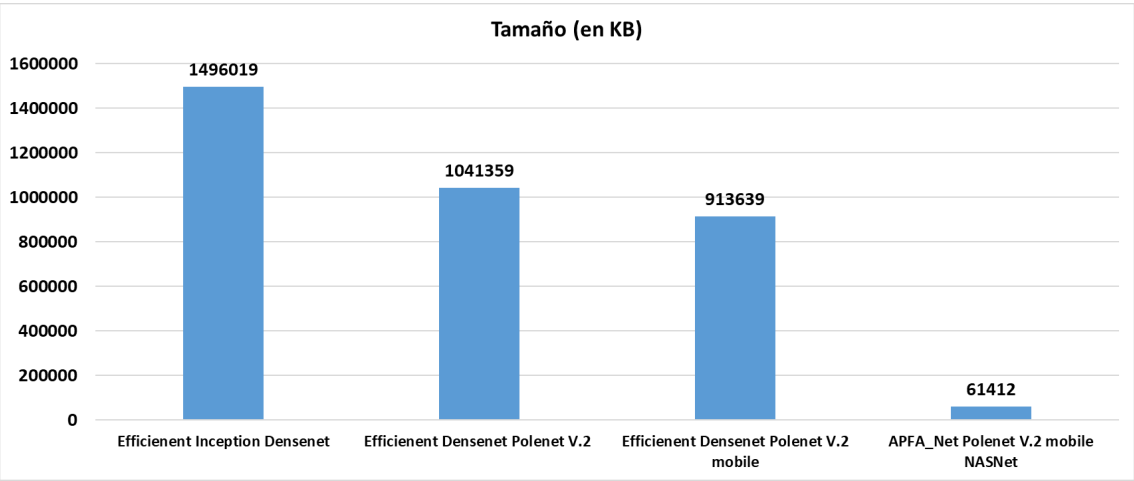


Figura 7.10: *Gráfica comparativa del tamaño de todos los conjuntos de redes empleados para las pruebas de ensamble.*

La primera conclusión que se puede apreciar con los datos obtenidos es que, en todos los casos, el criterio de mejor resultado es mucho más efectivo que el por votación. A nivel de exactitud, se puede apreciar que, comparando los resultados promedió, todas las combinaciones de redes salvo la ultima, presentan mejores resultados que el resto de redes de manera individual. Por lo que si se trata de una técnica efectiva.

Es importante mencionar el último caso analizado. Aunque puede parecer poco relevante ya que presenta una exactitud muy inferior al resto de las pruebas, si se contextualizar con las redes individuales, se puede apreciar que ha conseguido una exactitud de casi el 98 %, siendo este un valor muy similar a los obtenidos por redes como la InceptionV3 o la EfficientNetV2. La gran diferencia es que mientras que esas redes su peso supera los 500000 KB, el conjunto de estas tres redes no supera los 61412 KB. Logrando así el sistema de clasificación de imágenes con mejor relación tamaño exactitud visto en este trabajo.

CAPÍTULO 8
Trabajos Futuros

Como se ha podido ver a lo largo de todo el trabajo, este está dividido en tres lineas de desarrollo muy diferenciadas: el *dataset*, la arquitectura de la red propia y la herramienta de etiquetado. A continuación, se describirán las posibles ramas de investigación y desarrollo para cada campo.

Por lo que respecta al *dataset*, es probablemente la línea con un desarrollo más claro. El objetivo es aumentar los variedades de polen de las que se disponen, así como aumentar el número total de imágenes para todas las variedades. Esto se conseguirá de manera indirecta mientras se usa la herramienta de etiquetado.

En cuanto a la arquitectura de las redes desarrolladas, es necesario seguir realizando pruebas, quizás partiendo de un modelo distinto de las VGG y más similar a redes como la Inception o la DenseNet, ya que, las dos arquitecturas propuestas siguen manteniendo un rendimiento de casi un 1 % menos frente a las redes más exactas. Por supuesto, también es importante analizar las futuras posibles arquitecturas que se desarrollen, ya que este ámbito esta en constante crecimiento y pueden aparecer nuevos paradigmas para las topologías que cambien radicalmente su planteamiento teórico.

Por ultimo, el punto más importante a desarrollar tras este proyecto son las posibles mejoras que se pueden aplicar a la herramienta de etiquetado. La primera incorporación será añadir un sistema de clasificación autónomo para las muestras de polen que indiquen los técnicos. Esto consigue, en cierto modo, eliminar parcialmente el componente subjetivo del análisis polínico, así como servir como elemento comparativo para los técnicos a la hora de realizar comprobaciones.

Aunque en una primera versión se podría trabajar con solo una red, a la vista de los resultados, la introducción de varias redes en la aplicación permitiría a su vez aplicar técnicas de *ensamble*, las cuales afectan positivamente a los resultados del conjunto.

Por ultimo, la etapa final del desarrollo de la aplicación consistiría en introducir un sistema que no solo clasificaría las muestras de polen seleccionadas por los técnicos, sino aplicar redes del tipo R-CNN. Con estas redes, como se vio en el ejemplo 2.2.3, se produciría una clasificación y acote autónomo para las variedades de polen, permitiendo así disminuir drásticamente el tiempo necesario para realizar los análisis polínicos.

CAPÍTULO 9
Conclusiones

A la vista de los resultados dispuestos a lo largo del proyecto, ahora se dispone de una herramienta eficaz y útil para el proceso de etiquetado, con la cual, se ha logrado desarrollar uno de los *datasets* más completos en lo que respecta a pólenes de origen apícola.

En cuanto al desarrollo de las arquitecturas propias, se ha logrado el objetivo principal de mejorar el rendimiento y tamaño de la primera versión de la POLENET. Esto sitúa las dos versiones desarrolladas por encima de muchas arquitecturas ya asentadas en el campo de las redes convolucionales, tanto en rendimiento como en tamaño. Pese a ello, aún están algo lejos de las mejores arquitecturas, y por lo tanto, sigue siendo un área de posible mejora y futura investigación.

Por último, es importante destacar el valor de las técnicas de *ensamble*. Pese a sacrificar el tamaño del conjunto, ha quedado ampliamente demostrada su eficiencia frente a prácticamente cualquier red de manera individual, destacando sobre todo el planteamiento de mayor porcentaje.

A modo de conclusión del proyecto, se puede considerar que se ha logrado obtener grandes resultados en los cinco objetivos planteados. Pese a ello, este proyecto abre nuevas líneas de investigación y mejora que pueden permitir, en un futuro, grandes avances en este campo.

Bibliografía

[1] Kumar, K. S., Bhowmik, D., Biswajit, C., and Chandira, M. R. (2010). Medicinal uses and health benefits of honey: an overview. *J. Chem. Pharm. Res, 2(1), 385-395.* Recuperado el 19 de septiembre de 2023, de https://www.jocpr.com/

[2] UN Comtrade. (mayo 4, 2023). Ranking de los principales países exportadores de miel natural del mundo en función del valor de las exportaciones en 2022(en millones de dólares). [Gráfica]. In Statista. Recuperado el 19 de septiembre de 2023, de: https://es.statista.com/estadisticas/1008656/principales-exportadores-de-miel-natural-del-mundo/

[3] Ministerio de Agricultura, Alimentación y Medioambiente. 2002. Orden APA/3209/2002, de 3 de diciembre, por la que se ratifica el Reglamento de la Denominación de Origen Protegida «Miel de Granada» y de su Consejo Regulador. *B*OE num 301 del 17 de diciembre.

[4] Agencia de Gestión Agraria y Pesca de Andalucía. Apicultura: una ganadería particular e imprescindible (20/05/22). Recuperado el 19 de septiembre de 2023, de: https://ws229.juntadeandalucia.es/agenciaagrariaypesquera/ova/actualidad/noticia/apicultura-una-ganaderia-particular-e-imprescindible

[5] Ministerio de agricultura, pesca y Alimentación. INDICADORES ECONÓMICOS SECTOR APÍCOLA 2022. Recuperado el 20 de septiembre de 2023, de: https://www.mapa.gob.es/es/ganaderia/temas/produccion-y-mercados-ganaderos/indicadoreseconomicos2022_apicultura_versiontridion_tcm30-576093.pdf

[6] LOUVEAUX, J., & MAURIZIO,. V A. METHODS OF MELISSOPALYNOLOGY. Recuperado el 21 de septiembre de 2023, de: https://bitininkas.lt/lt/wp-content/uploads/2016/02/methmels_str1.pdf

[7] Navarrete Carolina, Muñoz-Olivera Gladys, Wells Guillermo, Becerra Julio, Alarcón Julio, Finot Víctor L. Espectro polínico y análisis fisicoquímico de mieles de la Región del Biobío, Chile. Gayana Bot. [Internet]. Recuperado el 21 de septiembre de 2023, de: http://www.scielo.cl/scielo.php?script=sci_arttext&pid=S0717-66432016000200268&lng=es.http://dx.doi.org/10.4067/S0717-66432016000200268.

[8] Erdtman, G. (1986). Pollen morphology and plant taxonomy: angiosperms (Vol. 1). Brill Archive. Recuperado el 21 de septiembre de 2023, de: https://books.google.es/books?hl=es&lr=&id=e-QUAAAAIAAJ&oi=fnd&pg=PA3&dq=ERDTMAN,+G.+1952.+Pollen+morphology+and+plant+taxonomy:+Angiosperms.+Almquist+%26+Wiksell.+Waltham,+USA.&ots=zQmt4MyqCC&sig=s7zpa_uKQFO-VHYV8ltpTNFkSvs#v=onepage&q&f=false

[9] Article Source: Feature Extraction and Machine Learning for the Classification of Brazilian Savannah Pollen Grains Gonçalves AB, Souza JS, Silva GGd, Cereda MP, Pott A, et al. (2016) Feature Extraction and Machine Learning for the Classification of Brazilian Savannah Pollen Grains. PLOS ONE 11(6): e0157044. Recuperado el 26 de septiembre de 2023, de: https://doi.org/10.1371/journal.pone.0157044

[10] Gilberto Astolfi, Ariadne Barbosa Gonçalves, Geazy Vilharva Menezes, Felipe Silveira Brito Borges, Angelica Christina Melo Nunes Astolfi, Edson Takashi Matsubara, Marco Alvarez, Hemerson Pistori, POLLEN73S: An image dataset for pollen grains classification, Ecological Informatics, Volume 60, 2020, 101165, ISSN 1574-9541. Recuperado el 26 de septiembre de 2023, de: https://www.sciencedirect.com/science/article/pii/S1574954120301151

[11] Tsiknakis, Nikos, Savvidaki, Elisavet, Kafetzopoulos, Sotiris, Manikis, Georgios, Vidakis, Nikolas, Marias, Kostas, and Alissandrakis, Eleftherios. (2021). Cretan Pollen Dataset v1 (CPD-1) (Version 1) [Data set]. Zenodo. Recuperado el 26 de septiembre de 2023, de: https://doi.org/10.5281/zenodo.4756361

[12] Battiato, S., Guarnera, F., Ortis, A., Trenta, F., Ascari, L., Siniscalco, C., ... and Suárez, E. 469 (2021). Pollen grain classification challenge 2020. International Conference on Pattern 470 Recognition (pp. 469-479). Springer, Cham. Recuperado el 28 de septiembre de 2023, de: https://doi.org/10.1007/978-3-030-68793-4719_34

[13] Taboga, Marco (2021). "K-fold cross-validation", Lectures on machine learning. Recuperado el 29 de septiembre de 2023, de: https://www.statlect.com/machine-learning/k-fold-cross-validation.

[14] Hastie, T., Tibshirani, R. & Friedman, J. H. (2009). The elements of statistical learning: data mining, inference, and prediction. 2nd Edition, New York, Springer.

[15] Aravinda (2023) How to split image dataset into train, validation and test set? Recuperado el 2 de octubre de 2023, de: https://aravinda-gn.medium.com/how-to-split-image-dataset-into-train-validation-and-test-set-5a41c48af332

[16] Javůrková, Z., Pospiech, M., Ljasovká, S., Hrabec, P., and Tremlová, B. (2021). Numerical methods and image processing techniques for melissopalynological honey analysis. Slovak Journal of Food Sciences, 15.

[17] Hernández, J. A., González, C. M. T., Rivas, J. T., Mora, F. M., Huertas, O. S., Bogantes, M. R., and Chavez, L. S. (2013). Sistema de detección y clasificación automática de granos de polen mediante técnicas de procesado digital de imágenes. Uniciencia, 27(1), 59-73.

[18] Marcos, J. V., Nava, R., Cristóbal, G., Redondo, R., Escalante-Ramírez, B., Bueno, G., ... and Rodríguez, T. (2015). Automated pollen identification using microscopic imaging and texture analysis. Micron, 68, 36-46.

[19] Fisher, R.A., 1936. The use of multiple measurements in taxonomic problems. 676 Annals of eugenics 7, 179–188.

[20] Microsoft Ignite Çognitive Toolkit, Tutorial 2"Micorosoft 2023 Recuperado el 16 de octubre de 2023, de: https://learn.microsoft.com/es-es/cognitive-toolkit/tutorial2/tutorial2

[21] López-Rubio, E., and García-González, J. (2020). Clasificación de imágenes de lesiones cutáneas mediante redes neuronales convolucionales y aprendizaje profundo.

[22] Bahrani, Hom. (2022). What is the role of "Flattenïn Keras?. Stack Overflow. Recuperado el 16 de octubre de 2023, de: https://stackoverflow.com/questions/43237124/what-is-the-role-of-flatten-in-keras

[23] Jeong, Jiwon (2019). The Most Intuitive and Easiest Guide for Convolutional Neural Network. Medium. Recuperado el 16 de octubre de 2023, de: https://towardsdatascience.com/the-most-intuitive-and-easiest-guide-for-convolutional-neural-network

[24] Simonyan, K., and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

[25] Pérez-Aguilar, D., Risco-Ramos, R., and Casaverde-Pacherrez, L. (2021). Transfer learning en la clasificación binaria de imágenes térmicas transfer learning for binary classification of thermal images. Mach. Learn., 550(26), 4.

[26] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[27] Data Science Team. (2020). Redes neuronales residuales – Lo que necesitas saber (ResNet). Data Science. Recuperado el 17 de octubre de 2023, de: https://datascience.eu/es/aprendizaje-automatico/una-vision-general-de-resnet-y-sus-variantes/

[28] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2818-2826).

[29] Guía avanzada de Inception v3. (2023). Google cloud. Recuperado el 17 de octubre de 2023, de: https://cloud.google.com/tpu/docs/inception-v3-advanced?hl=es-419

[30] Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1251-1258).

[31] Mehmood, A. (2021). Efficient anomaly detection in crowd videos using pre-trained 2D convolutional neural networks. IEEE Access, 9, 138283-138295.

[32] Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4700-4708).

[33] Feng, X., Yao, H. and Zhang, S. An efficient way to refine DenseNet. SIViP 13, 959–965 (2019). Recuperado el 18 de octubre de 2023, de: https://doi.org/10.1007/s11760-019-01433-4

[34] Tan, M., and Le, Q. (2019, May). Efficientnet: Rethinking model scaling for convolutional neural networks. In International conference on machine learning (pp. 6105-6114). PMLR.

[35] Huang, M. L., and Liao, Y. C. (2023). Stacking ensemble and ECA-EfficientNetV2 convolutional neural networks on classification of multiple chest diseases including COVID-19. Academic Radiology, 30(9), 1915-1935.

[36] Howard, A., Zhmoginov, A., Chen, L. C., Sandler, M., and Zhu, M. (2018). Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation.

[37] Akay, M., Du, Y., Sershen, C. L., Wu, M., Chen, T. Y., Assassi, S., ... and Akay, Y. M. (2021). Deep learning classification of systemic sclerosis skin using the MobileNetV2 model. IEEE Open Journal of Engineering in Medicine and Biology, 2, 104-110.

[38] Bui, D. T., Tran, T. D., Nguyen, T. T., Tran, Q. L., and Nguyen, D. V. (2018). Aerial image semantic segmentation using neural search network architecture. In Multi-disciplinary Trends in Artificial Intelligence: 12th International Conference, MIWAI 2018, Hanoi, Vietnam, November 18–20, 2018, Proceedings 12 (pp. 113-124). Springer International Publishing.

[39] Mahmood, T., Choi, J., and Park, K. R. (2023). Artificial intelligence-based classification of pollen grains using attention-guided pollen features aggregation network. Journal of King Saud University-Computer and Information Sciences, 35(2), 740-756.

[40] Raghu, M., Unterthiner, T., Kornblith, S., Zhang, C., and Dosovitskiy, A. (2021). Do vision transformers see like convolutional neural networks?. Advances in Neural Information Processing Systems, 34, 12116-12128.

[41] Gaudenz Boesch. Vision Transformers (ViT) in Image Recognition – 2023 Guide. (2023). viso.ai Recuperado el 18 de octubre de 2023, de: https://viso.ai/deep-learning/vision-transformer-vit/

[42] Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 580-587).

[43] Park, M., Yang, W., Cao, Z., Kang, B., Connor, D., and Lea, M. A. (2019). Marine vertebrate predator detection and recognition in underwater videos by region convolutional neural network. In Knowledge Management and Acquisition for Intelligent Systems: 16th Pacific Rim Knowledge Acquisition Workshop, PKAW 2019, Cuvu, Fiji, August 26–27, 2019, Proceedings 16 (pp. 66-80). Springer International Publishing.

[44] TensorFlow. (2023). "tf.keras.metrics.Precision". Recuperado el 22 de octubre de 2023, de: https://www.tensorflow.org/api_docs/python/tf/keras/metrics/Precision

[45] TensorFlow. (2023). "tf.keras.metrics.Accuracy". Recuperado el 22 de octubre de 2023, de: https://www.tensorflow.org/api_docs/python/tf/keras/metrics/Accuracy

[46] TensorFlow. (2023). "tf.keras.metrics.Recall". Recuperado el 22 de octubre de 2023, de: https://www.tensorflow.org/api_docs/python/tf/keras/metrics/Recall

[47] TensorFlow. (2023). "tf.keras.metrics.F1Score". Recuperado el 22 de octubre de 2023, de: https://www.tensorflow.org/api_docs/python/tf/keras/metrics/F1Score

[48] TensorFlow. (2023). "tfa.metrics.MatthewsCorrelationCoefficient". Recuperado el 22 de octubre de 2023, de: https://www.tensorflow.org/addons/api_docs/python/tfa/metrics/MatthewsCorrelationCoefficient

[49] Hernández-García, A., and König, P. (2018). Further advantages of data augmentation on convolutional neural networks. In Artificial Neural Networks and Machine Learning–ICANN 2018: 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part I 27 (pp. 95-103). Springer International Publishing.

[50] Arroyo Hernandez, J., Travieso Gonzalez, C. M., Ticay Rivas, J. R., Mora Mora, F., Salas Huertas, O., Ramirez Bogantes, M., and Sanchez Chaves, L. (2013). System Detection And Automatic Classification Of Pollen Grain Applies Technical Digital Imaging Process. Uniciencia.

[51] Keras. Keras Applications. Recuperado el 26 de octubre de 2023, de: `https://keras.io/api/applications/`

[52] Optuna. Optuna code examples. Recuperado el 07de noviembre de 2023, de: `https://optuna.org/#code_examples`

[53] Gorodkin, J. (2004). Comparing two K-category assignments by a K-category correlation coefficient. Computational biology and chemistry, 28(5-6), 367-374.

[54] Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., ... and He, Q. (2020). A comprehensive survey on transfer learning. Proceedings of the IEEE, 109(1), 43-76.

APÉNDICE A
Sección de Ambiente de Desarrollo

En el siguiente apartado se especificarán tanto las piezas de hardware relevantes para el correcto funcionamiento del ecosistema como las versiones de sistema operativo, lenguajes y bibliotecas que se han empleado a lo largo del proyecto para el entrenamiento de las distintas redes presentadas:

- **Tarjeta gráfica:** Se han realizado pruebas tanto la *Nvidea GeForce RTX 3080* como con la *Nvidea GeForce RTX 3090*
- **Sistema operativo:** Se han realizado pruebas tanto en Windows 10 como en Windows 11.
- **Plataforma de cómputo paralelo:** CUDA `versión 11.2.`
- **Bibliotecas de CUDA:** CuDNN `versión 8.1.` para acelerar para acelerar el entrenamiento y la inferencia de redes neuronales profundas en hardware compatible con GPU
- **Herramientas de desarrollo adicionales:** Paquete de *desarrollo para el escritorio con c++* del entorno Visual Studio 2019
- **Lenguaje de programación:** Pyhton `versión 3.10.`
- **Bibliotecas de Python utilizadas:**
  - **NumPy** → `versión 1.25.2`
  - **Keras** → `versión 2.10`
  - **Tensorflow** → `versión 2.10`
  - **Matplotlib** → `versión 3.7.2`
  - **Optuna** → `versión 3.3`
  - **Pydot_ng** → `versión 2.0`
  - **Sklearn** → `versión 0.0.post7`
  - **vit-keras** → `versión 0.1.2`
  - **tensorflow-addons** → `versión 0.21.0`
  - **opencv-python** → `versión 4.8.0.76`

APÉNDICE B
Código en Python

B.1 División del *dataset* y aplicación de la metodología *5-fold*

Este código se ha empleado para poder aplicar la división del *dataset* en grupos de entrenamiento con el 80 % de las imágenes, validación con el 10 % y test con el otro 10 %. Además, aplicará la metodología de *k-fold* generando las 5 divisiones distintas.

```
import os
import random
import shutil
import tkinter as tk
from tkinter import filedialog
from collections import defaultdict


root = tk.Tk()
root.withdraw()  # Ocultar la ventana principal

data_path = filedialog.askdirectory(title="Selecciona la carpeta de origen")

if not data_path:
    print("No se ha seleccionado una carpeta de origen. El proceso ha sido cancelado.")
else:
    output_base_folder = "C:/Users/JUAMAROS/Desktop/Nueva carpeta (2)/POLLEN23S_dataset_preparado"

    image_extensions = ['.jpg', '.jpeg', '.png', '.bmp', '.tif', '.TIF']

    num_images = 0
    num_folders = 0

    for root, dirs, files in os.walk(data_path):
        num_folders += 1
        for filename in files:
            if os.path.splitext(filename)[-1] in image_extensions:
                num_images += 1

print(f"Numero de carpetas encontradas en {data_path}: {num_folders}")
print(f"Numero de imagenes encontradas en {data_path}: {num_images}")

    for repetition in range(1, 6):
        output_folder = os.path.join(output_base_folder, f"output{repetition}")
        train_folder = os.path.join(output_folder, 'train')
        val_folder = os.path.join(output_folder, 'val')
        test_folder = os.path.join(output_folder, 'test')
```

```
38
39          if not os.path.exists(output_folder):
40              os.makedirs(output_folder)
41          if not os.path.exists(train_folder):
42              os.makedirs(train_folder)
43          if not os.path.exists(val_folder):
44              os.makedirs(val_folder)
45          if not os.path.exists(test_folder):
46              os.makedirs(test_folder)
47
48
49          image_types = defaultdict(list)
50
51          for root, dirs, files in os.walk(data_path):
52              for filename in files:
53                  if os.path.splitext(filename)[-1] in image_extensions:
54                      image_type = os.path.basename(root)
55                      image_path = os.path.join(root, filename)
56                      image_types[image_type].append(image_path)
57
58
59          for image_type, image_list in image_types.items():
60              random.shuffle(image_list)
61              total_images = len(image_list)
62              train_size = int(0.8 * total_images)
63              val_size = int(0.1 * total_images)
64              test_size = total_images - train_size - val_size
65
66              train_images = image_list[:train_size]
67              val_images = image_list[train_size:train_size + val_size]
68              test_images = image_list[train_size + val_size:]
69
70              for dest_folder, images in [(train_folder, train_images), (
                    val_folder, val_images), (test_folder, test_images)]:
71                  type_folder = os.path.join(dest_folder, image_type)
72                  os.makedirs(type_folder, exist_ok=True)
73                  for src_path in images:
74                      relative_path = os.path.relpath(src_path, data_path)
75                      dest_file_path = os.path.join(type_folder, os.path.basename
                            (src_path))
76                      os.makedirs(os.path.dirname(dest_file_path), exist_ok=True)
77                      shutil.copy(src_path, dest_file_path)
78
79      print("Proceso finalizado")
```

B.2 Entrenamiento de una red neuronal

En este apéndice se muestra un ejemplo del código utilizado para entrenar las distintas redes neuronales, en este caso la arquitectura nombrada como prueba 2. El código no solo realiza el entrenamiento, sino que además realiza el proceso de prueba, dando como resultado, entre otros elementos de análisis, la matriz de confusión.

```
1 # Hyperparameters
2 target_size = (256, 256)
3 batch_size = 32
4
5 import os
6 from tensorflow.keras.preprocessing.image import ImageDataGenerator
7 # from tensorflow.keras.preprocessing import image_dataset_from_directory
```

```
from tensorflow.keras.applications.vgg16 import preprocess_input
from tensorflow.keras.applications import VGG16  # , imagenet_utils
from tensorflow.keras import optimizers
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dropout, Flatten, Dense
from tensorflow.keras import Model
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, matthews_corrcoef
import numpy as np
from tensorflow.keras import layers
from tensorflow.keras import models
import pydot_ng as pydot
from tensorflow.keras.utils import plot_model

epochs = 350

# Importando el set de datos
dataset_path = "./PollenSamplesAll_100/"
min_num_samples = 10


print("all samples:")
dirs = os.listdir(dataset_path)
num_samples = [len(files) for r, d, files in os.walk(dataset_path)]
num_samples = num_samples[1:]  # exclude first top directory
print(dirs)
print(num_samples)

ok_samples = [[j, i] for (i, j) in zip(num_samples, dirs) if i >=
    min_num_samples]

labels = [j for (i, j) in zip(num_samples, dirs) if i >= min_num_samples]
labels.sort();


# Data generators para poder hacer data augmentation
datagen = ImageDataGenerator(
    rotation_range=int(180 * 0.1),
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True,
    vertical_flip=True,
    # preprocessing_function = preprocess_input
    rescale=1. / 255
)

print("train_ds:")
train_ds = datagen.flow_from_directory("./output1/train", classes=labels,
    target_size=target_size,batch_size=batch_size, class_mode='categorical')
print("val_ds:")
val_ds = datagen.flow_from_directory("./output1/val", classes=labels,
    target_size=target_size, batch_size=batch_size,class_mode='categorical')

datagen = ImageDataGenerator(
    # preprocessing_function = preprocess_input
    rescale=1. / 255)

print("test_ds:")
test_ds = datagen.flow_from_directory("./output1/test", classes=labels,
    target_size=target_size,class_mode='categorical', shuffle=False)


```

```
68 def Polenet(width, height, depth, classes):
69     inputShape = (height, width, depth)
70 
71     model = Sequential()
72     model.add(layers.Conv2D(input_shape=(256, 256, 3), filters=64, kernel_size
           =(5, 5), padding="same", activation="relu",name="Conv2D1"))
73 
74     model.add(layers.MaxPool2D(pool_size=(3, 3), strides=(2, 2), name="
           MaxPool2D1"))
75     model.add(layers.Conv2D(filters=128, kernel_size=(5, 5), padding="same",
           activation="relu", name="Conv2D2"))
76 
77     model.add(layers.MaxPool2D(pool_size=(3, 3), strides=(2, 2), name="
           MaxPool2D2"))
78     model.add(layers.Conv2D(filters=256, kernel_size=(5, 5), padding="same",
           activation="relu", name="Conv2D3"))
79 
80     model.add(layers.MaxPool2D(pool_size=(3, 3), strides=(2, 2), name="
           MaxPool2D3"))
81     model.add(layers.Conv2D(filters=512, kernel_size=(5, 5), padding="same",
           activation="relu", name="Conv2D4"))
82 
83     model.add(layers.MaxPool2D(pool_size=(3, 3), strides=(2, 2), name="
           MaxPool2D4"))
84     model.add(layers.Conv2D(filters=512, kernel_size=(5, 5), padding="same",
           activation="relu", name="Conv2D5"))
85 
86     model.add(layers.MaxPool2D(pool_size=(3, 3), strides=(2, 2), name="
           MaxPool2D5"))
87     model.add(layers.Flatten(name="Flatten"))
88     model.add(layers.Dense(units=1000, activation="relu", name="Dense1"))
89 
90     model.add(layers.Dense(units=200, activation="relu", name="Dense2"))
91     model.add(layers.Dense(units=train_ds.num_classes, activation='softmax',
           name="FinalStage"))
92 
93     return model
94 
95 
96 # Instanciamos el modelo
97 model = Polenet(target_size[0], target_size[1], 3, train_ds.num_classes)
98 # Compilamos el modelo
99 
100 # Compilamos el modelo
101 model.compile(loss='categorical_crossentropy', optimizer=SGD(0.005), metrics=['
        accuracy'])
102 
103 
104 model.summary()
105 
106 # Entrenamos el modelo
107 print("[INFO]: Entrenando la red...")
108 H = model.fit(train_ds, epochs=epochs, validation_data=val_ds, verbose=1)
109 
110 # Graficas
111 plt.style.use("ggplot")
112 plt.figure()
113 plt.plot(np.arange(1, epochs + 1), H.history["loss"], label="train_loss")
114 plt.plot(np.arange(1, epochs + 1), H.history["val_loss"], label="val_loss")
115 plt.plot(np.arange(1, epochs + 1), H.history["accuracy"], label="train_acc")
116 plt.plot(np.arange(1, epochs + 1), H.history["val_accuracy"], label="val_acc")
117 plt.title("Training Loss and Accuracy")
118 plt.xlabel("Epoch #")
119 plt.ylabel("Loss/Accuracy")
```

```
plt.legend()
plt.savefig('Polenet_all_layers_output1.png')

# Almacenamos el modelo empleando la funcion mdoel.save de Keras
model.save("./Polenet_all_layers_output1.h5")

plot_model(model, show_shapes=True, to_file='Polenet_model.png',
    show_layer_names=True, rankdir='TB')

from tensorflow.keras.models import load_model
from sklearn.metrics import confusion_matrix

print("test_ds:")
test_ds = datagen.flow_from_directory("./output1/test", classes=labels,
    target_size=target_size,
                                      class_mode='categorical', shuffle=False)


# Evaluamos con las muestras de test
print("[INFO]: Evaluando modelo...")
results = model.evaluate(test_ds)
print("test loss, test acc:", results)

# Generate predictions (probabilities -- the output of the last layer)
# on new data using predict
print("Generate predictions")
predictions = model.predict(test_ds)
print("predictions shape:", predictions.shape)
# Obtenemos el report
print(classification_report(test_ds.classes, predictions.argmax(axis=1),
    target_names=labels))  # Etiquetas en decimal #(X)
print('Confusion Matrix')
print(confusion_matrix(test_ds.classes, predictions.argmax(axis=1)))
MCC = matthews_corrcoef(test_ds.classes, predictions.argmax(axis=1))
print('MCC: ', MCC)
```

B.3 Código de ajuste de hipeparámetros y entrenamiento mediante Optuna

En este apéndice se muestra un ejemplo de código para entrenar la red POLENET V.2. Antes del proceso de entrenamiento y prueba, se emplea la biblioteca Optuna para realizar el ajuste de los hiperparámetros de *learning rate* y *dropout*.

```
import os
import optuna
from optuna.integration.tfkeras import TFKerasPruningCallback
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, matthews_corrcoef
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, BatchNormalization, MaxPooling2D,
    GlobalAveragePooling2D, Dense, Flatten, Dropout
from tensorflow.keras import layers
from tqdm import tqdm


```

```
target_size = (256, 256)
batch_size = 32
epochs_optuna = 50
epochs_fianles = 350
min_num_samples = 10
n_optuna  = 50


def Polenet(width, height, depth, classes):
    inputShape = (height, width, depth)

    model = Sequential()
    model.add(layers.Conv2D(input_shape=inputShape, filters=64, kernel_size=(3,
        3), padding="same", activation="relu",name="Conv2D1"))
    model.add(layers.BatchNormalization())
    model.add(layers.MaxPool2D(pool_size=(2, 2), strides=(2, 2), padding='valid
        ', name="MaxPool2D1"))

    model.add(layers.Conv2D(filters=128, kernel_size=(3, 3), padding="same",
        activation="relu", name="Conv2D2"))
    model.add(layers.BatchNormalization())
    model.add(layers.MaxPool2D(pool_size=(2, 2), strides=(2, 2), padding='valid
        ', name="MaxPool2D2"))

    model.add(layers.Conv2D(filters=256, kernel_size=(3, 3), padding="same",
        activation="relu", name="Conv2D3"))
    model.add(layers.BatchNormalization())
    model.add(layers.MaxPool2D(pool_size=(2, 2), strides=(2, 2), padding='valid
        ', name="MaxPool2D3"))

    model.add(layers.Conv2D(filters=512, kernel_size=(3, 3), padding="same",
        activation="relu", name="Conv2D4"))
    model.add(layers.BatchNormalization())
    model.add(layers.MaxPool2D(pool_size=(2, 2), strides=(2, 2), padding='valid
        ', name="MaxPool2D4"))

    model.add(layers.Conv2D(filters=512, kernel_size=(3, 3), padding="same",
        activation="relu", name="Conv2D5"))
    model.add(layers.BatchNormalization())
    model.add(layers.MaxPool2D(pool_size=(2, 2), strides=(2, 2), padding='valid
        ', name="MaxPool2D5"))

    model.add(layers.Flatten(name="Flatten"))
    model.add(layers.Dense(units=1000, activation="relu", name="Dense1"))
    model.add(layers.Dense(units=200, activation="relu", name="Dense2"))
    model.add(layers.Dense(units=classes, activation='softmax', name="
        FinalStage"))

    return model



def build_model(params):
    learning_rate = params["learning_rate"]
    dropout_rate = params["dropout_rate"]

    model = Polenet(target_size[0], target_size[1], 3, train_ds.num_classes)
    model.compile(
        loss='categorical_crossentropy',
        optimizer=SGD(learning_rate),
        metrics=['accuracy']
    )
    return model

```

```
68 def objective(trial):
69     def build_model(trial):
70         learning_rate = trial.suggest_loguniform("learning_rate", 1e-5, 1e-1)
71         dropout_rate = trial.suggest_uniform("dropout_rate", 1e-5, 0.5)
72 
73         model = Polenet(target_size[0], target_size[1], 3, train_ds.num_classes
               )
74         model.compile(
75             loss='categorical_crossentropy',
76             optimizer=SGD(learning_rate),
77             metrics=['accuracy']
78         )
79         return model
80 
81     model = build_model(trial)
82 
83     H = model.fit(
84         train_ds,
85         epochs=epochs_optuna,
86         validation_data=val_ds,
87         verbose=0,
88         callbacks=[
89             TFKerasPruningCallback(trial, "val_loss")
90         ]
91     )
92 
93     val_accuracy = H.history["val_accuracy"][-1]
94 
95 
96     trial.set_user_attr("val_accuracy", val_accuracy)
97     tqdm.write(f"Trial #{trial.number}: Validation Accuracy = {val_accuracy:.4f
           }")
98 
99     return val_accuracy
100 
101 
102 dataset_path = "./PollenSamplesAll_100/"
103 
104 dirs = os.listdir(dataset_path)
105 num_samples = [len(files) for r, d, files in os.walk(dataset_path)]
106 num_samples = num_samples[1:]
107 
108 ok_samples = [[j, i] for (i, j) in zip(num_samples, dirs) if i >=
        min_num_samples]
109 labels = [j for (i, j) in zip(num_samples, dirs) if i >= min_num_samples]
110 labels.sort()
111 
112 datagen1 = ImageDataGenerator(
113     rotation_range=90,
114     width_shift_range=0.15,
115     height_shift_range=0.15,
116     zoom_range=0.15,
117     horizontal_flip=True,
118     vertical_flip=True,
119     brightness_range=[0.5, 1.5],
120     rescale=1./255
121 )
122 
123 datagen2 = ImageDataGenerator(rescale=1. / 255)
124 
125 train_ds = datagen1.flow_from_directory("./output5/train", classes=labels,
        target_size=target_size, batch_size=batch_size, class_mode='categorical')
126 val_ds = datagen2.flow_from_directory("./output5/val", classes=labels,
        target_size=target_size, batch_size=batch_size, class_mode='categorical')
```

```
127
128 datagen3 = ImageDataGenerator(rescale=1./255)
129
130 test_ds = datagen3.flow_from_directory("./output5/test", classes=labels,
        target_size=target_size, class_mode='categorical', shuffle=False)
131
132 study = optuna.create_study(direction="maximize")
133 with tqdm(total=n_optuna) as pbar:
134     study.optimize(objective, n_trials=n_optuna, callbacks=[lambda study, trial
            : pbar.update(1)])
135
136 best_params = study.best_params
137 best_learning_rate = best_params["learning_rate"]
138 best_dropout_rate = best_params["dropout_rate"]
139
140 print("Mejores hiperparametros encontrados:")
141 print("Learning Rate:", best_learning_rate)
142 print("Dropout Rate:", best_dropout_rate)
143
144
145 final_model = build_model(best_params)
146 H = final_model.fit(train_ds, epochs=epochs_fianles, validation_data=val_ds,
        verbose=1)
147
148 plt.style.use("ggplot")
149 plt.figure()
150 plt.plot(np.arange(1, epochs_fianles+1), H.history["loss"], label="train_loss")
151 plt.plot(np.arange(1, epochs_fianles+1), H.history["val_loss"], label="val_loss
        ")
152 plt.plot(np.arange(1, epochs_fianles+1), H.history["accuracy"], label="
        train_acc")
153 plt.plot(np.arange(1, epochs_fianles+1), H.history["val_accuracy"], label="
        val_acc")
154 plt.title("Training Loss and Accuracy")
155 plt.xlabel("Epoch #")
156 plt.ylabel("Loss/Accuracy")
157 plt.legend()
158 plt.savefig('loss_vs_accuracy_Polenet_V2_all_layers_output5.png')
159
160
161 final_model.save("./Polenet_V2_all_layers_output5.h5")
162
163
164
165 from tensorflow.keras.models import load_model
166 from sklearn.metrics import confusion_matrix
167
168 print("test_ds:")
169
170 print("[INFO]: Evaluando modelo...")
171 results = final_model.evaluate(test_ds)
172 print("test loss, test acc:", results)
173
174 print("Generate predictions")
175 predictions = final_model.predict(test_ds)
176 print("predictions shape:", predictions.shape)
177
178 print(classification_report(test_ds.classes, predictions.argmax(axis=1),
        target_names=labels))
179 print('Confusion Matrix')
180 print(confusion_matrix(test_ds.classes, predictions.argmax(axis=1)))
181 MCC = matthews_corrcoef(test_ds.classes, predictions.argmax(axis=1))
182 print('MCC: ', MCC)
```

B.4 Código *ensamble* con el criterio de mayor porcentaje

En este apéndice se muestra un ejemplo del código utilizado para aplicar la técnica de *ensemble* con el criterio de mayor porcentaje. Al ejecutar este código, aparece un menú desplegable para seleccionar las tres redes y la carpeta con las imágenes de prueba.

```
target_size1 = (224, 224, 3)
target_size2 = (331, 331, 3)
target_size3 = (256, 256, 3)

import numpy as np
from tensorflow.keras.models import load_model
import imutils
from tensorflow.keras import Model
from tensorflow.keras import models
from matplotlib import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
import cv2
import tkinter as tk
from tkinter import filedialog
from tensorflow.keras.applications.inception_v3 import preprocess_input as
    prepro_inceptionV3
from tensorflow.keras.applications.vgg16 import preprocess_input as
    prepro_vgga16
import os

labelNames = ['type_000', 'type_003', 'type_004', 'type_007', 'type_008', '
    type_011', 'type_012',
'type_014', 'type_017', 'type_021', 'type_022', 'type_024', 'type_026', '
    type_027', 'type_028', 'type_030', 'type_033', 'type_036', 'type_037', '
    type_040', 'type_043', 'type_047', 'type_050', 'type_055']

NRed1 = 0
NRed2 = 0
NRed3 = 0

total = 0
aciertos= 0

root = tk.Tk()
root.withdraw()

model1_file_path = filedialog.askopenfilename(title="Seleciona la red 1 para el
     ensamble (.h5)", filetypes=[("HDF5 files", "*.h5")])
model2_file_path = filedialog.askopenfilename(title="Seleciona la red 2 para el
     ensamble (.h5)", filetypes=[("HDF5 files", "*.h5")])
model3_file_path = filedialog.askopenfilename(title="Seleciona la red 3 para el
     ensamble (.h5)", filetypes=[("HDF5 files", "*.h5")])

model1_name= os.path.basename(model1_file_path)
model2_name = os.path.basename(model2_file_path)
model3_name = os.path.basename(model3_file_path)

model_Red1 = load_model(model1_file_path)
model_Red2 = load_model(model2_file_path)
model_Red3 = load_model(model3_file_path)



folder_path = filedialog.askdirectory(title="Seleccione la carpeta de prueba")
```

```
48
49 datagen = ImageDataGenerator()
50
51 test = datagen.flow_from_directory(folder_path, classes=labelNames, class_mode=
       'categorical', shuffle=False)
52
53 for i in range(test.samples):
54   img = image.imread(test.filepaths[i])
55
56   img1 = cv2.resize(img, (224, 224))
57   img2 = cv2.resize(img, (331, 331))
58   img3 = cv2.resize(img, (256, 256))
59
60   img3 = img3.astype("double") / 255.0
61
62   img1 = np.expand_dims(img1, axis=0)
63   img2 = np.expand_dims(img2, axis=0)
64   img3 = np.expand_dims(img3, axis=0)
65
66   img1=prepro_vgga16(img1)
67   img2 = prepro_vgga16(img2)
68
69   prob_red_1 = model_Red1.predict(img1)
70   prob_red_2 = model_Red2.predict(img2)
71   prob_red_3 = model_Red3.predict(img3)
72
73   proba_red_1 = max(max(prob_red_1))
74   proba_red_2 = max(max(prob_red_2))
75   proba_red_3 = max(max(prob_red_3))
76
77
78   idx_red_1= np.argmax(prob_red_1)
79   label_red_1 = labelNames[idx_red_1]
80   print(model1_name)
81   print(label_red_1)
82   print (proba_red_1)
83
84   idx_red_2 = np.argmax(prob_red_2)
85   label_red_2 = labelNames[idx_red_2]
86   print(model2_name)
87   print(label_red_2)
88   print(proba_red_2)
89
90   idx_red_3 = np.argmax(prob_red_3)
91   label_red_3 = labelNames[idx_red_3]
92   print(model3_name)
93   print(label_red_3)
94   print(proba_red_3)
95
96
97   if proba_red_1 >= proba_red_2 and proba_red_1 >= proba_red_3 :
98     NRed1 = NRed1 +1
99     idx = np.argmax(prob_red_1)
100    label = labelNames[idx]
101    if label == labelNames[test.labels[i]]:
102      aciertos = aciertos + 1
103    print(label, labelNames[test.labels[i]])
104    print(model1_name)
105    total = total +1
106
107  elif proba_red_2 >= proba_red_1 and proba_red_2 >= proba_red_3 :
108    NRed2 = NRed2 + 1
109    idx = np.argmax(prob_red_2)
110    label = labelNames[idx]
```

```
    if label == labelNames[test.labels[i]]:
      aciertos = aciertos + 1
    print(label, labelNames[test.labels[i]])
    print(model2_name)
    total = total +1

  else:
    NRed3 = NRed3 +1
    idx = np.argmax(prob_red_3)
    label = labelNames[idx]
    if label == labelNames[test.labels[i]]:
     aciertos = aciertos + 1
    print(label, labelNames[test.labels[i]])
    print(model3_name)
    total = total +1

  print(total,aciertos)
  print()

fallos = total - aciertos
print('---------------------------------------')
print('Total: ', total)
print('Aciertos: ', aciertos)
print('Fallos: ', fallos)
print('Accuracy: ', aciertos/total)
print(model1_name, ":",NRed1)
print(model2_name, ":",NRed2)
print(model3_name,":" ,NRed3)
print('---------------------------------------')
```

B.5 Código *ensamble* con el criterio de votación

En este apéndice se muestra un ejemplo del código utilizado para aplicar la técnica de *ensamble* con el criterio de votación, al ejecutar este código, aparece un menú desplegable para seleccionar las tres redes y la carpeta con las imágenes de pruebas.

```
target_size1 = (224, 224, 3)
target_size2 = (331, 331, 3)
target_size3 = (256, 256, 3)


import numpy as np
from tensorflow.keras.models import load_model
import imutils
from tensorflow.keras import Model
from tensorflow.keras import models
from matplotlib import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
import cv2
import tkinter as tk
from tkinter import filedialog
from tensorflow.keras.applications.inception_v3 import preprocess_input as
    prepro_inceptionV3
from tensorflow.keras.applications.vgg16 import preprocess_input as
    prepro_vgga16
import os

```

```
labelNames = ['type_000', 'type_003', 'type_004', 'type_007', 'type_008', '
    type_011', 'type_012',
'type_014', 'type_017', 'type_021', 'type_022', 'type_024', 'type_026', '
    type_027', 'type_028', 'type_030', 'type_033', 'type_036', 'type_037', '
    type_040', 'type_043', 'type_047', 'type_050', 'type_055']

NRed1 = 0
NRed2 = 0
NRed3 = 0

total = 0
aciertos= 0

root = tk.Tk()
root.withdraw()

model1_file_path = filedialog.askopenfilename(title="Seleciona la red 1 para el
     ensamble (.h5)", filetypes=[("HDF5 files", "*.h5")])
model2_file_path = filedialog.askopenfilename(title="Seleciona la red 2 para el
     ensamble (.h5)", filetypes=[("HDF5 files", "*.h5")])
model3_file_path = filedialog.askopenfilename(title="Seleciona la red 3 para el
     ensamble (.h5)", filetypes=[("HDF5 files", "*.h5")])

model1_name= os.path.basename(model1_file_path)
model2_name = os.path.basename(model2_file_path)
model3_name = os.path.basename(model3_file_path)

model_Red1 = load_model(model1_file_path)
model_Red2 = load_model(model2_file_path)
model_Red3 = load_model(model3_file_path)

folder_path = filedialog.askdirectory(title="Seleccione la carpeta de prueba")

datagen = ImageDataGenerator()

test = datagen.flow_from_directory(folder_path, classes=labelNames, class_mode=
    'categorical', shuffle=False)

mi_vector = []
posicion = 0
for i in range(len(labelNames)):
    mi_vector.insert(posicion, labelNames[i])
    posicion = posicion + 1
    mi_vector.insert(posicion, 0)
    posicion = posicion + 1

print(mi_vector)

for i in range(test.samples):
    print("etiqueta real:" + labelNames[test.labels[i]] + "\n")

for i in range(test.samples):
  img = image.imread(test.filepaths[i])

  img1 = cv2.resize(img, (224, 224))
  img2 = cv2.resize(img, (331, 331))
  img3 = cv2.resize(img, (256, 256))

  img3 = img3.astype("double") / 255.0

  img1 = np.expand_dims(img1, axis=0)
  img2 = np.expand_dims(img2, axis=0)
  img3 = np.expand_dims(img3, axis=0)

```

```
78  img1=prepro_vgga16(img1)
79  img2 = prepro_vgga16(img2)
80
81  prob_red_1 = model_Red1.predict(img1)
82  prob_red_2 = model_Red2.predict(img2)
83  prob_red_3 = model_Red3.predict(img3)
84
85
86  proba_red_1 = max(max(prob_red_1))
87  proba_red_2 = max(max(prob_red_2))
88  proba_red_3 = max(max(prob_red_3))
89
90
91  idx_red_1= np.argmax(prob_red_1)
92  label_red_1 = labelNames[idx_red_1]
93  print(model1_name)
94  print(label_red_1)
95  print (proba_red_1)
96
97  idx_red_2 = np.argmax(prob_red_2)
98  label_red_2 = labelNames[idx_red_2]
99  print(model2_name)
100 print(label_red_2)
101 print(proba_red_2)
102
103 idx_red_3 = np.argmax(prob_red_3)
104 label_red_3 = labelNames[idx_red_3]
105 print(model3_name)
106 print(label_red_3)
107 print(proba_red_3)
108
109 for e in range(len(mi_vector)):
110   if label_red_1 == mi_vector[e]:
111       mi_vector[e + 1] = mi_vector[e + 1] + 1
112
113   if label_red_2== mi_vector[e]:
114       mi_vector[e + 1] = mi_vector[e + 1] + 1
115
116   if label_red_3 == mi_vector[e]:
117       mi_vector[e + 1] = mi_vector[e + 1] + 1
118
119 valor_aux = mi_vector[1]
120 etiqueta_aux = mi_vector[0]
121 for u in range(3, len(mi_vector), 2):
122   if mi_vector[u] > valor_aux:
123       valor_aux = mi_vector[u]
124       etiqueta_aux = mi_vector[u-1]
125
126 if etiqueta_aux == labelNames[test.labels[i]]:
127   aciertos = aciertos + 1
128
129   if label_red_1 == labelNames[test.labels[i]]:
130       NRed1 = NRed1 +1
131
132   if label_red_2 == labelNames[test.labels[i]]:
133       NRed2 = NRed2 + 1
134
135   if label_red_3 == labelNames[test.labels[i]]:
136       NRed3 = NRed3 + 1
137
138
139 total = total + 1
140
141 for j in range(1, len(mi_vector), 2):
```

```
142         mi_vector[j] = 0
143
144   print("etiqueta predicha:" + etiqueta_aux + "\n")
145   print("etiqueta real:" + labelNames[test.labels[i]] + "\n")
146
147   print("total ::: aciertos \n" )
148   print(total,aciertos)
149   print()
150
151 fallos = total - aciertos
152 print('-------------------------------------')
153 print('Total: ', total)
154 print('Aciertos: ', aciertos)
155 print('Fallos: ', fallos)
156 print('Accuracy: ', aciertos/total)
157 print(model1_name, ":",NRed1)
158 print(model2_name, ":",NRed2)
159 print(model3_name,":" ,NRed3)
160 print('-------------------------------------')
```

APÉNDICE C

Resultados complementarios del proceso de entrenamiento de las redes neuronales

Con el objetivo de amenizar el cuerpo principal del proyecto, se ha delimitado esta sección dentro de los apéndices del proyecto para completar la información del entrenamiento de todas las redes entrenadas para el proyecto. Dejando en este apéndice tanto las gráficas de *"loss vs accuracy"* resultantes del entrenamiento, como la matriz de confusión por tipos resultante del proceso de prueba de las redes ya entrenadas.

C.1 Gráficas resultantes de los entrenamientos del apartado 4.3.

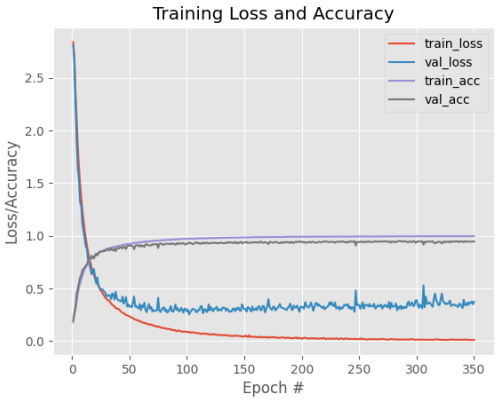


Figura C.1: *Gráfica de 'loss vs accuracy' resultante del entrenamiento de la red Polenet V.1*

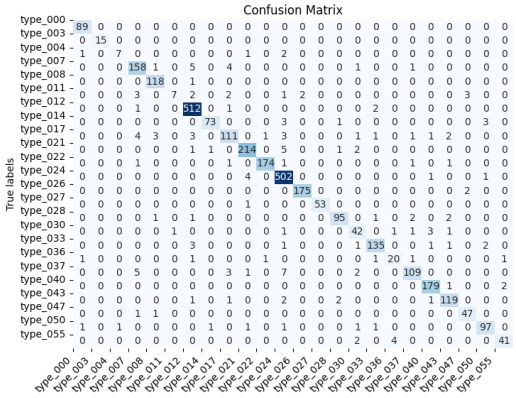


Figura C.2: *Matriz de confusión por tipos resultante del entrenamiento de la red Polenet V.1*

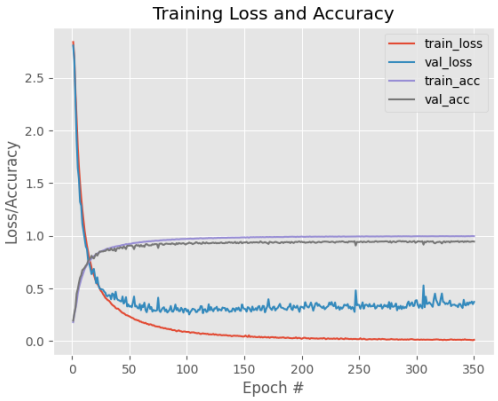


Figura C.3: *Gráfica de 'loss vs accuracy' resultante del entrenamiento de la red Prueba 1*

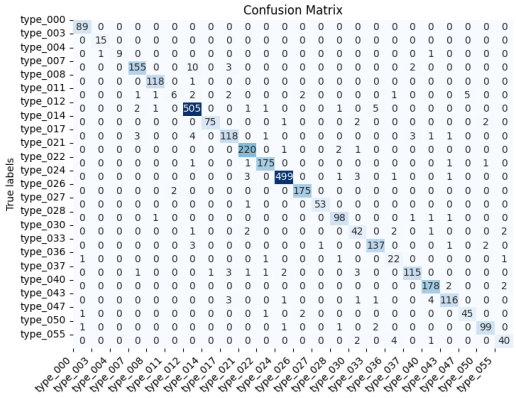


Figura C.4: *Matriz de confusión por tipos resultante del entrenamiento de la red Prueba 1*

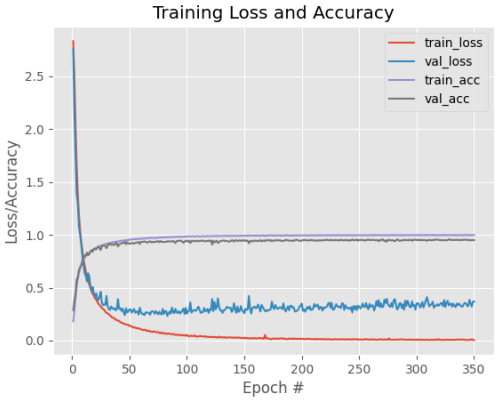


Figura C.5: *Gráfica de 'loss vs accuracy' resultante del entrenamiento de la red Prueba 2*

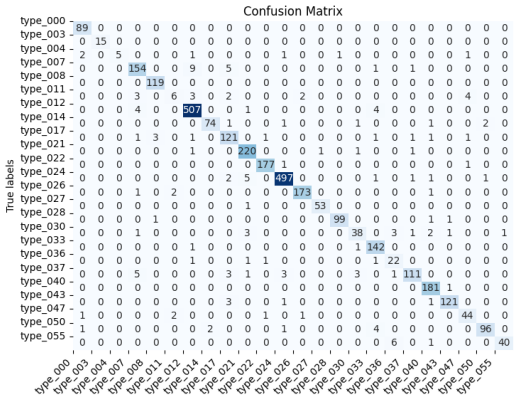


Figura C.6: *Matriz de confusión por tipos resultante del entrenamiento de la red Prueba 2*

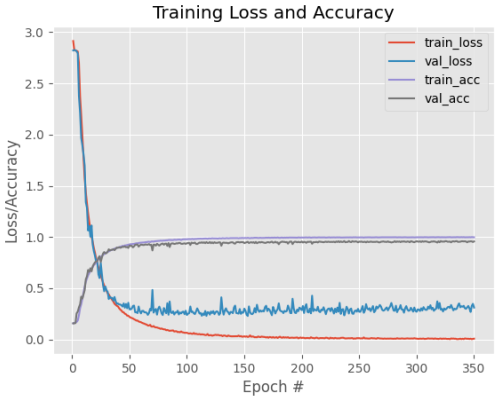


Figura C.7: *Gráfica de 'loss vs accuracy' resultante del entrenamiento de la red Prueba 3*

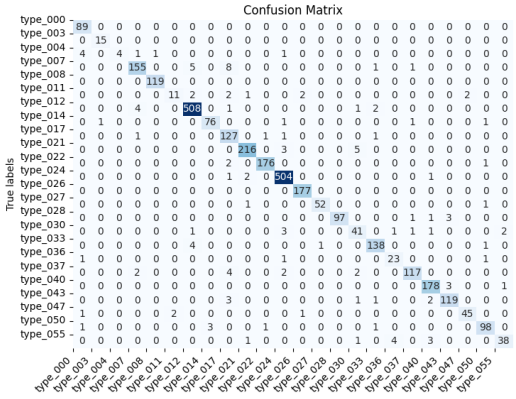


Figura C.8: *Matriz de confusión por tipos resultante del entrenamiento de la red Prueba 3*

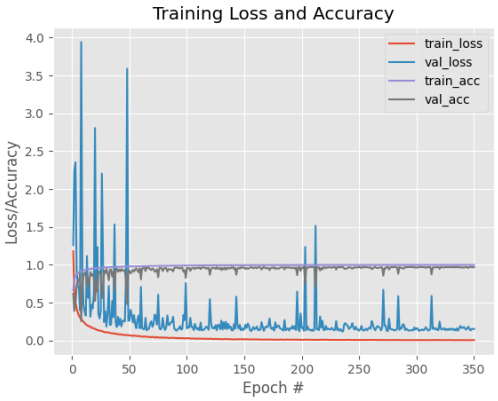


Figura C.9: *Gráfica de 'loss vs accuracy' resultante del entrenamiento de la red Prueba 4*

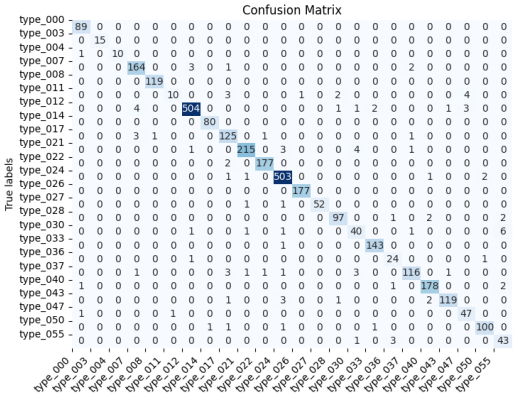


Figura C.10: *Matriz de confusión por tipos resultante del entrenamiento de la red Prueba 4*

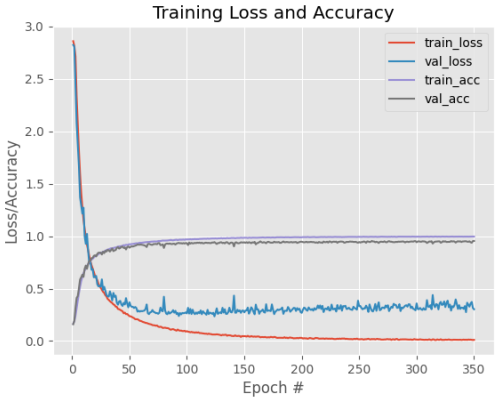


Figura C.11: *Gráfica de 'loss vs accuracy' resultante del entrenamiento de la red Prueba 5*

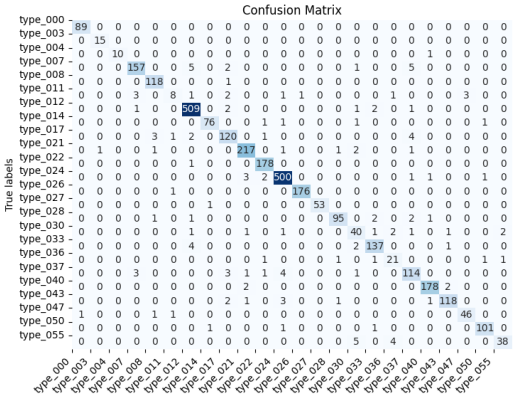


Figura C.12: *Matriz de confusión por tipos resultante del entrenamiento de la red Prueba 5*

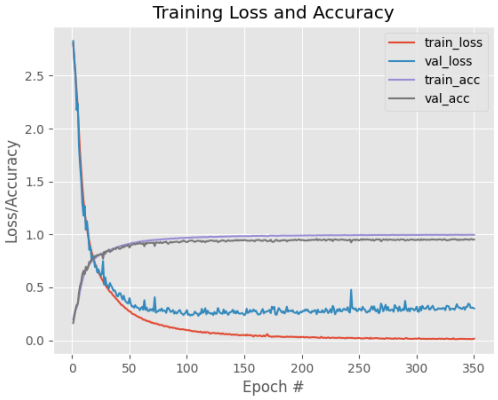


Figura C.13: *Gráfica de 'loss vs accuracy' resultante del entrenamiento de la red Prueba 6*

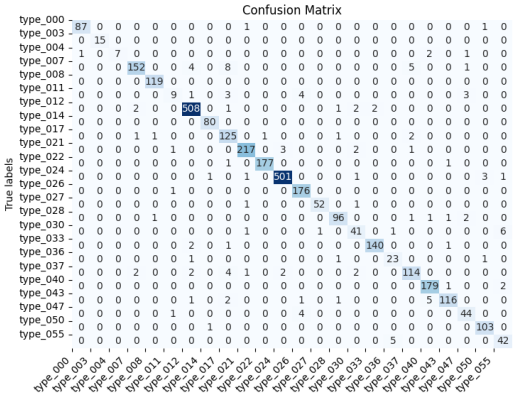


Figura C.14: *Matriz de confusión por tipos resultante del entrenamiento de la red Prueba 6*

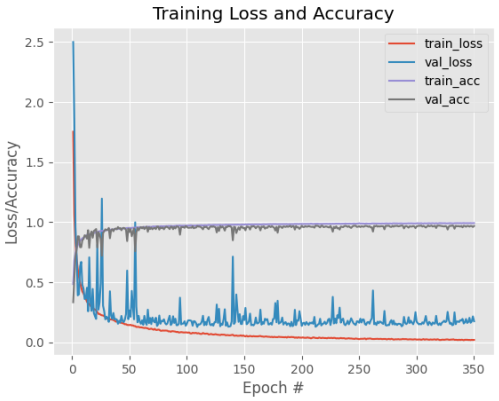


Figura C.15: *Gráfica de 'loss vs accuracy' resultante del entrenamiento de la red Prueba 7*

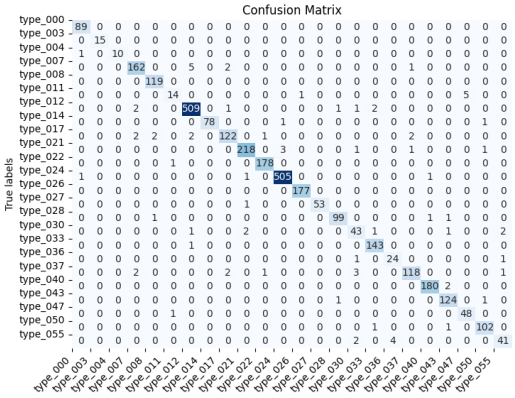


Figura C.16: *Matriz de confusión por tipos resultante del entrenamiento de la red Prueba 7*

Figura C.17: *Gráfica de 'loss vs accuracy' resultante del entrenamiento de la red Prueba 8*

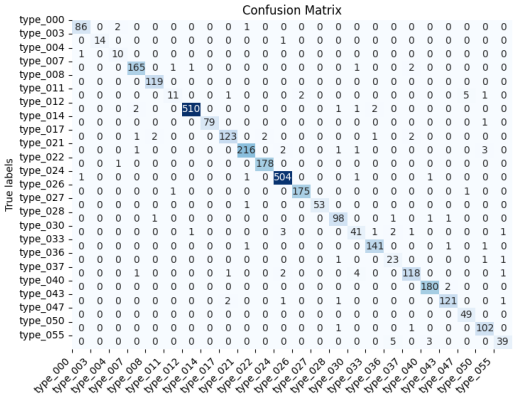


Figura C.18: *Matriz de confusión por tipos resultante del entrenamiento de la red Prueba 8*

Figura C.19: *Gráfica de 'loss vs accuracy' resultante del entrenamiento de la red Prueba 9*

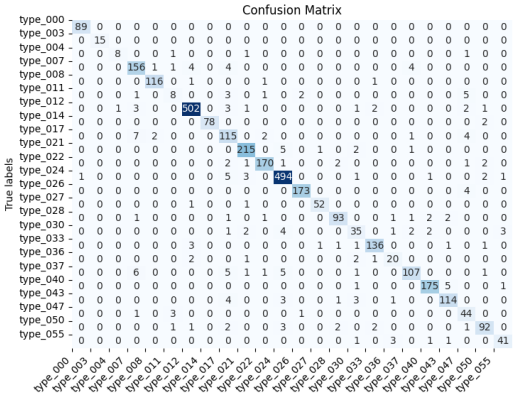


Figura C.20: *Matriz de confusión por tipos resultante del entrenamiento de la red Prueba 9*

APÉNDICE D

Muestras de mieles obtenidas del microscopio óptico

En este apéndice se mostrarán algunas de las muestras de miel empleadas posteriormente para crear el dataset propio. Con esto, se pretende mostrar también la dificultad del proceso de detección y clasificación en este contexto.

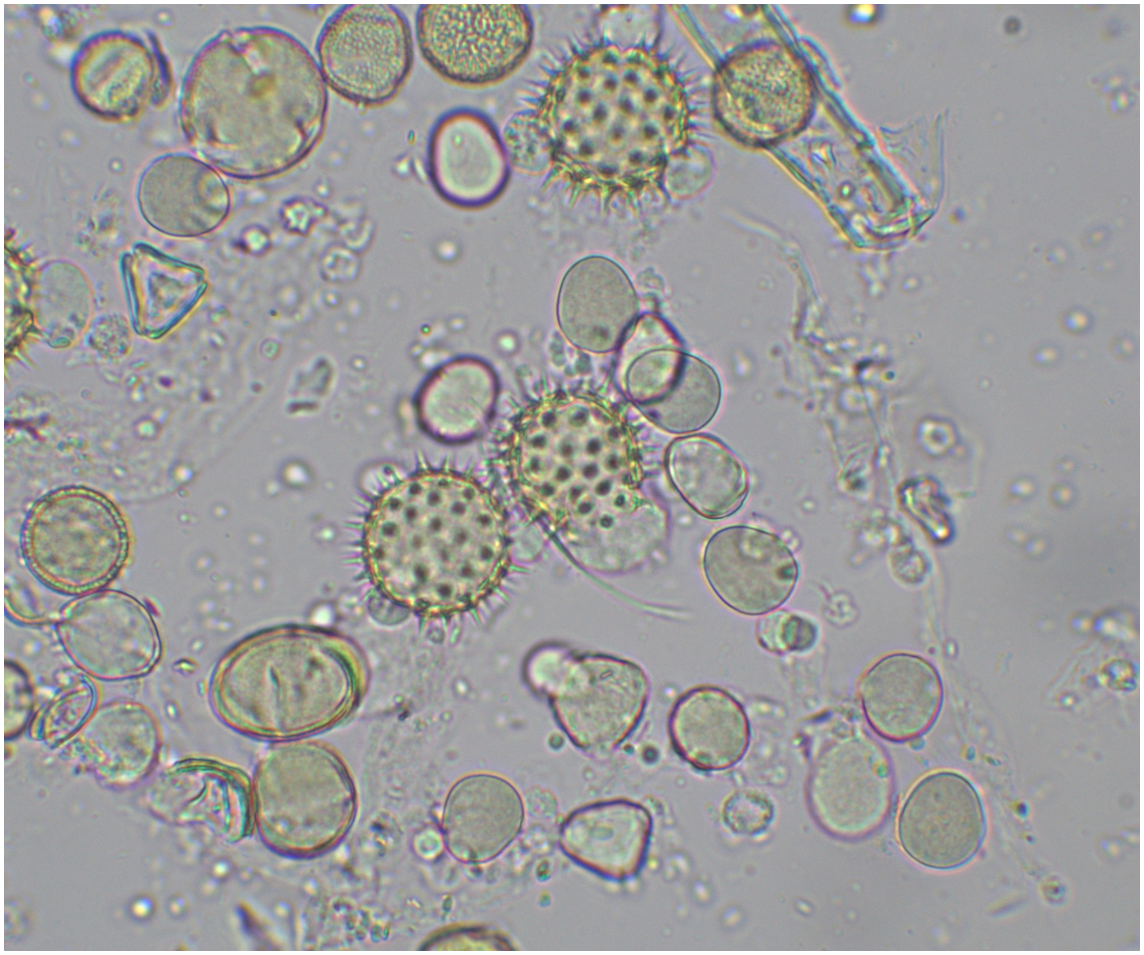


Figura D.1: *Imagen de una muestra de miel de milflores obtenida con el microscopio óptico.*

Figura D.2: *Imagen de una muestra de miel de azahar obtenida con el microscopio óptico.*

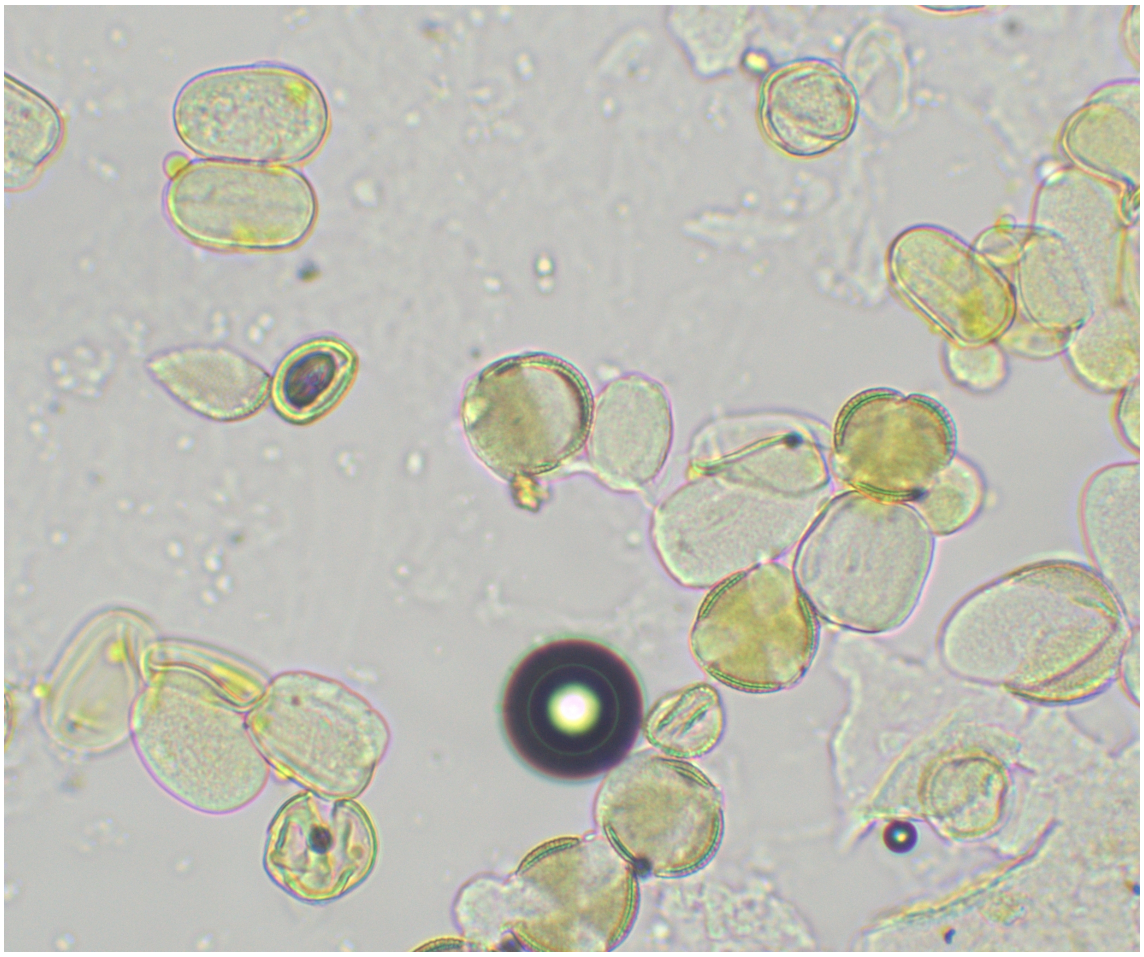


Figura D.3: *Imagen de una muestra de miel de brassica obtenida con el microscopio óptico.*