UNIVERSIDAD AUTÓNOMA DE SINALOA
FACULTAD DE INFORMÁTICA CULIACÁN - FACULTAD DE CIENCIAS DE LA TIERRA Y EL ESPACIO
MAESTRÍA EN CIENCIAS DE LA INFORMACIÓN

Uso de Redes Neuronales Profundas para la Identificación de Plantas a partir de Imágenes

Presenta:
Lic. Zuriel Ernesto Morales Casas

Directores:
Dr. Inés Fernando Vega López
Dr. Arturo Yee Rendón

10 de febrero de 2021

Agradecimientos

Al Consejo Nacional de Ciencia y Tecnología

A la Universidad Autónoma de Sinaloa

Al Posgrado en Ciencias de la Información

A la Facultad de Informática Culiacán

Al Parque de Innovación Tecnológica

Y especialmente al Fondo Sectorial CONACyT INEGI con clave 291772 CONACyT-INEGI 2017 01.

Índice general

1 Introducción **1**

2 Marco teórico y estado del arte **4**

2.1 Clasificación automatizada de plantas tradicional 4

2.2 Clasificación de imágenes 5

2.3 Redes Neuronales Artificiales 6

2.3.1 Estructura 6

2.3.2 Tensores 8

2.3.3 Entrenamiento 8

2.3.4 Parámetros e Hiper-parámetros 9

2.3.5 Tasa de aprendizaje 10

2.3.6 Transferencia de aprendizaje (*Transfer-learning*) 10

2.3.7 Perceptrón 11

2.3.8 Algoritmos de optimización 11

2.4 Redes Neuronales Convolucionales 14

2.4.1 Capa Convolucional 16

2.4.2 Capa de agregación 17

2.4.3 Capa densa 18

2.4.4 Problemas comunes de redes neuronales 18

2.4.5 Funciones de activación 19

2.5 Uso de RNC en tareas de clasificación de imágenes 20

3 Metodología **32**

3.1 Conjunto de datos 32
3.2 Herramientas 34
3.2.1 Herramienta de recortes 35
3.2.2 Arquitecturas 35

4 Pruebas y análisis de resultados **49**
4.1 Descripción de experimentos 49
4.1.1 Entrenamiento general, primera ronda 50
4.1.2 Entrenamiento general, segunda ronda 51
4.1.3 Entrenamiento general, tercera ronda 52
4.1.4 Entrenamiento general, cuarta ronda 53
4.1.5 Entrenamiento por órgano 54
4.2 Análisis de resultados 55
4.2.1 Entrenamientos generales 55
4.2.2 Entrenamiento por órgano 60

5 Conclusiones **65**

A Código **73**
A.0.1 Script de entrenamiento principal 73
A.0.2 Generador de lotes 77
A.0.3 Optimizadores 78
A.0.4 Sumarizador de modelos 79

B Entrenamiento de modelos **80**

C Desempeños por clase, entrenamientos generales **122**

D Desempeños por clase, entrenamientos por órgano **137**

Índice de figuras

2.1 Representación visual de una Red Neuronal Artificial. 7
2.2 Diagrama de un perceptrón que implementa la función lógica OR. 11
2.3 Ilustración del campo receptivo. 16
2.4 Operación de convolución. 17
2.5 Operación de agregación máxima con un kernel de 2×2 y paso de 2. . . . 18

3.1 Ejemplo de alta variabilidad intraclase. 33
3.2 Ejemplo de baja variabilidad interclase. 34
3.3 Interfaz de PitCrop. 36
3.4 Ejemplo de recorte. 36
3.5 Módulo Inception clásico 39
3.6 Módulo Inception v2 40
3.7 Factorización de capas convolucionales en Inception v2 41
3.8 Diagrama de una conexión residual 42
3.9 Módulo Inception-ResNet-B 43
3.10 Módulo Xception simplificado 44
3.11 Bloque denso de cuatro capas 45

4.1 Ranking de puntajes Top-1 y Top-5 de arquitecturas, primera ronda. 56
4.2 Ranking de puntajes Top-1 y Top-5 de arquitecturas, segunda ronda. 56
4.3 Ranking de puntajes Top-1 y Top-5 de arquitecturas, tercera ronda. 57
4.4 Ranking de puntajes Top-1 y Top-5 de arquitecturas, cuarta ronda. 57
4.5 Ranking de puntajes Top-1 y Top-5 de arquitecturas por órgano, flores. . . . 62
4.6 Ranking de puntajes Top-1 y Top-5 de arquitecturas por órgano, frutos. . . . 63

4.7 Ranking de puntajes Top-1 y Top-5 de arquitecturas por órgano, hojas. . . . 63
4.8 Ranking de puntajes Top-1 y Top-5 de arquitecturas por órgano, tallos. . . . 64

Índice de tablas

2.1 Comparativa de desempeño en el dataset MalayaKew con clasificadores distintos. 22
2.2 Desempeño del equipo ECOUAN en el reto de clasificación LifeCLEF 2015. 23
2.3 Desempeño del clasificador propuesto por Lee et al. en el reto ImageNet 2016. 24
2.4 Evaluación de resultados ordenados por puntaje MAP oficial para el reto LifeCLEF 2016. 26
2.5 Resultados de LeafNet en datasets Flavia, Foliage y LeafSnap 28
2.6 Muestra del incremento en precisión lograda con distintos enfoques sobre conjuntos de datos de plantas conocidos. 31

3.1 Arquitectura de VGG19. 38
3.2 Arquitecturas de Densenet-121, -169 y -201. 46
3.3 Arquitectura de MobileNet 48

4.1 Resultados finales de entrenamiento general, primera ronda 51
4.2 Resultados finales de entrenamiento general, segunda ronda. 52
4.3 Resultados finales de entrenamiento general, tercera ronda. 53
4.4 Resultados finales de entrenamiento general, cuarta ronda. 54
4.5 Cardinalidad de los subconjuntos de datos para cada órgano individual . . . 55

C.1 Desempeño Top-1 por clase, primera ronda. 125
C.2 Desempeño Top-1 por clase, segunda ronda. 129
C.3 Desempeño Top-1 por clase, tercera ronda. 132

C.4 Desempeño Top-1 por clase, cuarta ronda. 136

D.1 Desempeño Top-1 por clase, primera ronda por órgano, flores. 139
D.2 Desempeño Top-1 por clase, primera ronda por órgano, frutos. 139
D.3 Desempeño Top-1 por clase, primera ronda por órgano, hojas. 140
D.4 Desempeño Top-1 por clase, primera ronda por órgano, tallos. 140

Abstract

Nowadays, the field of computer vision is dominated by Convolutional Neural Networks (CNN), which provide state of the art performance in detection and classification of objects from digital images. This has resulted in CNNs being used in a wide variety of fields, from medicine to robotics. One task where CNNs have gained significant popularity is the automatic detection and classification of plants, which is important in fields such as agriculture and botany. In this work, we make use of Deep Learning techniques, specifically Convolutional Neural Networks, to produce a predictive model that can identify plants found within the geography of México. We focused on working with existing and well-known architectures, such as VGGNet and Inception, that have had good performance in image classification tasks in the past. We measured these architectures' performance on the task of plant classification by training them on a custom dataset, which is comprised of images of plants downloaded from the iNaturalist website, as well as photos taken by a group of expert botanists. The final product of this work is a trained CNN capable of making fairly accurate predictions of a plant's species by taking its photograph as input.

Resumen

Hoy en día, el campo de la visión por computadora se encuentra dominado por las Redes Neuronales Convolucionales (RNC), las cuales proveen desempeño de vanguardia en la detección y clasificación de objetos a partir de imágenes. Como resultado de esto, las RNC son utilizadas en una amplia variedad de disciplinas, desde la medicina hasta la robótica. Una de las tareas donde las RNC se han popularizado es la detección y clasificación automática de plantas, la cual es de gran importancia en campos tales como la agricultura y la botánica. En este trabajo, hacemos uso de técnicas de Aprendizaje Profundo, específicamente Redes Neuronales Convolucionales, para producir un modelo predictivo capaz de identificar plantas que se encuentran de manera silvestre en México. Nos centramos en trabajar con arquitecturas existentes y bien conocidas, tales como VGGNet e Inception, que hayan tenido un buen desempeño en otras tareas de clasificación. Medimos el desempeño de estas arquitecturas en la tarea de clasificación de plantas entrenándolas con un conjunto de datos original, el cual se compone de imágenes de plantas descargadas del sitio web iNaturalist, así como fotografías tomadas por un grupo de botánicos expertos. El producto final de este trabajo es una RNC capaz de realizar predicciones relativamente precisas de la especie de una planta al recibir como entrada una fotografía de la misma.

Capítulo 1

Introducción

La taxonomía de plantas ha tomado un lugar de gran prioridad en el ámbito de la visión por computadora debido a su alta importancia para el estudio de la biodiversidad y la conservación ambiental [1]. Sin embargo, la tarea de identificar plantas manualmente es un proceso laborioso y tardado. Esto se debe no solo a la enorme variedad de plantas que existen, sino también a la alta diversidad que existe entre las propiedades morfológicas de las mismas. Como consecuencia de esto, existe una creciente necesidad de sistemas de software que apoyen a las tareas de identificación de plantas de manera automatizada [2].

Recientemente, las técnicas basadas en Aprendizaje Profundo han sobresalido en tareas de identificación de objetos a partir de imágenes digitales, particularmente las Redes Neuronales Convolucionales (RNC), las cuales presentan resultados notables en diversos retos de clasificación de imágenes. Debido a esto, las RNC han tomado un rol central en los enfoques modernos de detección y clasificación de plantas a partir de imágenes. Los modelos basados en Aprendizaje Profundo han superado de manera consistente a otras técnicas en retos de detección y clasificación de plantas, lo cual ha quedado demostrado por proyectos tales como Bluefield [3] y AlexNet [4] en las ediciones 2016 y 2017, respectivamente, del reto PlantCLEF (uno de los retos más importantes en el área de identificación de plantas a partir de imágenes). Ambos enfoques están basados en RNCs.

Sin embargo, a pesar de que las RNCs han traído consigo avances en el campo de la clasificación automática de imágenes, aún se encuentran lejos de alcanzar la capacidad de reconocimiento de los sistemas de visión humana. Esto afecta, por consecuencia, al

problema de clasificación automática de plantas, el cual presenta varios retos adicionales, entre los que encontramos principalmente:

- La gran cantidad de especies distintas de plantas que existen. En México existen 25,008 especies de plantas vasculares conocidas, pero se estima que puede haber hasta 31,100 [5].

- El alto nivel de variabilidad intraespecie, lo cual se refiere a las diferencias que se encuentran entre dos ejemplares de una misma especie.

- Las similitudes interespecie, la cual se refiere a las similitudes que se encuentran entre ejemplares de especies cercanamente relacionadas pero distintas y que dificultan la discriminación entre estas.

- Las variaciones que se producen durante el proceso de adquisición de imágenes, lo cual se refiere a las diferencias en condiciones que se dan al tomar fotografías de las plantas, tales como iluminación, enfoque, ángulo, entre otros, y que introducen incertidumbre en los algoritmos.

Como consecuencia de esto, la detección automática de plantas se ha convertido en uno de los problemas de mayor interés en el ámbito de aprendizaje profundo y visión por computadora. Al mismo tiempo, los investigadores de Aprendizaje Profundo constantemente proponen nuevas y mejores arquitecturas y técnicas, lo cual presenta una ventana de oportunidad para evaluar el desempeño estos nuevos desarrollos en problemas de clasificación de plantas. Esto es importante debido a que permite tomar mejores decisiones al diseñar sistemas que apoyen a la clasificación automática de plantas.

La clasificación de especies de plantas es una tarea muy importante para los campos de la agricultura y botánica, así como para tareas de protección y conservación de especies, entre otras. Sin embargo, incluso para los botánicos expertos, el identificar correctamente a una especie a partir de imágenes digitales constituye también una tarea difícil. Al mismo tiempo, las RNCs han logrado mejorar de manera considerable el desempeño de los modelos de clasificación de imágenes actuales, aún con sus limitaciones, y actualmente existen

proyectos orientados a la identificación de plantas en regiones geográficas específicas, tales como WTPlant [6], el cual identifica especies que se encuentran en la región de Hawaii.

En México, aún existe una necesidad de sistemas que apoyen a la identificación de plantas. En 2017, la agencia mexicana INEGI (Instituto Nacional de Estadística y Geografía) lanzó una convocatoria en la que se destaca, para este trabajo, la demanda de una aplicación para reconocimiento de especies vegetales y características de los tipos de vegetación de México a partir de fotografías tomadas con dispositivos móviles, citando como principal motivo de interés la necesidad de una actualización práctica para clasificar los tipos de vegetación existentes en el territorio nacional.

El resto de este trabajo se organiza de la forma siguiente: en el Capítulo 2 de esta tesis se describe qué son las Redes Neuronales Convolucionales – su estructura y funcionamiento general –, así como conceptos básicos de Aprendizaje Profundo. También se hace referencia a los métodos de clasificación de plantas tradicionales, en el Capítulo 3 se detalla la metodología utilizada. Esto incluye las arquitecturas elegidas para los experimentos y las herramientas que se desarrollaron para auxiliar el proceso. También se describe el conjunto de datos utilizado, en el Capítulo 4 se presentan los resultados obtenidos a partir de los experimentos realizados, así como un análisis y discusión de los mismos, y finalmente en el Capítulo 5 se presentan las conclusiones y el trabajo a futuro.

Capítulo 2

Marco teórico y estado del arte

En las secciones siguientes se describe el proceso general de clasificación de plantas. También se habla de manera breve acerca de los métodos de clasificación automatizados que preceden a aquellos basados en aprendizaje máquina. Adicionalmente, se describe la estructura y funcionamiento de las redes neuronales artificiales, así como las variantes utilizadas en este proyecto, conocidas como redes neuronales convolucionales. Finalmente, se habla sobre los trabajos de clasificación automatizada de plantas realizados previamente por diversos investigadores.

2.1. Clasificación automatizada de plantas tradicional

Antes de entrar al tema de aprendizaje profundo para clasificación de plantas, es importante mencionar los métodos de clasificación tradicionales. Los primeros esfuerzos por automatizar la tarea de clasificación de plantas (aquellos que preceden al aprendizaje profundo) se centraban en identificar y extraer características generales en las formas de las plantas. Estos primeros esfuerzos se orientaban a la clasificación usando principalmente las hojas de las plantas, esto con la finalidad de limitar la complejidad del problema, y también porque las hojas son fácilmente accesibles y poseen características altamente discriminatorias de la especie [2].

Existen numerosos estudios orientados a la detección de plantas utilizando técnicas tradicionales de aprendizaje máquina. El proyecto LeafSnap [7] tiene como propósito identi-

ficar especies de árboles a partir de fotografías de sus hojas. Para esto hace uso de técnicas de segmentación y extracción de características, en concreto medidas de curvatura de las hojas. Así mismo, ApLeaf [8] hace uso de extracción de características (principalmente la forma de las hojas) para identificar especies vegetales de la zona del mediterráneo francés a partir de fotografías de hojas.

2.2. Clasificación de imágenes

En términos generales, el proceso de clasificación de imágenes puede dividirse en cuatro fases [9]: adquisición de imágenes, preprocesamiento de imágenes, extracción y descripción de características, y clasificación.

Durante la primera fase, adquisición de imágenes, se obtienen las imágenes de las plantas que se van a identificar. Esto puede implicar tomar fotografías o descargar imágenes de internet.

La siguiente fase, preprocesamiento de imágenes, consiste en preparar la imagen, eliminando ruido y enfatizando características de la imagen que sean de utilidad para el proceso de clasificación (por ejemplo, al segmentar las imágenes para eliminar el fondo y conservar áreas de interés).

Después, durante la fase de extracción de características, se toman medidas de áreas de interés en la imagen, para después almacenar esta información en vectores denominados descriptores. Existe una amplia gama de descriptores, los cuales se basan en forma, color, y textura, entre otros.

Finalmente, en la fase de clasificación, las características extraídas se concatenan en un vector, el cual es posteriormente clasificado utilizando algoritmos como K vecinos más cercanos (*K-Nearest Neighbors*), SVMs (*Support Vector Machines*), Árboles de decisión o RNCs, por mencionar algunos.

De estas cuatro fases, la más laboriosa es la extracción y descripción de características. Los algoritmos utilizados para extraer descriptores de características se denominan detectores de características, e inicialmente su desarrollo se realizaba de forma manual – esto es, los investigadores diseñaban manualmente los detectores basándose en las características

de los objetos que estaban interesados en clasificar. Esto traía como resultado un modelo altamente específico, orientado únicamente, por ejemplo, a la detección de flores u hojas en ciertas especies predefinidas [10]. Por su parte, los descriptores de características son vectores que representan propiedades de las plantas u órganos. Existe una amplia gama de descriptores, diseñados específicamente para clasificar diversas especies de plantas, los cuales se basan en forma, color, textura, tamaño, así como diversas combinaciones de los mismos. Así mismo, existen múltiples descriptores "genéricos", tales como SIFT (*Scale-invariant feature transform*), SURF (*Speeded-up robust features*) y HOG (*Histogram of oriented gradients*), los cuales presentan una ventaja sobre los enfoques que se centran en características morfológicas específicas de las plantas, al permitir un mayor grado de generalización [9].

2.3. Redes Neuronales Artificiales

Las Redes Neuronales Artificiales (RNA) se basan vagamente en la estructura cerebral de los animales. La idea tuvo sus inicios con la investigación de Warren McCulloch y Walter Pitts, quienes en 1943 presentaron un modelo computacional simplificado de como las neuronas biológicas trabajan juntas para realizar cálculos complejos, utilizando lógica proposicional [11]. Esta fue la primera arquitectura de Red Neuronal Artificial. McCulloch y Pitts presentaron un modelo simple de neurona artificial, el cual recibía como entrada una serie de señales binarias (encendido/apagado), y se activaba (producía su propia señal) cuando un cierto número de señales de entrada se activaban.

2.3.1. Estructura

Como su nombre lo indica, las RNA están conformadas por colecciones de unidades o nodos, denominados neuronas artificiales. Estas neuronas se estructuran en grupos llamados capas. La primera capa de una red neuronal se denomina capa de entrada, puesto que esta es la capa que se encarga de recibir los datos que procesará la red. La última capa se denomina capa de salida, y constituye la respuesta de la red para una entrada determinada. Finalmente, todas las capas intermedias se denominan capas ocultas.

Las neuronas en una capa procesan una serie de datos de entrada recibidos de la capa anterior, generando un vector de activaciones. Posteriormente, este vector de activaciones es enviado a la capa siguiente para ser procesado de una manera similar. La estructura general de una RNA se puede apreciar en la Figura 2.1.

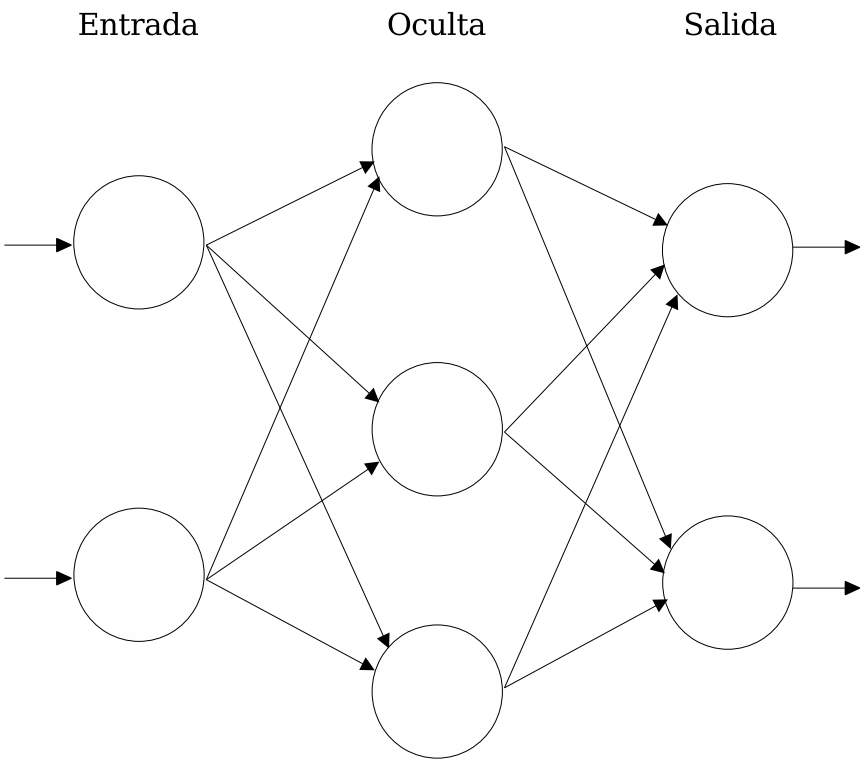

Figura 2.1: Representación visual de una Red Neuronal Artificial.

En una RNA tradicional, todas las neuronas de una capa determinada se interconectan con todas las neuronas de la capas anterior y siguiente. Una neurona recibe como información de entrada la suma de todas las activaciones en las neuronas de la capa anterior, multiplicadas por el valor de su conexión a cada neurona individual. Posteriormente, las neuronas aplican una función, denominada función de activación, al valor resultante. El resultado de esta función representa la activación de una neurona, y es la información que se enviará a las neuronas de la capa siguiente para su procesamiento. Las conexiones entre

neuronas poseen una propiedad denominada “peso”, la cual corresponde a la intensidad de la conexión entre dos neuronas. Mientras mayor sea la intensidad de la conexión de una neurona con otra, la activación de la primera tendrá una mayor influencia sobre la segunda.

2.3.2. Tensores

Las estructuras que representan los datos de entrada y salida de las capas convolucionales de una red neuronal se denominan tensores. Un tensor, a grandes rasgos, es una generalización de los conceptos de vectores y matrices, que puede conformarse de n dimensiones. El número de dimensiones que un tensor posee es dictado por su orden. Más específicamente, un tensor de una dimensión (vector) es un tensor de orden uno, un tensor de dos dimensiones (matriz) es un tensor de orden dos, un tensor de tres dimensiones es un tensor de orden tres, y así sucesivamente. Los valores escalares son tensores de orden cero.

2.3.3. Entrenamiento

Existen diversos paradigmas de entrenamiento de modelos de aprendizaje máquina, principalmente los denominados aprendizaje supervisado y no supervisado. Informalmente, el aprendizaje supervisado puede definirse como aquel donde el modelo aprende a producir respuestas correctas para datos no antes vistos a partir de datos cuya respuesta ya conocemos. Por otro lado, el entrenamiento no supervisado intenta encontrar estructuras en los datos, sin indicarle al modelo qué buscar de forma explícita. En el caso del problema de clasificación, se hace uso de aprendizaje supervisado para entrenar a las redes. Dicho proceso consiste en entregar a la red ejemplos de información cuya respuesta deseada conocemos de antemano. Posteriormente, se realiza una comparación de la respuesta de la red y la correcta, para obtener un valor de pérdida o error (*loss*). Finalmente, los valores de los pesos de la red se modifican con el propósito de reducir el valor de error. Una de las funciones de error más populares en problemas de clasificación es Entropía Categórica Cruzada, o *Categorical Crossentropy*, cuya fórmula es la siguiente:

$$L(y, \hat{y}) = -\sum_{j=0}^{M}\sum_{i=0}^{N}(y_{ij} * \log(\hat{y}_{ij}))$$

Esta función se utiliza en problemas de clasificación donde solo una respuesta puede ser correcta, lo cual la vuelve ideal para el problema de clasificación de plantas, donde cada espécimen individual solo puede pertenecer a una sola especie.

Por su parte, el proceso de entrenamiento consiste en calcular las gradientes de el valor de error con respecto de cada uno de los pesos en la red, para posteriormente realizar ajustes en la dirección opuesta a la gradiente y así reducir el valor de error. El algoritmo que hace esto se conoce como Descenso de Gradiente, debido a que la dirección en la que se alteran los pesos de la red es hacia "abajo", o en descenso, y fue popularizado por Paul Werbos en 1974 [12]. Este algoritmo hace uso de propagación hacia atrás (*backpropagation*), para calcular las gradientes del valor de error con respecto a los pesos en la red, partiendo desde la última capa, o capa de salida, y culminando en la primera capa, o capa de entrada.

El paso de información a través de la red desde la capa de salida hacia la capa de entrada (en reversa) se denomina "pase hacia atrás" o *backward pass*, mientras que el paso de la información desde la capa de entrada hasta la capa de salida para generar predicciones se denomina "pase hacia adelante", o *forward pass*. Cada vez que se han calculado los gradientes de todo el conjunto de datos, se dice que se ha completado una iteración. Usualmente es necesario entrenar durante múltiples iteraciones antes de que el modelo alcance el estado de convergencia.

En aprendizaje máquina, se dice que un modelo ha alcanzado un estado de convergencia cuando su valor de pérdida se acerca a cero como consecuencia de la configuración de sus pesos, y los cambios al valor de pérdida subsecuentes se vuelven demasiado pequeños.

2.3.4. Parámetros e Hiper-parámetros

En el ámbito del aprendizaje máquina, se conoce como parámetros a los valores internos que poseen los componentes de un modelo. En aprendizaje profundo, estos pueden ser los valores de los pesos de las capas densas, los filtros de una capa convolucional, etc. Generalmente, estos no son definidos manualmente, sino que son aprendidos automáticamente por el modelo durante la fase de entrenamiento. Por otro lado, los hiper-parámetros son variables externas al modelo que gobiernan el proceso de entrenamiento y, por lo tanto, no pueden ser aprendidos. Ejemplos de hiper-parámetros son la tasa de aprendizaje (descrito

en la siguiente sub-sección), el número de épocas que durará el entrenamiento, el algoritmo de optimización y sus respectivos parámetros, entre otros.

2.3.5. Tasa de aprendizaje

La tasa de aprendizaje, también conocida como *learning rate*, es un hiper-parámetro propio de los métodos de entrenamiento basados en la propagación hacia atrás. Una vez que se calculan los gradientes de una capa durante el proceso de entrenamiento, los valores de estos son multiplicados por la tasa de aprendizaje. De esta forma, la tasa de aprendizaje controla el tamaño de los ajustes que se hacen a los parámetros de las capas durante la propagación hacia atrás. Un valor demasiado bajo para este hiper-parámetro resultará en cambios demasiado pequeños a los pesos de una red neuronal, lo cual puede causar que la red se quede atrapada en un mínimo local de su función de pérdida. Por otro lado, un valor demasiado grande resultará en cambios muy grandes a los pesos, lo cual puede causar inestabilidad durante el proceso de entrenamiento, dificultando la convergencia del modelo.

2.3.6. Transferencia de aprendizaje (*Transfer-learning*)

Algunas secciones de este documento hacen referencia a trabajos de investigación que utilizan una técnica conocida como transferencia de aprendizaje o *transfer-learning*. Esta técnica consiste en tomar un modelo entrenado para una tarea específica (*e.g.,* clasificación de vehículos automotrices de uso común civil), y re-entrenarlo en una tarea diferente, pero relacionada con la tarea original para la cual se entrenó el modelo (*e.g.,* clasificación de vehículos de una marca automotriz particular). Esta técnica permite a los modelos converger más rápido, puesto que no requieren aprender filtros desde cero.

En el ejemplo de clasificación de automóviles mencionado anteriormente, el modelo clasificador ya posee filtros especializados en detección de características particulares de un automóvil (llantas, parabrisas, cofre, etc.), los cuales aprendió cuando fue entrenado para la primera tarea. Cuando el modelo es entrenado para clasificación de automóviles de una marca particular, el mismo es capaz de reutilizar los filtros especializados que ya posee, simplemente ajustándolos para clasificación de muestras del conjunto de datos nuevo. Este

enfoque de transferencia de aprendizaje se conoce como afinación o *fine-tuning*.

2.3.7. Perceptrón

El Perceptrón es una de las arquitecturas de RNA más populares. Fue propuesto por el psicólogo Frank Rosenblatt en 1957 [13]. Este se basa en un tipo de neurona artificial llamada *Linear Threshold Unit* (LTU), en la que las entradas ya no son binarias, sino números reales, y cada conexión de entrada está asociada con un peso. Las LTUs calculan la sumatoria de sus entradas, y posteriormente aplica una función de activación a dicha sumatoria. El resultado de esta función corresponde a la señal de salida. La Figura 2.2 presenta un ejemplo de un perceptrón que implementa la operación lógica OR, la cual retorna un valor booleano verdadero si cualquiera de sus entradas es verdadera. Este perceptrón multiplica los valores de entrada por uno; esto producirá un valor de salida positivo si cualquiera de los valores de entrada son uno, y cero solo si ambos son cero.

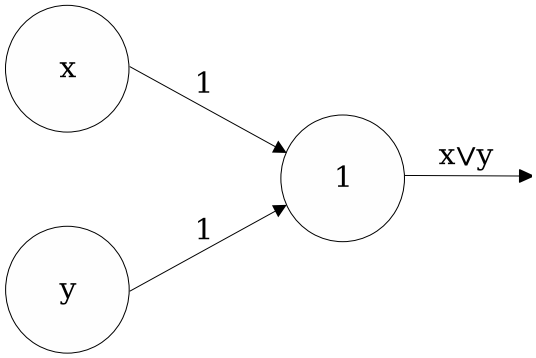

Figura 2.2: Diagrama de un perceptrón que implementa la función lógica OR.

2.3.8. Algoritmos de optimización

Los algoritmos de optimización son el mecanismo mediante el cual es posible entrenar modelos de aprendizaje máquina. En esencia, son algoritmos cuyo objetivo es minimizar una función de pérdida. La biblioteca utilizada en este trabajo, Keras, provee múltiples algoritmos de optimización, los cuales se describen a continuación.

Descenso de Gradiente

Anteriormente se habló acerca del descenso de gradiente, el cuál es un algoritmo iterativo que calcula el valor de los gradientes de la función de error $J(\theta)$ de un modelo de aprendizaje profundo, para posteriormente modificar los pesos θ del modelo con el objetivo de reducir este valor de error:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$$

El término η corresponde a la tasa de aprendizaje. En el descenso de gradiente clásico, el ajuste a los pesos se obtiene a partir del promedio de los gradientes del valor de error para cada muestra en el conjunto de datos. Esto significa que deben obtenerse los gradientes para cada muestra antes de actualizar los pesos, lo cual se vuelve lento para conjuntos de datos grandes.

Descenso de Gradiente Estocástico

El descenso de gradiente estocástico es una variante del algoritmo de descenso de gradiente que calcula y aplica cambios a los pesos del modelo por cada muestra x^i y etiqueta y^i del conjunto de datos:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^i; y^i)$$

El algoritmo continua empleando todos los datos de entrenamiento para calcular los gradientes. Esto significa que los pesos se actualizan con mayor frecuencia durante una iteración. El resultado es una función de optimización relativamente inestable, ya que los ajustes a los pesos son imprecisos.

Descenso de Gradiente Estocástico por Mini-lotes

Esta variante del descenso de gradiente estocástico calcula los gradientes de un subconjunto relativamente pequeño del conjunto de datos, denominado lote o *batch*, usando el promedio de estos gradientes. se conoce como descenso de gradiente estocástico por mini-lotes:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{i:i+n}; y^{i:i+n})$$

Esta variante del algoritmo es capaz de alcanzar un estado de convergencia con mayor rapidez que la versión clásica, y resulta más estable que la variante estocástica.

Adagrad

Adagrad, o *Adaptive Gradient* es una variante del algoritmo de descenso de gradiente estocástico. Este algoritmo mantiene una tasa de aprendizaje para cada peso de la red. Las actualizaciones a los pesos están dadas por:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$$

Donde $g_{t,i}$ es la derivada parcial de la función de error con respecto al parámetro θ_i en un momento t, y G es una matriz diagonal donde cada elemento diagonal ii es la suma de los cuadrados de los gradientes θ_i en un momento t, y ϵ es un término de suavizado.

La elección de tasa de aprendizaje en este algoritmo se vuelve menos relevante como consecuencia de su adaptabilidad, lo cual representa una ventaja. No obstante, sí los gradientes de los pesos son demasiado grandes y frecuentes, es posible terminar con una tasa de aprendizaje demasiado pequeña para ser significativa en muy poco tiempo.

Adadelta

Adadelta es una variante de Adagrad. Su diferencia radica en la estrategia de escalamiento de la tasa de aprendizaje: en lugar de acumular todos los gradientes, Adadelta solo mantiene un número fijo de gradientes previos, en la forma de un promedio móvil definido por la siguiente ecuación:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1 - \gamma) g_t^2$$

Donde g_t^2 representa los gradientes cuadrados de los pesos en un momento t y γ es un término de escalamiento. Finalmente, la función de optimización toma la siguiente forma:

$$\Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \cdot g_t$$

RMSProp

RMSProp, o *Root Mean Squares*, es una variante de Adagrad altamente similar a Adadelta. En contraste con Adadelta, RMSProp utiliza el promedio móvil de los gradientes cuadrados para escalar la tasa de aprendizaje:

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$

Esto reduce la influencia de los gradientes más antiguos sobre la tasa de aprendizaje, lo cual reduce el riesgo de que esta se vuelva demasiado pequeña o grande.

Adam

Adam es un algoritmo propuesto por Kingma y Ba [14]. Este algoritmo emplea el promedio móvil de los gradientes cuadrados para escalar la tasa de aprendizaje, de manera similar a RMSProp. Adicionalmente, utiliza el promedio ponderado de los gradientes para calcular las actualizaciones de los pesos. La ecuación de actualización se define de la siguiente forma:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}}\hat{m}_t$$

Dónde $\hat{m}_t$ es el promedio de los gradientes anteriores y $\hat{v}_t$ son los gradientes cuadrados anteriores.

2.4. Redes Neuronales Convolucionales

Las RNA son capaces de realizar tareas de clasificación de imágenes [15] [16]. Sin embargo, realizar esta tarea haciendo uso de capas densas resulta computacionalmente costoso, debido a la gran cantidad de cálculos producto de la interconectividad entre todas las neuronas de cada par de capas. Por ejemplo, supongamos que tenemos una imagen a blanco

y negro de 28×28 píxeles, como es el caso de las imágenes de dígitos del conjunto de datos MNIST. Una RNA tradicional recibe como entrada vectores de tamaño fijo. Para poder dar como entrada una imagen a una de estas redes, es necesario convertirla a un vector de dicho tamaño. Para este fin, es posible "aplanar" las dimensiones de una imagen, es decir, ordenar las filas de manera consecutiva para obtener un vector. Así, podríamos aplanar nuestra imagen de 28×28 píxeles y convertirla en un vector de 784 elementos. Así, una RNA con 784 neuronas en su capa de entrada podría procesar y clasificar imágenes de este tamaño, siempre y cuando estas sean aplanadas con anterioridad. Sin embargo, lo que vuelve a las RNA imprácticas para este tipo de tareas es, como se mencionó al principio de este párrafo, la total interconectividad entre todas las neuronas de dos capas consecutivas. En el caso de nuestra RNA de 784 neuronas en la capa de entrada, si consideramos una capa oculta de 1000 neuronas, nuestra red realizaría $784 \times 1000 = 784000$ operaciones tan solo en su primer capa oculta. Esto no es un problema al procesar imágenes de baja resolución con redes relativamente poco profundas, pero cuando se incrementa la resolución de las imágenes o se agregan más capas a la red, el costo computacional se eleva rápidamente.

Las Redes Neuronales Convolucionales (RNC) son un tipo de Red Neuronal Artificial. Estas fueron popularizadas por Krizhekvsky et al. [4] al presentar la arquitectura AlexNet. Esta arquitectura hace uso de capas convolucionales consecutivas, además de la función de activación ReLU. Este fue el modelo ganador en el reto ImageNet LSVRC-2012. En una RNC, las neuronas de la capa anterior solamente se interconectan con las neuronas de una región de la capa anterior. Es decir, la influencia de las activaciones en una capa solo influyen en un subconjunto de las neuronas de la capa siguiente. A esta región se le conoce como campo receptivo. En la Figura 2.3 se puede observar una descripción visual del campo receptivo: la información presentada en color verde de la capa N influye en la activación de una neurona particular de la capa N+1, también presentada en color verde. A su vez, la información de la región coloreada de amarillo en la capa N+1 influye en la activación de una sola neurona de la capa N+2, y así sucesivamente. Este esquema reduce el número de operaciones a realizar, lo cual permite a las RNC procesar entradas de mayor volumen en un tiempo razonable. Las RNC se componen principalmente de tres tipos de capas: capa convolucional, capa de agregación o *pooling*, y capa densa.

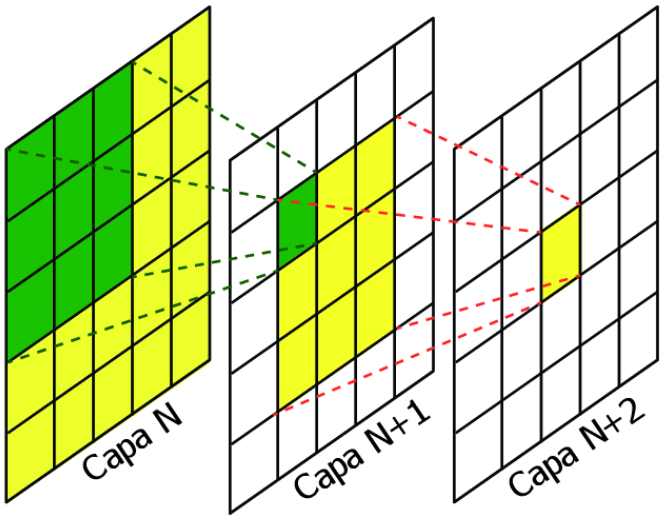

Figura 2.3: Ilustración del campo receptivo.

2.4.1. Capa Convolucional

La capa convolucional consiste de una serie de filtros (también llamados *kernels*) que se aplican a los datos de entrada. El propósito de estos filtros es detectar ciertas características en los datos de entrada. Las capas menos profundas detectan características sencillas, como bordes y esquinas. A medida que la red se vuelve más profunda, las características que detectan las capas convolucionales se vuelven más complejas. El concepto de convolución se ilustra en la Figura 2.4. Una matriz K de tamaño $n \times m$, la cual representa un filtro o *kernel*, se desliza sobre una matriz de entrada I, realizando una operación de multiplicación de matrices sobre cada segmento de tamaño $n \times m$ de la matriz de entrada. El valor resultante de dicha operación se almacena en una matriz de salida, denominada mapa de activaciones. Posteriormente, el filtro se mueve un cierto número de posiciones, para luego volver a aplicar la operación de multiplicación. La distancia que el filtro se mueve se denomina *stride*. Las operaciones que la capa convolucional realiza resultan en una reducción en la dimensionalidad sobre el volumen de entrada. Si se desea que los datos de salida conserven la dimensionalidad de los datos de entrada, comúnmente se incrementa la dimensionalidad de los mismos, rellenando los bordes con ceros. Este proceso se denomina rellenado o *padding*.

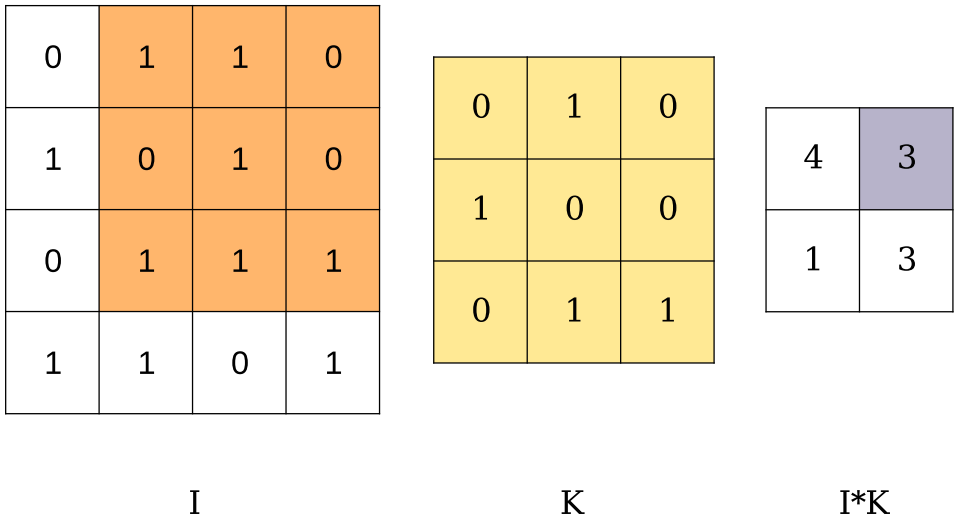

Figura 2.4: Operación de convolución.

2.4.2. Capa de agregación

La capa de agregación tiene como propósito unir o agrupar celdas del tensor de entrada. Esto reduce la dimensionalidad de la información, lo cual a su vez se traduce en una reducción del costo computacional. En estas capas, de forma similar a las capas convolucionales, una matriz de un cierto alto y ancho se desliza sobre la matriz de entrada, y se asigna el valor máximo (en el caso de una capa de agregación máxima) que se encuentre en la ventana a la posición correspondiente de la matriz de salida. Lo anterior puede verse ilustrado en la Figura 2.5, la cual presenta una operación de agregación máxima con un filtro de 2×2 y un paso de 2×2 (dos pasos verticales y dos horizontales). En esta figura, la matriz de 4×4 representa los datos de entrada, mientras que los cuadrantes de colores distintos representan las posiciones que tomará el filtro de agregación. Finalmente, la matriz más pequeña representa los datos de salida, dado que contiene los valores más altos contenidos en las submatrices de los colores correspondientes en la matriz de entrada. Existen tipos adicionales de operaciones de agregación, como agregación mínima, la cual toma el valor más pequeño en el filtro, y agregación media, la cual asigna el promedio de los valores en el filtro. Sin embargo, en la práctica es más prevalente la agregación máxima.

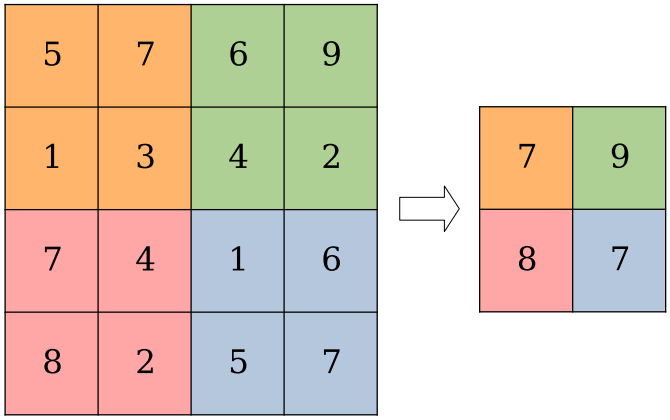

Figura 2.5: Operación de agregación máxima con un kernel de 2×2 y paso de 2.

2.4.3. Capa densa

La capa densa corresponde a las capas de neuronas de las RNA tradicionales, donde cada neurona en una capa particular se interconecta con cada neurona de la capa anterior. Generalmente, las capas densas se sitúan en los bloques finales de las RNC para realizar la clasificación final.

2.4.4. Problemas comunes de redes neuronales

En esta sub-sección se describen algunos de los problemas más conocidos que surgen al entrenar redes neuronales.

Saturación

La saturación en las redes neuronales es una condición que ocurre cuando las capas ocultas de una red neuronal arrojan valores que se encuentran en los extremos asintóticos de su función de activación [17]. Esto se traduce en gradientes muy pequeños durante el proceso de propagación hacia atrás, lo que ralentiza, y en algunos casos estanca, el entrenamiento de la red.

Gradiente que se desvanece y gradiente que explota

Como se mencionó anteriormente, el entrenamiento de una red neuronal se realiza propagando un valor de error desde la capa final de la red hasta la capa inicial, calculando los gradientes de los pesos de las capas con respecto del valor de error. Cuando los gradientes de las capas son demasiado pequeños, el valor de error que se propaga se reduce también, lo cual se traduce en cambios muy pequeños a los pesos de las capas. Esto se conoce como la gradiente que se desvanece o *vanishing gradient*. Este problema es más común en redes muy profundas, o cuyas capas cuentan con funciones de activación propensas a saturarse. Por otro lado, el problema de la gradiente que explota, o *exploding gradient* ocurre en condiciones opuestas: si las gradientes de las capas son demasiado grandes, el valor de error tiende a crecer demasiado, lo cual se traduce en cambios muy grandes a los pesos de la red.

2.4.5. Funciones de activación

Las funciones de activación son operaciones matemáticas que pueden aplicarse al resultado arrojado por las capas de una red neuronal. Ayudan a la red neuronal a aprender relaciones más complejas entre los datos de entrada y salida.

Sigmoid

La función Sigmoid está definida por la siguiente fórmula:

$$S(x) = \frac{1}{1 + e^{-x}}$$

El comportamiento básico de esta función es recibir un valor de entrada y reducirlo a un número real entre cero y uno. Esta solía ser la función de activación más común en las RNA, debido a que se adapta a la idea de una neurona que se activa, siendo 1 equivalente a una neurona completamente encendida y 0 equivalente a una neurona completamente apagada. Sin embargo, debido a su propiedad de reducir cualquier valor a un espacio entre cero y uno, cambios grandes en la entrada de la función provocan cambios muy pequeños en su salida. Esto introduce el problema de la gradiente que se desvanece (*vanishing gradient*) para redes muy profundas.

Tanh

La función Tanh se define con la fórmula:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Esta función se comporta de manera similar a la Sigmoid, dado que convierte su valor de entrada a un número real en el rango entre -1 y 1. La ventaja de esta función es que sus valores de salida tienen una media de cero, lo cual es favorable para el entrenamiento de las RNA ya que reduce el riesgo de que las capas se saturen. Sin embargo, también es propensa a introducir el problema de la gradiente que se desvanece en redes más profundas.

ReLU

La función ReLU (Rectified Linear Unit) es actualmente la función de activación más popular en redes neuronales convolucionales. Se define de la forma siguiente:

$$f(x) = max(0, x)$$

Esta función presenta varias ventajas en contraste con $Sigmoid$ y $Tanh$. Es computacionalmente sencilla de calcular, no se satura en la región positiva, y converge mucho más rápido [4].

2.5. Uso de RNC en tareas de clasificación de imágenes

En contraste con los métodos tradicionales de clasificación, las redes neuronales convolucionales presentan la ventaja de ser capaces de encontrar por si mismas las características que mejor ayudan a discriminar entre las clases que se le presenten, eliminando la necesidad de desarrollar manualmente detectores de características especializados y permitiendo su uso en diversas tareas de clasificación.

El principal problema que se encuentra en el uso de RNCs es la alta cantidad de datos que se requieren para su entrenamiento. Sin embargo, proyectos tales como ImageNet han contribuido significativamente en la obtención de datasets que cubren la alta diversidad de

plantas en el mundo, y actualmente proyectos como Pl@ntNet y iNaturalist se valen de comunidades de voluntarios para obtener y clasificar plantas alrededor del mundo.

En 2012, la arquitectura AlexNet [4] se convirtió en la primera propuesta de RNC en ganar el reto ILSVRC (*ImageNet Large Scale Visual Recognition Challenge*), el cual es un reto anual que consiste en realizar tareas de clasificación de imágenes, así como detección de objetos, usando imágenes del conjunto de datos ImageNet. El equipo de Alex Krizhevsky, comformado por investigadores de la Universidad de Toronto, propuso una arquitectura de RNC que presentaba capas convolucionales consecutivas. Esta arquitectura superó a las demás propuestas de la edición 2012 de ILSVRC por un margen de 10.9 % en tasa de error Top-5 (está métrica de evaluación, conocida como Top-K, se describe a mayor detalle en la Sección 4.1 de esta tesis). Desde entonces, las RNC se han mantenido al frente en tareas de clasificación a partir de imágenes, obteniendo resultados consistentes en diversos retos de clasificación como LifeCLEF, el cual consiste en proponer y medir el desempeño de modelos automatizados de identificación de plantas, con la finalidad de solucionar el problema de escasez de expertos capaces de realizar estas tareas.

En el estudio publicado por Lee et. al. [18], se realizó una comparación entre el uso de RNC y los métodos de clasificación tradicionales basados en la extracción manual de características. Para esto, los investigadores construyeron un conjunto de datos con imágenes de hojas pertenecientes a 44 especies vegetales, tomadas del Jardín Botánico Real en Kew, Inglaterra, el cual consiste en fotografías de 44 clases distintas de plantas. Este dataset se denominó *MalayaKew Leaf Dataset*.

El estudio realizó una comparativa de desempeño de distintos clasificadores, principalmente:

- Perceptrón Multi-Capa: modelos de aprendizaje profundo equivalentes a una Red Neuronal Artificial clásica. Se conforman de capas de perceptrones densamente interconectadas.
- Máquinas de Vectores de Soporte: modelos de aprendizaje máquina versátiles. Se utilizan en tareas de clasificación y regresión, entre otras.
- Funciones de Base Radial: funciones de una sola variable que se utilizan en aprendi-

Característica	Clasificador	Precisión (%)
De RNC Profunda (D1)	MLP	0.977
De RNC Profunda (D1)	SVM (Lineal)	0.981
De RNC Profunda (D2)	MLP	**0.995**
De RNC Profunda (D2)	SVM (Lineal)	0.993
Leafsnap	SVM (RBF)	0.420
Leafsnap	NN	0.589
HCF	SVM (RBF)	0.716
HCF-ScaleRobust	SVM (RBF)	0.665
Combine	Sum rule (SVM (Lineal))	0.951
SIFT	SVM (Lineal)	0.588

Tabla 2.1: Comparativa de desempeño en el dataset MalayaKew con clasificadores distintos. MLP = Perceptrón Multicapa (Multi-Layer Perceptron), SVM = Máquina de Vectores de Soporte (Support Vector Machine), y RBF = Función de Base Radial (Radial Base Function).

zaje máquina para aproximar funciones multivariable.

El estudio encontró que las RNC superan de manera significativa a los métodos tradicionales basados en extracción manual de características, como se observa en la Tabla 2.1.

Reyes et al. [19] aplicaron transferencia de aprendizaje (*transfer learning*) a una RNC con arquitectura AlexNet modificada con una capa de predicción adicional. Esta red fue pre-entrenada usando 1.8 millones de imágenes genéricas, la cual re-entrenaron con el dataset de LifeCLEF para el problema de identificación de plantas. La transferencia de aprendizaje es una técnica de aprendizaje máquina para entrenar redes neuronales, previamente descrita en la sub-sección 2.3.6. La Tabla 2.2 muestra los resultados obtenidos por los grupos participantes en la edición 2015 de LifeCLEF, donde los investigadores del grupo ECOUAN obtuvieron el cuarto lugar en 2015.

Lee et al. [1] utilizaron una RNC con arquitectura VGG16 modificada para el reto LifeCLEF 2016, cuyo objetivo es identificar 1000 especies vegetales a partir de imágenes de siete órganos distintos (la arquitectura VGG16 se describe a detalle en la Sección 3.2.2

Grupo Investigador	Precisión, imagen individual	Precisión, observación
SNUMED INFO	0.652	0.667
QUT RV	0.590	0.624
INRIA ZENTH	0.581	0.609
ECOUAN	**0.486**	**0.487**
MICA	0.194	0.209
SABANCI	0.153	0.160
UAIC	0.013	0.013

Tabla 2.2: Desempeño del equipo ECOUAN en el reto de clasificación LifeCLEF 2015. La precisión individual describe el desempeño del modelo con una sola imagen. La precisión por observación describe el desempeño utilizando un conjunto de cinco imágenes de distintos órganos de una misma planta.

de este documento). Las modificaciones a la red consistieron en la adición de dos canales de capas, orientados a la detección de dos elementos distintos: órgano y especie. Estos dos canales comparten un subconjunto de capas iniciales. Posteriormente los mapas de características se concatenan para la clasificación final.

Al mismo tiempo, se buscó rechazar especies desconocidas. Para esto, la arquitectura desarrollada por los investigadores tenía dos clasificadores en paralelo para identificar características de especie y de órgano por separado. Estos dos clasificadores compartían un cierto número de capas iniciales, y fueron pre-entrenados con el conjunto de datos ImageNet. Posteriormente, uno de los clasificadores fue re-entrenado para identificar órganos, mientras que el otro fue re-entrenado para identificar especies. Una vez hecho esto, se agregaron capas de "fusión", las cuales concatenaban la salida de las capas de órgano y las de especie. Finalmente, la arquitectura fue refinada de principio a fin utilizando la misma tasa de aprendizaje.

Además del modelo propuesto, los investigadores aplicaron aumentación de datos (*data augmentation*) al conjunto de datos. Esta técnica consiste en aplicar transformaciones aleatorias a las imágenes, para incrementar la variedad en las imágenes y ayudar al modelo a aprender generalizaciones útiles para el proceso de clasificación. El método particular utilizado consistió en re-escalar las imágenes del conjunto de datos a diversos tamaños, y

Método	Sin aumentación	Con aumentación
VGG16 con capa final refinada	56.4	71.2
Modelo con fusión de alto nivel	54.4	68.9

Tabla 2.3: Desempeño del clasificador propuesto por Lee et al. en el reto ImageNet 2016.

posteriormente extraer recortes aleatorios de 224 por 224 píxeles para entrenar a la red.

Los resultados obtenidos por los investigadores pueden observarse en la Tabla 2.3. Estos resultados fueron obtenidos por su modelo con fusión de características y comparados con los de una red VGG16 con una capa final refinada. Al final, la precisión obtenida por la red VGG16 fue superior a la del modelo propuesto. Sin embargo, también se observó que los modelos entrenados con aumentación de datos obtuvieron resultados significativamente mejores a aquellos entrenados con datos sin modificar.

Hang et al. [3] utilizaron una RNC con arquitectura VGG16 para el reto LifeCLEF 2016. La red recibió algunas modificaciones. Primero, la última capa de agregación fue reemplazada por una capa de agregación piramidal espacial (*Spatial Pyramid Pooling*). Esta capa de agregación divide la imagen en subregiones de 1×1, 2×2 y 4×4, y posteriormente aplica agregación de promedio a cada una de estas regiones. El resultado es un vector de tamaño fijo sin importar el tamaño de la imagen de entrada, lo que le permite al modelo recibir entradas de tamaño arbitrario. Además de esto, la función de activación ReLU estándar fue reemplazada por ReLU paramétrica (*Parametric ReLU*), la cual es un tipo de función ReLU con "fuga". Una función ReLU con fuga multiplica valores de entrada negativos por un factor muy pequeño, lo cual evita que las neuronas "mueran" como consecuencia de la multiplicación por cero que ocurre con una activación ReLU común. En una función ReLU con fuga estándar, el valor por el cual se multiplica la entrada es fijo. En contraste, ReLU paramétrica convierte este valor en un parámetro que la red puede ajustar. La Tabla 2.4 muestra los resultados obtenidos por los participantes del reto. El modelo propuesto por los investigadores alcanzó un promedio de precisiones medias (en inglés *Mean Average Precision*) o MAP de 0.827, el más alto de entre todos los participantes del reto. Esta métrica está dada por la siguiente función:

$$MAP = \frac{\sum_{q=1}^{Q} AveP(q)}{Q}$$

Donde Q es el número de muestras en el conjunto de validación y $AveP(q)$ es la precisión media para una muestra q.

Mehdipour et al. [20] pre-entrenaron, usando el conjunto de datos ILSVRC 2012, dos RNC con arquitecturas GoogLeNet, la cual es una variante de la arquitectura Inception descrita en la Sección 3.2.2 de esta tesis, y VGG19, la cuál se describe en la Sección 3.2.2. Los modelos fueron posteriormente afinados con el conjunto de datos LifeCLEF 2015, esto con el propósito de mejorar los resultados obtenidos en dicho conjunto de datos. Para esto, experimentaron con distintas maneras de afinar sistemas de aprendizaje profundo, particularmente usando diferentes tamaños de lote y número de iteraciones, así como haciendo uso de aumentación de datos. Adicionalmente, entrenaron GoogLeNet para que rechazara imágenes que no correspondiesen a plantas en general. Esto se logró entrenando la red GoogLeNet como un clasificador binario que decidía si una muestra es o no una planta. Al final, el mejor modelo alcanzó un puntaje MAP de 0.738 en el dataset PlantCLEF 2016.

Toma et al. [21] utilizaron transferencia de aprendizaje (*transfer learning*) con una RNC tipo AlexNet para el reto de clasificación PlantCLEF 2017. Los investigadores aplicaron refinamiento con el conjunto de datos EOL (Encyclopedia of Life) de PlantCLEF 2017, el cual consiste de aproximadamente 260,000 imágenes pertenecientes a 10,000 especies. El modelo fue entrenado con una tasa de aprendizaje de 0.001 y un tamaño de lote de 512 imágenes. Para evitar el problema de sobreajuste, los investigadores aplicaron dos estrategias distintas: primero, el proceso de entrenamiento era detenido cuando la precisión media de las últimas 50 iteraciones era mayor a 99 %, y segundo, impusieron un factor de regularización L2 de 0.001. Este factor L2 tiene como objetivo mitigar el riesgo de sobreajuste del modelo. Esto se logra sumando un factor de penalización a la función de error, equivalente a la suma de los cuadrados de los pesos del modelo, multiplicada por un factor λ:

$$L2 = \lambda \sum_{i=1}^{n} \sum_{j=1}^{m} w_{ij}^2$$

Adicionalmente, el proceso de aprendizaje fue acelerado en la última capa densa, mul-

Corrida	Puntaje MAP oficial	MAP restringido a especies potencialmente invasivas	MAP ignorando especies desconocidas
Bluefield 4	0.742	0.717	0.827
SabanciUGebzeTU 1	0.738	0.704	0.806
SabanciUGebzeTU 3	0.737	0.703	0.807
Bluefield 3	0.736	0.718	0.820
SabanciUGebzeTU 2	0.736	0.683	0.807
SabanciUGebzeTU 4	0.735	0.695	0.802
CMP 1	0.710	0.653	0.790
LIIR KUL 3	0.703	0.674	0.761
LIIR KUL 2	0.692	0.667	0.744
LIIR KUL 1	0.669	0.652	0.708
UM 4	0.669	0.598	0.742
CMP 2	0.644	0.564	0.729
CMP 3	0.639	0.590	0.723
QUT 3	0.629	0.610	0.696
Floristic 3	0.627	0.533	0.693
UM 1	0.627	0.537	0.700
Floristic 1	0.619	0.541	0.694
Bluefield 1	0.611	0.600	0.692
Bluefield 2	0.611	0.600	0.693
Floristic 2	0.611	0.538	0.681
QUT 1	0.601	0.563	0.672
UM 3	0.589	0.509	0.652
QUT 2	0.564	0.562	0.641
UM 2	0.481	0.446	0.552
QUT 4	0.367	0.359	0.378
BME TMIT 4	0.174	0.144	0.213
BME TMIT 3	0.170	0.125	0.197
BME TMIT 1	0.169	0.125	0.196
BME TMIT 2	0.066	0.128	0.101

Tabla 2.4: Evaluación de resultados ordenados por puntaje MAP oficial para el reto LifeCLEF 2016.

tiplicando por 10 la tasa de aprendizaje de ésta. Al final, el modelo entrenado alcanzó un puntaje MRR (*Mean Reciprocal Rank* o Rango Recíproco Medio) de 0.361 en su mejor corrida. El puntaje MRR es una métrica utilizada para evaluar procesos que producen una lista de respuestas ordenadas por probabilidad de corrección. Está dado por la siguiente fórmula:

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{rank_i}$$

Donde $rank_i$ es la posición en la cual se encuentra la respuesta correcta.

Pawara et al. [22] utilizaron AlexNet y GoogleNet para clasificar clases de tres conjuntos de datos diferentes: Folio [23], AgrilPlant [24], y Swedish Leaf [25]. A su vez, utilizaron una serie de técnicas de aumentación de datos (*data augmentation*) para contrastar el desempeño con modelos entrenados sin aumentación de datos y así medir la eficacia de esta técnica. La conclusión a la cual llegaron los investigadores con sus resultados, es que los métodos de aumentación de datos son importantes cuando se busca incrementar la precisión de Redes Neuronales Convolucionales cuando los modelos son entrenados desde cero. Esto demuestra la importancia de tener una alta cantidad de datos de entrenamiento. También concluyeron que las técnicas de rotación-iluminación y rotación-contraste son especialmente útiles en este aspecto.

Barre et al. [26] diseñaron un sistema de identificación de plantas basado en Redes Neuronales Convolucionales, al cual denominaron LeafNet. Esta arquitectura posee módulos de reducción de dimensionalidad, los cuales consisten de capas convolucionales consecutivas. En el caso de LeafNet, los módulos se conforman de dos capas convolucionales seguidas por una capa de agregación máxima con filtro de 2×2. LeafNet se compone de cinco de estos módulos, seguidos por una capa convolucional y una capa de agregación, y termina con tres capas densas al final.

Los modelos fueron entrenados utilizando en conjunto de datos LeafSnap [7]. Adicionalmente, utilizaron aumentación de datos para incrementar el tamaño del conjunto de datos original. La red utilizada para LeafNet fue entrenada durante 200,000 iteraciones, utilizando un valor de impulso de 0.9, lotes de 10 imágenes y una tasa de aprendizaje de 0.001. El valor de impulso o *momentum* consiste en un promedio ponderado de los gra-

Dataset	# especies	Top-1	Top-5	Puntaje MRR	Puntaje MAP
LeafSnap	184	86.3 %	97.8 %	92.2 %	83.7 %
Foliage	60	95.8 %	99.6 %	97.6 %	95.3 %
Flavia	32	97.9 %	99.9 %	98.8 %	97.2 %

Tabla 2.5: Resultados de LeafNet en datasets Flavia, Foliage y LeafSnap

dientes previos. Agregar este término al valor por el cual se actualizan los pesos de un modelo tiene como propósito mantener la dirección general en la que se mueve la función de pérdida.

Al cabo de 100,000 iteraciones la tasa de aprendizaje se redujo a 0.0001 para fomentar la convergencia del modelo. Además de LeafSnap, la red fue entrenada para dos conjuntos de datos adicionales: Foliage y Flavia. Los resultados obtenidos por la red en cada dataset se muestran en la Tabla 2.5. Al final, el sistema demostró tener un desempeño superior a métodos tradicionales con detectores de características diseñados a mano para la clasificación de imágenes en los datasets Foliage, LeafSnap y Swedish Leaf.

Carpentier et al. [27] entrenaron múltiples redes con arquitectura ResNet, descrita en la Sección 3.2.2, para identificar especies de árboles nativos de Canadá a partir de imágenes de cortezas. Para esto, contruyeron un conjunto de datos público que contiene más de 23,000 imágenes de las cortezas de 23 especies disintas de árboles presentes en en parques y bosques en la ciudad de Quebec, Canadá, al cual denominaron BarkNet 10.0. Los investigadores utilizaron modelos con arquitectura ResNet, en concreto ResNet-18 y ResNet-34, pre-entrenados con el conjunto de datos ImageNet. La primera capa de los modelos fue congelada (es decir, se deshabilitó el ajuste de los pesos de esta capa), y posteriormente los modelos fueron afinados con una tasa de aprendizaje inicial de 0.0001. La tasa de aprendizaje se redujo en un factor de 5 en las épocas 16 y 33, mientras que el entrenamiento duró 40 épocas. El algoritmo de optimización utilizado fue Adam, con un valor de decaída de pesos de 0.0001. Por su parte, las fotografías del conjunto de datos fueron reducidas en tamaño a la mitad, esto con el propósito de acelerar la velocidad de carga y preprocesamiento de las mismas. Al final, los modelos entrenaros lograron una precisión máxima de 97.81 % al integrar las decisiones del clasificador sobre múltiples imágenes de un mismo árbol, y

93.88 % clasificando imágenes de manera individual. La precisión se obtiene dividiendo el número de clasificaciones correctas entre el número de clasificaciones totales.

Recientemente, los trabajos con RNC se han enfocado en optimizar las arquitecturas de estas para obtener mejores grados de precisión en las predicciones, así como la clasificación utilizando múltiples órganos, por ejemplo Lee et al. [28], quienes presentaron dos *frameworks* de clasificación. El primero, denominado Red Neuronal Convolucional Híbrida Genérico-Órgano (HGO CNN), combina características particulares de los órganos de las plantas con características genéricas para realizar la clasificación. La arquitectura del modelo consiste en un bloque de capas compartidas. Estas tienen como propósito detectar características de bajo nivel, como esquinas y bordes. Posteriormente, la salida de este bloque es enviada a dos bloques paralelos. Uno de estos bloques se encarga de procesar características de los órganos, mientras el otro es entrenado para detectar características genéricas. Finalmente, la salida de ambos bloques es concatenada para realizar la clasificación final. Además de GO CNN, los investigadores desarrollaron un enfoque basado en Redes Neuronales Recurrentes (RNN por sus siglas en inglés), denominado PlantStruct-Net. Las RNN son una clase de redes neuronales capaces de almacenar información de datos de entrada anteriores, y así mismo utilizar esta información en el procesamiento de datos subsecuentes. El modelo PlantStruct-Net fue entrenado para modelar las dependencias conextuales de alto nivel que existen entre imágenes de una misma planta vista desde ángulos diferentes.

Los resultados obtenidos por Waldchen et al. [10] en su revisión del estado del arte en materia de identificación de plantas demuestran que las RNC se desempeñan consistentemente mejor que los métodos de clasificación tradicionales. Esto puede observarse en la Tabla 2.6. Así mismo, los autores identificaron un número de problemas por resolver y direcciones futuras que puede tomar la investigación de clasificación de plantas, tales como:

- La necesidad de crear conjuntos de datos de referencia más representativos, dado que los conjuntos de referencia actuales se encuentran limitados tanto en número de especies como en número de observaciones. Esto trae como consecuencia una baja representatividad de condiciones reales en estos conjuntos de datos, lo cual los hace

poco útiles para aplicaciones reales.

- La posibilidad de usar colaboración masiva (*crowdsourcing*) para construir conjuntos de datos. iNaturalist y Pl@ntNET son dos ejemplos de proyectos que han tenido éxito al usar colaboración masiva para construir conjuntos de datos.

- Utilizar información contextual para apoyar al proceso de identificación. Datos tales como la ubicación de una observación, fecha, hora, tipo de suelo y condiciones climáticas son potencialmente útiles para el proceso de identificación.

- La posibilidad de utilizar los avances más recientes en materia de aprendizaje profundo. Los investigadores de aprendizaje profundo constantemente proponen nuevas técnicas y algoritmos, los cuales pueden utilizarse para atacar el problema de clasificación de plantas.

- Utilizar los millones de ejemplares contenidos en herbarios alrededor del mundo para agilizar el proceso de recolección de datos. Si bien los especímenes digitalizados en herbarios se encuentran en condiciones muy distintas a los especímenes en entornos naturales, la posibilidad de utilizar estos especímenes para entrenar modelos que identifiquen especímenes en un entorno natural continua siendo de interés para los investigadores.

- Realizar colaboraciones interdisciplinarias entre expertos en computación y expertos en biología. Los expertos en biología pueden implementar técnicas de aprendizaje profundo de manera más efectiva con la ayuda de los expertos en computación, mientras que estos últimos pueden obtener una mejor comprensión del problema al que se enfrentan con la ayuda de los biólogos.

	Basados en modelos		Libres de modelos		Aprendizaje profundo	
Dataset	Precisión	Autor	Precisión	Autor	Precisión	Autor
Swedish Leaf	82.0 %	[29]	93.7 %	[30]	99.8 %	[22]
Flavia	90.3 %	[31]	95.9 %	[32]	99.7 %	[33]
Leafsnap	73.0 %	[7]	72.6 %	[24]	97.6 %	[24]
ICL	83.8 %	[34]	91.3 %	[34]	93.9 %	[35]
Oxford Flower 17	-		91.8 %	[36]	96.6 %	[37]
Oxford Flower 102	-		90.2 %	[38]	96.6 %	[39]

Tabla 2.6: Muestra del incremento en precisión lograda con distintos enfoques sobre conjuntos de datos de plantas conocidos.

Capítulo 3

Metodología

La propuesta consiste en medir la precisión de varias arquitecturas de redes neuronales existentes para identificar especies, elegir la arquitectura con el mejor desempeño y tomar medidas para aumentar su precisión. Las medidas posibles son intentar reducir el tamaño de la red sin afectar su precisión, y hacer alteraciones a la arquitectura con el objetivo de aumentar su precisión.

Las arquitecturas más prometedoras fueron entrenadas con un conjunto de datos de la flora nativa de México. Este conjunto de datos se obtuvo de diversas fuentes, en concreto salidas a campo realizadas por equipos de fotógrafos guiados por biólogos expertos, así como descargas de sitios como iNaturalist. Posteriormente las imágenes recolectadas fueron recortadas utilizando un software diseñado específicamente para la tarea. El propósito de estos recortes es aislar los órganos distintivos de las plantas, para así mejorar el desempeño de los modelos de clasificación, debido a que se reduce la cantidad de ruido, es decir, información no útil para el modelo.

3.1. Conjunto de datos

El conjunto de datos utilizado consiste en recortes de áreas de interés en fotografías. Elementos distintivos de las plantas, aquellos que ayuden a su identificación, tales como flores, frutos y hojas, constituyen áreas de interés. De esta forma se espera mejorar la eficacia de los modelos de clasificación, debido a que se reduce la cantidad de ruido, es decir,

información no útil para el modelo.

El conjunto de datos utilizado en este trabajo consiste en 5560 imágenes, divididas en 75 especies distintas. De las imágenes de estas 75 especies, se extrajeron regiones de interés correspondientes a elementos anatómicos distintivos de cada especie, obteniendo al final 112 clases diferentes, cada una representando un órgano específico de una especie en particular.

Es relativamente sencillo proporcionar evidencia de algunas de las dificultades que presenta la tarea de clasificación de plantas en las imágenes del conjunto de datos. Por ejemplo, la apariencia de las plantas puede variar drásticamente dependiendo del ángulo del cual fueron tomadas las fotografías. También es posible observar varios ejemplos de alta variabilidad intraclase y baja variabilidad interclase en las imágenes del conjunto de datos. La alta variabilidad intraclase se presenta cuando ejemplares de una misma especie lucen muy diferentes entre sí. Esto puede observarse en la Figura 3.1. En esta figura se muestran cuatro imágenes de un mismo órgano de una especie particular, las cuales lucen muy diferentes entre sí.

Por otro lado, la baja variabilidad interclase se presenta cuando ejemplares de diferentes especies lucen muy similares entre sí. Un ejemplo de esto puede verse en la Figura 3.2, donde puede apreciarse un alto grado de similitud en la forma de las flores de cinco especies distintas.

Figura 3.1: Ejemplo de alta variabilidad intraclase. Todos los ejemplares mostrados pertenecen a la especie *Dodonaea viscosa*.

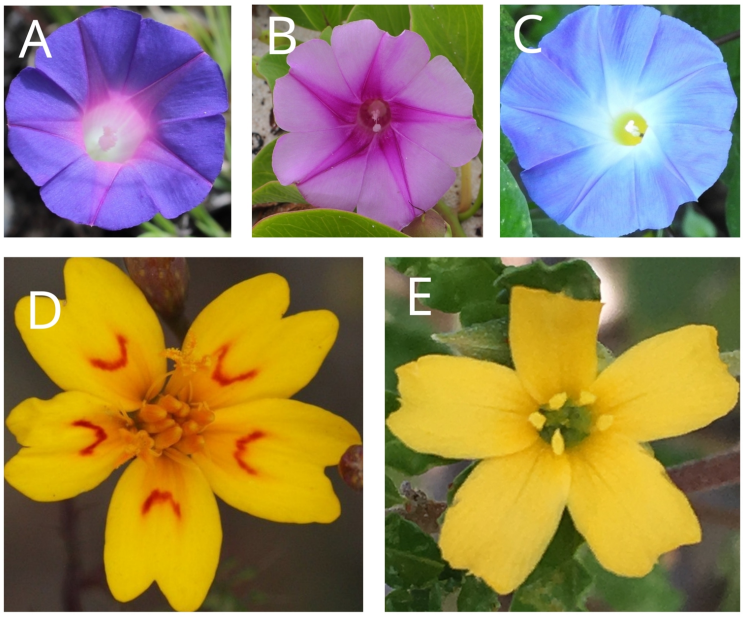

Figura 3.2: Ejemplo de baja variabilidad interclase. Los ejemplares mostrados corresponden a las siguientes especies: (A) *Ipomoea purpurea*, (B) *Impomoea tricolor*, (C) *Ipomoea pes-caprae*, (D) *Tagetes lunulata*, (E) *Turnera diffusa*.

3.2. Herramientas

Para la implementación de los diversos modelos de RNCs utilizadas en los experimentos de esta tesis se utilizó Keras. Keras es una API orientada al diseño y manejo de modelos de aprendizaje máquina, escrita en Python. Esta API se apoya sobre distintos *backends* tales como Tensorflow, para simplificar y facilitar el trabajar con modelos de aprendizaje máquina.

Se diseñó un script en Python, que nos permite tomar una de las diversas arquitecturas de RNC presentes en Keras (*e.g.,* VGG16, Xception, ResNet) y entrenarla con cualquier conjunto de datos de imágenes. El script permite especificar todos los hiperparámetros involucrados en el entrenamiento de una red neuronal, desde número de épocas, tamaño de lote, tasa de aprendizaje, algoritmo de optimización y sus parámetros, etc.

Los modelos actuales de clasificación visual trabajan con tamaños de imágenes fijos (224×224 en el caso de VGGNet, por ejemplo). Esto quiere decir que una imagen que no es del tamaño de entrada estándar del modelo tendrá que redimensionarse para ser acepta-

da por el mismo. Esto tiene como consecuencia pérdidas de información de la imagen al reducir su resolución, y distorsionamiento en casos donde la imagen tenga que estirarse o encogerse en uno de sus ejes para ajustarse a la relación de aspecto requerida por el modelo. El recortar las imágenes nos ayuda a reducir este problema, puesto que todos los recortes son de una dimensión estándar proporcional al tamaño de entrada de los modelos.

3.2.1. Herramienta de recortes

Para llevar a cabo el pre-procesamiento de imágenes, requerido para construir el conjunto de datos para entrenar los modelos construidos, fue necesario desarrollar un software de etiquetado personalizado. Este software, denominado PitCrop, nos permite definir regiones de interés en las imágenes de nuestro conjunto de datos. Un ejemplo de esto puede observarse en la Figura 3.4, donde se ha seleccionado la región correspondiente a la flor de la planta. La información que describe las coordenadas y dimensiones de las regiones a extraer de una imagen determinada es almacenada en archivos CSV durante el proceso de definición de regiones. Posteriormente, un script de Python se encarga de leer las regiones especificadas de las imágenes correspondientes, para después almacenarlas como imágenes nuevas en un directorio por separado. De esta forma, podemos extraer imágenes que corresponden a los órganos de las plantas en las imágenes, mientras que las imágenes originales permanecen intactas. Dicho software fue desarrollado usando la versión de código abierto del framework Qt, y se encuentra a disposición del público bajo una licencia de uso libre, de forma que pueda ser utilizado en otros proyectos de etiquetado de datos. La interfaz de esta herramienta puede verse en la Figura 3.3.

3.2.2. Arquitecturas

En este apartado se describen las arquitecturas utilizadas en los experimentos realizados por nosotros. Todas las implementaciones se encuentran gratuitamente dentro del framework Keras.

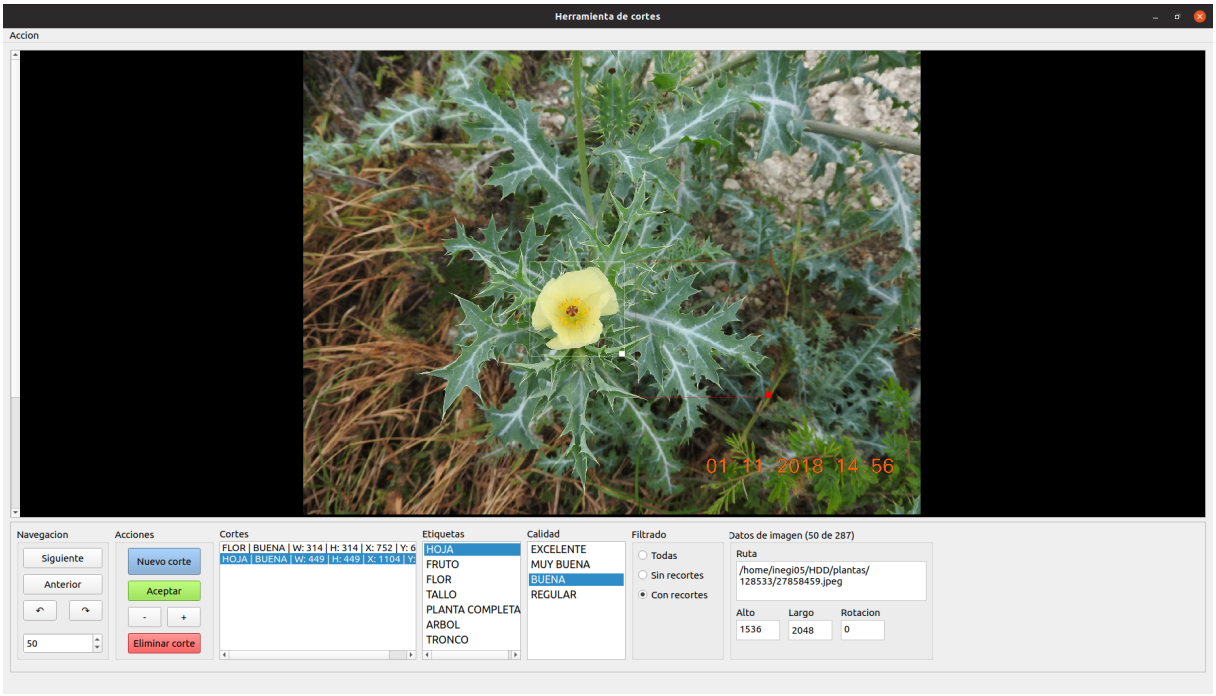

Figura 3.3: Interfaz de PitCrop.

Figura 3.4: Ejemplo de recorte.

VGG16 y 19

Las arquitecturas VGG [40] fueron propuestas en 2014, siendo utilizadas para clasificar el conjunto de datos del reto ImageNet 2014, el cual contiene imágenes de 1000 clases distintas de objetos, las cuales van desde animales hasta vehículos. El estudio presentó seis variantes: tres de estas contienen 133 millones de parámetros (A, A-LRN, B), mientras que el resto contiene 134, 138 y 144 millones (C, D y E, respectivamente).

Las arquitecturas VGG están compuestas por secuencias de bloques, los cuales a la vez están conformados por cierta cantidad de capas convolucionales y una capa de agregación, seguidos por 3 capas completamente conectadas y una capa softmax al final. El tamaño de las imágenes de entrada es fijo, de 224×224.

VGG19 está conformada por cinco de estos bloques: los dos primeros contienen dos capas convolucionales de 64 y 128 filtros respectivamente; el bloque intermedio contiene tres capas convolucionales de 256 filtros, y los dos últimos contienen ambos tres capas convolucionales de 512 filtros.

Los números en VGG16 y VGG19 representan la cantidad de capas entrenables contenidas en la arquitectura: 16 capas convolucionales y 3 capas densas en el caso de VGG19, y 13 capas convolucionales y 3 capas densas en el caso de VGG16.

Las capas convolucionales cuentan con un campo receptivo de 3×3, paso (*stride*) de 1×1 y relleno (*padding*) de 1 pixel. Las operaciones de agregación se realizan con un kernel de 2×2 y un paso de 2×2. Cada capa oculta de la red cuenta con la función de activación ReLU. La arquitectura completa está descrita en la Tabla 3.1, donde "Conv2D" corresponde a una capa convolucional bidimensional, "Agreg. Max." corresponde a una capa de agregación máxima, "Densa" corresponde a una capa densa, y "Sotfmax" corresponde a una capa Softmax.

Tipo	N filtros/parámetros
Conv2D	64
Conv2D	64
Agreg. Max,	N/A
Conv2D	128
Conv2D	128
Agreg. Max,	N/A
Conv2D	256
Conv2D	256
Conv2D	256
Conv2D	256
Agreg. Max,	N/A
Conv2D	512
Conv2D	512
Conv2D	512
Conv2D	512
Densa	N/A
Conv2D	512
Conv2D	512
Conv2D	512
Conv2D	512
Agreg. Max.	N/A
Densa	4096
Densa	4096
Densa	1000
Softmax	N

Tabla 3.1: Arquitectura de VGG19.

Inception

Google presentó la primera versión de la arquitectura Inception en 2015, bajo el nombre de GoogLeNet [41]. En su trabajo, los investigadores señalan el problema de seleccionar un tamaño de kernel correcto, debido a la potencial variación en el tamaño y localización de las áreas de interés en las imágenes.

Para solucionar esto, los investigadores diseñaron el módulo Inception, el cual consiste de una combinación de capas convolucionales paralelas, cuyos mapas de activación se concatenan al final. En concreto, las capas del módulo Inception se estructuran en múltiples canales, de forma que cada canal recibe su propia copia de los datos de entrada, y realiza el procesamiento de estos de forma única. Lo anterior dicho se ilustra en la Figura 3.5, donde observamos el módulo Inception clásico con cuatro canales: uno compuesto de una capa de agregación máxima seguida de una capa convolucional de 1×1, dos compuestos de dos capas convolucionales cada uno, con una capa convolucional de 1×1 al principio pero con tamaños de kernel diferentes en la segunda capa (3×3 y 5×5), y uno más compuesto solamente de una capa convolucional de 1×1. Todas las variantes de la arquitectura Inception consisten en diversas configuraciones de este módulo.

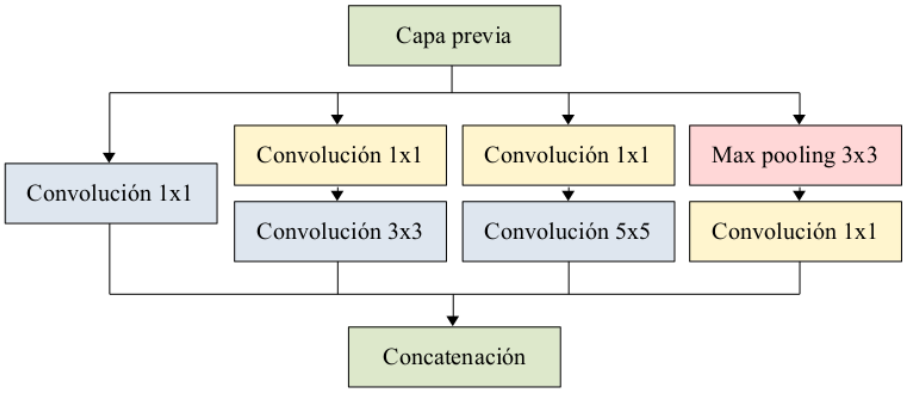

Figura 3.5: Módulo Inception clásico [42].

La principal ventaja del módulo Inception con respecto a las RNC tradicionales, es que le permite a la red decidir por cuenta propia el tamaño de filtro más relevante para aprender, ya que utiliza filtros de distintos tamaños en sus capas convolucionales. Con el propósito de mitigar el incremento en costo computacional, el módulo Inception hace uso

de convoluciones de 1×1 para reducir la dimensionalidad de los datos antes de enviarlos a las capas de convolución de 3×3 y 5×5. De esta forma es posible crear una red profunda sin sacrificar demasiado en desempeño.

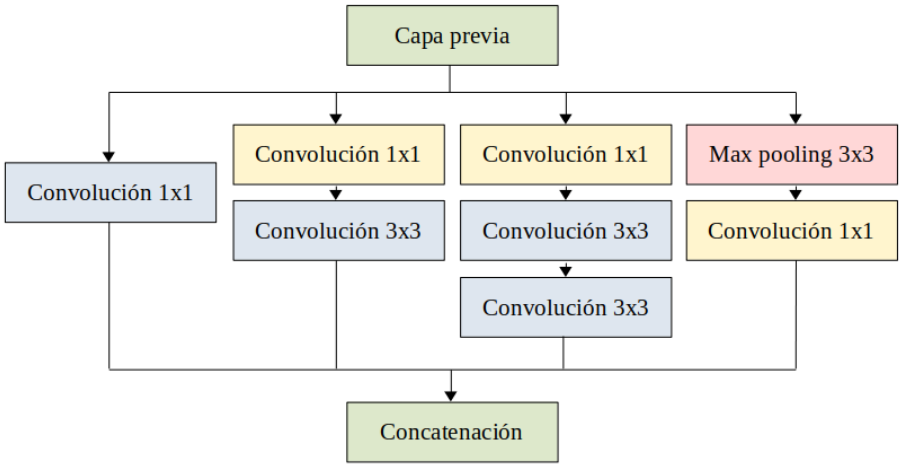

Figura 3.6: Módulo Inception v2 [42].

Adicionalmente, la arquitectura cuenta con dos clasificadores auxiliares, cuyo propósito es mitigar el problema de la gradiente que se desvanece (*vanishing gradient*), el cual ocurre cuando los valores de los ajustes a los pesos de la red (las gradientes) se vuelven demasiado pequeños para ser significativos. En resumen, estos clasificadores producen dos valores de error auxiliares, los cuales se suman parcialmente al valor de error real durante el entrenamiento.

La segunda versión de esta arquitectura se presentó en conjunto con la versión 3. Como puede verse en la Figura 3.6, la nueva versión del módulo Inception era bastante similar a la anterior, siendo el cambio más notable la factorización de la capa convolucional de 5×5 a dos capas convolucionales de 3×3, esto con el fin de reducir el costo computacional. Adicionalmente, las convoluciones de $N \times N$ son reemplazadas por dos convoluciones con las forma $1 \times N$ y $N \times 1$ (Figura 3.7). Este método resultó ser hasta 33 % más eficiente en número de operaciones que las convoluciones tradicionales.

Además de las mejoras presentes en la versión 2, Inception v3 [42] posee convoluciones de 7×7 factorizadas, normalización por lotes en los clasificadores auxiliares, y hace uso de suavizado de etiquetas. La normalización por lotes es una técnica para mejorar el

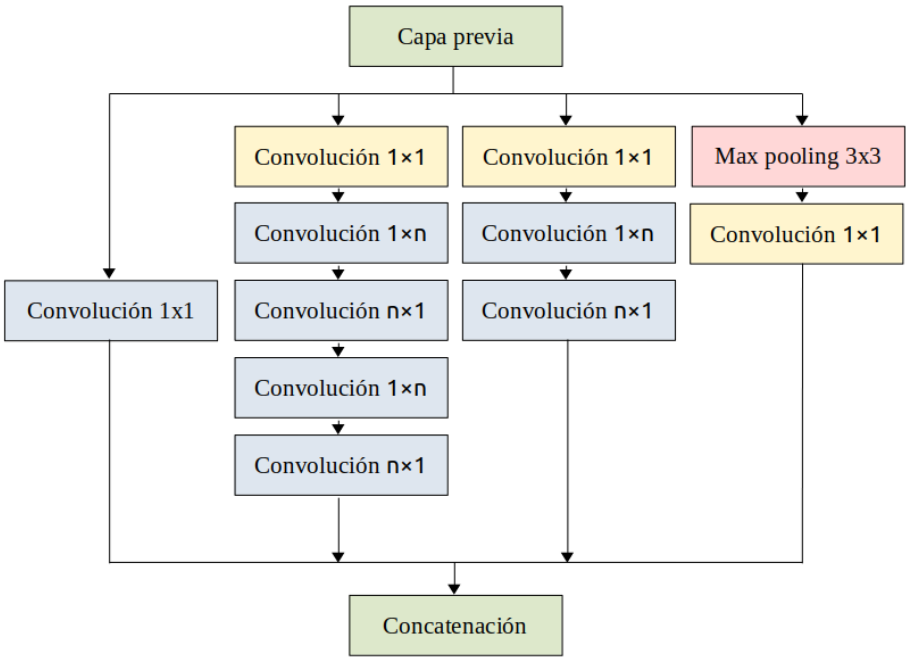

Figura 3.7: Factorización de capas convolucionales en Inception v2 [42].

rendimiento y la estabilidad de las redes neuronales, normaliza la capa de entrada y las capas ocultas, ajustando la media y la escala de las activaciones con el fin de evitar el sobre ajuste (overfitting). El suavizado de etiquetas es un mecanismo para regularizar la capa de clasificación, consiste en asignar un peso a cada clase, en lugar de asignar a la etiqueta correcta el peso completo, la red se sobreajustará menos en las etiquetas de entrenamiento, y así podrá generalizar mejor.

ResNet50

La arquitectura ResNet fue presentada por He et. al. [43] en 2015. La idea detrás de esta arquitectura es atacar el problema de degradación de desempeño en redes neuronales profundas. En su trabajo, los investigadores destacan como el desempeño de una red neuronal de una cierta profundidad se degrada conforme se incrementa el número de capas. La causa detrás de esto es un tema de debate, pero los investigadores denotan que *no* se debe al problema de la gradiente que se desvanece, sino a una incapacidad de los modelos para aprender la función identidad.

La función identidad, $F(x) = x$, en resumen, entrega sin alteraciones la información

que recibe como entrada. Si las capas de una red fuesen capaces de aprender esta función, sería posible agregar una cantidad arbitraria de capas a la red sin que hubiese degradación alguna en su desempeño. Para solucionar este problema, He et. al. proponen el aprendizaje residual. Como se aprecia en la Figura 3.8, la conexión residual se salta las capas subsecuentes. La información que las capas reciben como datos de entrada se suman a sus datos de salida. La hipótesis detrás de esto es que es más sencillo para las capas de la red aprender una función residuo en lugar de la función identidad.

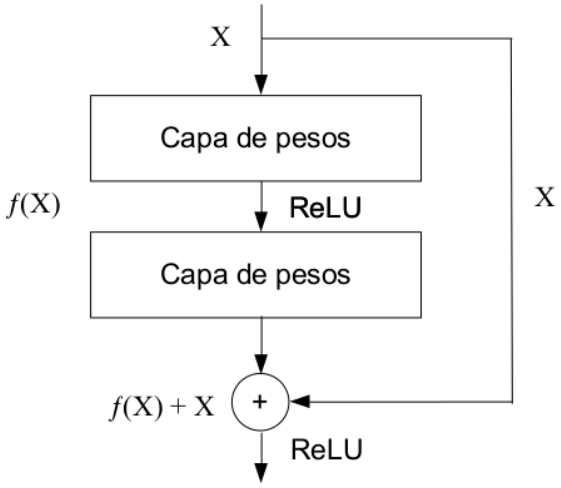

Figura 3.8: Diagrama de una conexión residual [43].

InceptionResNet

Luego de la introducción de las conexiones residuales por parte de He et. al, los investigadores responsables de la arquitectura decidieron combinar la idea con el módulo Inception. El estudio, llevado a cabo en 2016, dio como resultado la arquitectura Inception-ResNet [44]. La arquitectura posee tres módulos Inception rediseñados (A, B y C), menos costosos que sus antecesores. Como se puede apreciar en la Figura 3.9, posee de igual forma una conexión residual que omite las capas intermedias del módulo.

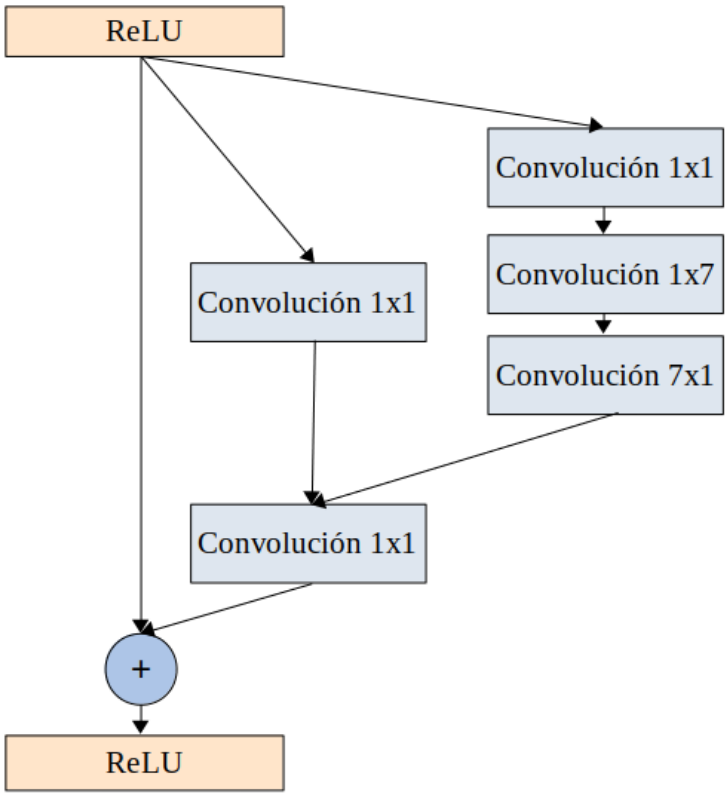

Figura 3.9: Módulo Inception-ResNet-B [44].

Xception

La arquitectura Xception fue presentada por Google en 2017 [45] y como su nombre lo indica, está inspirada en la arquitectura Inception, también desarrollada por Google. El nombre Xception es una abreviación de Xtreme Inception, el cual es un término con el cual los investigadores denominan su versión extrema del módulo Inception.

La arquitectura Xception se basa en convoluciones separables. Específicamente, Xception hace uso de una variante de convoluciones separables en profundidad. Los investigadores mencionan dos puntos que distinguen al módulo Xtreme Inception de las convoluciones separables convencionales:

- El orden de la operaciones. Las convoluciones separables por profundidad convencionales aplican primero convoluciones espaciales a cada canal de la imagen de entrada y posteriormente aplican convoluciones de 1×1 a los mapas de activación concatenados. El módulo Xtreme Inception invierte el orden de estas operaciones:

primero aplica convoluciones de 1×1 a los datos de entrada, y posteriormente aplica convoluciones espaciales a cada mapa de activación individual. Lo anterior se ilustra en la Figura 3.10.

- La presencia o ausencia de una función de activación después de la primer operación. Mientras que, en Inception, ambas operaciones son seguidas de una función de activación ReLU, las convoluciones separables en profundidad usualmente se implementan sin estas.

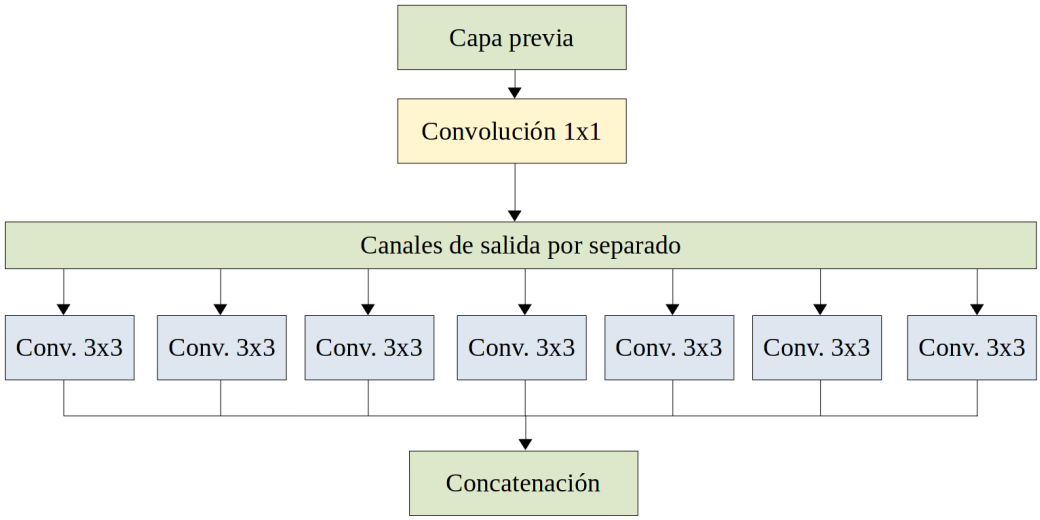

Figura 3.10: Módulo Xception simplificado [45].

DenseNet

La arquitectura DenseNet fue propuesta en 2017 por investigadores de la Universidad de Cornell en conjunto con investigadores de la Universidad de Tsinghua y del departamento de Inteligencia Artificial de Facebook [46]. Lo que distingue a este tipo de RNC es la forma en la que se conectan sus capas convolucionales.

Tradicionalmente, los datos de entrada siguen una ruta mayormente lineal a través de la red. Los datos de salida de la capa L serán los datos de entrada de la capa L+1, mientras que los datos de salida de esta a su vez serán los datos de entrada de la capa L+2, y así sucesivamente. Algunas arquitecturas, tales como Inception o ResNet, presentan arreglos

donde la información se duplica y pasa a través de dos o más capas, pero usualmente la información era procesada de forma distinta para concatenarse al final.

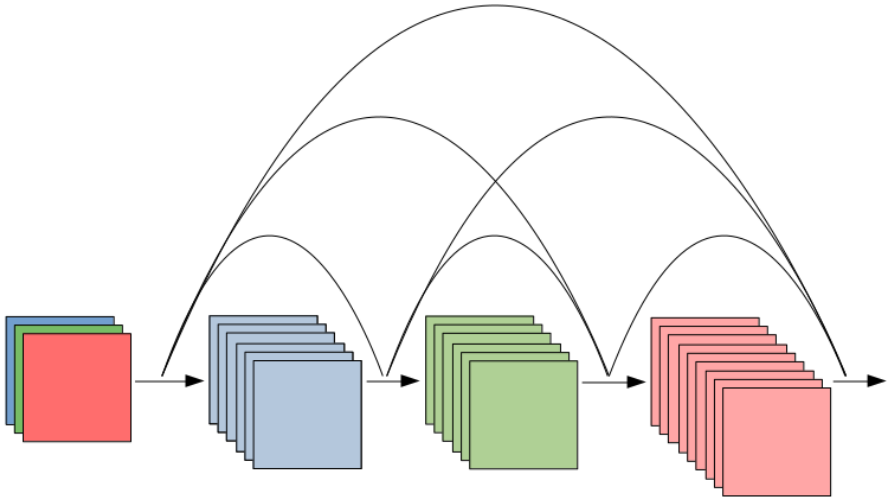

Figura 3.11: Bloque denso de cuatro capas. Basada en una imagen presentada en [46]. Todas las conexiones entre capas van hacia adelante.

La arquitectura DenseNet se conforma principalmente de los llamados bloques densos, en los cuales cada capa de un bloque en particular recibe como entrada los datos de todas las capas en el mismo bloque que le preceden, haciendo uso de concatenación por canales. De esta forma, cada capa dentro de un bloque denso tiene acceso a toda la información de las capas anteriores, lo cual les permite reutilizar características previas según sea necesario. La estructura de estos bloques puede visualizarse en la Figura 3.11, la cual ilustra un bloque denso de cuatro capas, con conexiones entre cada capa L y las capas que le preceden. Por su parte, las configuraciones de las arquitecturas Densenet-121, -169 y -201 se incluyen en la Tabla 3.2.

MobileNet

La primer versión de la arquitectura MobileNet [47] fue propuesta por investigadores de Google, y está orientada a dispositivos móviles y embebidos. Se caracteriza por su uso de la convolución separable en profundidad, o *depthwise-separable convolution*. Antes de hablar sobre la convolución separable en profundidad, es útil recordar la operación de convolución

Capas	Tamaño de salida	DenseNet-121	DenseNet-169	DenseNet-201
Conv	112 x 112	7 x 7 conv, paso 2		
Pool	56 x 56	3 x 3 Agreg. Max,, paso 2		
Densa 1	56 x 56	[1 x 1 conv, 3 x 3 conv] x 6		
Transición 1	56 x 56	1 x 1 conv		
	28 x 28	2 x 2 Agreg. Prom., paso 2		
Densa 2	28 x 28	[1 x 1 conv, 3 x 3 conv] x 12		
Transición 2	28 x 28	1 x 1 conv		
	14 x 14	2 x 2 Agreg. Prom., paso 2		
Densa 3	14 x 14	[1 x 1 conv, 3 x 3 conv] x 24	[1 x 1 conv, 3 x 3 conv] x 32	[1 x 1 conv, 3 x 3 conv] x 48
Transcición 3	14 x 14	1 x 1 conv		
	7 x 7	2 x 2 Agreg. Prom., paso 2		
Densa 4	7 x 7	[1 x 1 conv, 3 x 3 conv] x 16	[1 x 1 conv, 3 x 3 conv] x 32	[1 x 1 conv, 3 x 3 conv] x 32
Clasificación	14 x 14	7 x 7 global Agreg. Prom.		
		1000 neuronas, densa, activación softmax		

Tabla 3.2: Arquitecturas de Densenet-121, -169 y -201. La estructura de las redes es altamente similar, con la excepción de las capas densas 3 y 4, donde el número de filtros utilizados difiere entre las tres.

tradicional.

La convolución bidimensional tradicional opera sobre una ventana de $N \times M$. Sin embargo, también opera sobre todos las dimensiones del tensor de entrada; esto significa que si el tensor de entrada es una imagen RGB (3 canales de profundidad), un kernel de convolución de 5×5 en realidad tendría la forma $5 \times 5 \times 3$. Así, cada operación de convolución realiza $5 \times 5 \times 3 = 75$ operaciones.

La convolución separable en profundidad se constituye de dos operaciones de convolución: convolución por profundidad y convolución por punto. La convolución por profundidad aplica convoluciones por separado para cada dimensión del tensor de entrada. Es decir, una convolución tradicional con un kernel de 5×5 en una imagen a color aplicará una convolución de $5 \times 5 \times 1$ a los tres canales (rojo, verde y azul) de la imagen de entrada. Una vez hecho esto, los mapas de activación de estas convoluciones se concatenan sobre el eje de profundidad, y se aplica una convolución tradicional de 1×1 al tensor resultante (convolución por punto), la cual combina los canales de los mapas de activación concatenados. Al final, el número de operaciones realizadas por la convolución separable en profundidad se reduce a $5 \times 5 + 3 = 28$, lo cual la vuelve significativamente más eficiente.

La arquitectura de la red MobileNet se basa en las convoluciones separables en profun-

didad, con la excepción de su primera capa, la cual es una capa de convolución tradicional. Todas las capas son seguidas de normalización por lotes y cuentan con la función de activación ReLU, excepto la última capa densa. La arquitectura completa se describe en la Tabla 3.3.

Tipo / Paso	Forma de filtro	Tamaño de input
Conv / s2	3 x 3 x 3 x 32	224 x 224 x 3
Conv dw / s1	3 x 3 x 32 dw	112 x 112 x 32
Conv / s1	1 x 1 x 32 x 64	112 x 112 x 32
Conv dw / s2	3 x 3 x 64 dw	112 x 112 x 64
Conv / s1	1 x 1 x 64 x 128	56 x 56 x 64
Conv dw / s1	3 x 3 x 128 dw	56 x 56 x 128
Conv / s1	1 x 1 x 128 x 128	56 x 56 x 128
Conv dw / s2	3 x 3 x 128 dw	56 x 56 x 128
Conv / s1	1 x 1 x 128 x 256	28 x 28 x 128
Conv dw / s1	3 x 3 x 256 dw	28 x 28 x 256
Conv / s1	1 x 1 x 256 x 256	28 x 28 x 256
Conv dw / s2	3 x 3 x 256 dw	28 x 28 x 256
Conv / s1	1 x 1 x 256 x 512	14 x 14 x 256
Conv dw / s1	3 x 3 x 512 dw	14 x 14 x 512
Conv / s1	1 x 1 x 512 x 512	14 x 14 x 512
Conv dw / s2	3 x 3 x 512 dw	14 x 14 x 512
Conv / s1	1 x 1 x 512 x 1024	7 x 7 x 512
Conv dw / s2	3 x 3 x 1024 dw	7 x 7 x 1024
Conv / s1	1 x 1 x 1024 x 1024	7 x 7 x 1024
Agreg. Prom. / s1	Pool 7 x 7	7 x 7 x 1024
Densa / s1	1024 x 1000	1 x 1 x 1024
Softmax / s1	Clasificador	1 x 1 x 1000

Tabla 3.3: Arquitectura de MobileNet

Capítulo 4

Pruebas y análisis de resultados

En este capítulo se describen los experimentos realizados. Específicamente, se describen los conjuntos de datos con los cuales se entrenaron los modelos y los parámetros de entrenamiento. También se presentan las gráficas de desempeño de los modelos durante el entrenamiento y el desempeño final de cada modelo por cada clase individual.

4.1. Descripción de experimentos

Como se mencionó con anterioridad, el conjunto de datos o conjunto de datos utilizado está compuesto de 5560 imágenes distribuidas en 112 clases. Estas 112 clases a su vez corresponden cada una a uno de cuatro órganos (flor, fruto, hoja o tallo) de 75 especies diferentes de plantas nativas de México.

Se realizaron dos tipos de entrenamiento: entrenamiento general y entrenamiento por órgano. Un entrenamiento general consiste en entrenar cada modelo individual para clasificar entre 112 clases disponibles — es decir, los modelos intentan clasificar imágenes de los cuatro órganos establecidos.

Por su parte, un entrenamiento por órgano agrupa las 112 clases por órgano (es decir, se crean grupos con clases únicamente de flores, otro de frutos, otro de tallos y uno más de flores), y los modelos son entrenados individualmente en cada grupo. Esto quiere decir que por cada arquitectura se entrenaron cuatro modelos diferentes: uno orientado a clasificación de imágenes de hojas, otro a imágenes de frutos, otro para tallos y finalmente uno para

flores.

En ambos casos, los modelos fueron entrenados con el algoritmo de Descenso Estocástico de Gradiente, o *Stochastic Gradient Descent*, utilizando valores distintos en los parámetros para cada ronda. Definimos una ronda como el proceso de entrenar al conjunto completo de modelos, todos con los mismos datos de entrenamiento y validación, y utilizando los mismos hiperparámetros.

Para entrenar los modelos, se desarrolló un script de Python que hace uso de la biblioteca Keras, disponible en el repositorio de GitHub `https://github.com/zmoralesc/tesis-maestria.git`. Adicionalmente, el código se incluye en la sección de Apéndices de este documento.

El script desarrollado organiza los datos de entrenamiento, para posteriormente entrenar todos los modelos contemplados para este trabajo. Los hiperparámetros a utilizar durante el entrenamiento son definidos por el usuario, y se aplican de forma igual a todos los modelos. Los modelos utilizados fueron las implementaciones de referencia incluidas por defecto en Keras. Por su parte, el hardware utilizado fue una workstation con sistema operativo Ubuntu 18.04, con 32 GB de RAM, 2TB de almacenamiento secundario, procesador Intel Xeon W-2133 de 12 núcleos a 3.6GHz y una Unidad de Procesamiento de Gráficos (GPU) NVIDIA GTX 1080 con 8GB de NVRAM.

La métrica utilizada para medir la precisión de los modelos, a la cual llamaremos Top-K, consiste en obtener las K clases con mayor índice de confianza con las que responde el modelo. Si la respuesta correcta se encuentra dentro de estas K respuestas, entonces se dice que el modelo acertó. Así, la respuesta Top-1 solo se considera un acierto si la clase correcta corresponde a la clase con mayor índice de confianza, y la respuesta Top-5 se considera un acierto si la clase correcta se encuentra entre las cinco clases con mayor índice de confianza.

4.1.1. Entrenamiento general, primera ronda

Para el entrenamiento general se generaron tres subconjuntos del conjunto de datos general, cada uno con una cierta fracción de las imágenes del conjunto de datos general: uno de entrenamiento con 70 % de las imágenes, uno con 15 % para probar la precisión

Tabla 4.1: Resultados finales de entrenamiento general, primera ronda

	Entrenamiento		Prueba	
Arquitectura	Error	Precisión	Error	Precisión
VGG16	0.0598	0.833	5.188	0.320
VGG19	0.000	1.000	17.677	0.268
INCEPTIONV3	3.105	0.522	5.532	0.265
RESNET50	3.139	0.542	6.131	0.272
INCEPTIONRESNET	1.727	0.683	3.541	0.432
XCEPTION	1.824	0.689	3.909	0.413
DENSENET121	3.973	0.419	5.550	0.249
DENSENET169	4.052	0.440	5.826	0.260
DENSENET201	4.077	0.425	5.734	0.249
MOBILENETV2	1.393	0.687	3.917	0.360

del modelo después de cada iteración, y uno con 15 % para la validación final. Al final, los subconjuntos contenían 3893, 885 y 782 imágenes, respectivamente.

La primer ronda del entrenamiento general consistió en entrenar los modelos por 200 épocas cada uno. Como se mencionó anteriormente se utilizó el algoritmo de Descenso de Gradiente Estocástico, con los valores siguientes: La tasa de aprendizaje (*learning rate*) se estableció en 0.01, impulso (*momentum*) en 0 y decaída de peso (*weight decay*) en 0. Adicionalmente, se incluyó una capa de desactivaciones antes de la capa de clasificación final, con probabilidad de 0.6. Estos valores fueron elegidos a criterio del investigador, como punto de partida para los experimentos.

En la Tabla 4.1 se reportan los valores de error y precisión alcanzados por cada modelo individual en la primer ronda de experimentos, tanto para el conjunto de entrenamiento como el conjunto de validación.

4.1.2. Entrenamiento general, segunda ronda

La segunda ronda del entrenamiento general consistió en entrenar los modelos por 300 épocas cada uno. Se utilizó el algoritmo de Descenso de Gradiente Estocástico, con los valores siguientes: la tasa de aprendizaje (*learning rate*) se estableció en 0.01, impulso

Tabla 4.2: Resultados finales de entrenamiento general, segunda ronda.

	Entrenamiento		Prueba	
Arquitectura	Error	Precisión	Error	Precisión
XCEPTION	2.545	0.448	3.872	0.279
INCEPTIONRESNET	1.835	0.589	3.220	0.350
INCEPTIONV3	2.985	0.410	4.722	0.203
RESNET50	4.354	0.351	6.200	0.204
DENSENET121	3.859	0.305	5.189	0.178
DENSENET169	4.179	0.339	5.350	0.211
DENSENET201	4.567	0.329	5.801	0.196
MOBILENETV2	2.765	0.390	4.211	0.211
VGG16	0.010	1.000	6.136	0.259
VGG19	0.000	1.000	7.645	0.270

(*momentum*) en 0.9 y decaída de peso (*weight decay*) en 0.005. Adicionalmente, se incluyó una capa de desactivaciones antes de la capa de clasificación final, con probabilidad de 0.7. Los valores fueron ligeramente modificados de manera aleatoria con respecto de los valores usados en la primera ronda, con el propósito de buscar hiperparámetros que funcionen suficientemente bien.

En la Tabla 4.2 se reportan los valores de error y precisión alcanzados por cada modelo individual en la segunda ronda de experimentos, tanto para el conjunto de entrenamiento como el conjunto de validación.

4.1.3. Entrenamiento general, tercera ronda

La tercera ronda del entrenamiento general consistió en entrenar los modelos por 150 épocas cada uno. Se utilizó el algoritmo de Descenso de Gradiente Estocástico, con los valores siguientes: la tasa de aprendizaje (*learning rate*) se estableció en 0.01, impulso (*momentum*) en 0 y decaída de peso (*weight decay*) en 0. No se agregaron capas de desactivaciones. Los valores fueron ligeramente modificados de manera aleatoria con respecto de los valores usados en la primera ronda, con el propósito de buscar hiperparámetros que funcionen suficientemente bien.

Tabla 4.3: Resultados finales de entrenamiento general, tercera ronda.

	Entrenamiento		Prueba	
Arquitectura	Error	Precisión	Error	Precisión
XCEPTION	0.621	0.853	2.412	0.534
INCEPTIONRESNET	1.834	0.658	3.402	0.428
INCEPTIONV3	2.626	0.546	4.551	0.348
RESNET50	3.696	0.460	5.282	0.297
DENSENET121	3.229	0.464	4.755	0.311
DENSENET169	3.637	0.450	5.360	0.276
DENSENET201	3.421	0.453	5.021	0.312
MOBILENETV2	1.230	0.722	3.335	0.419
VGG16	0.000	1.000	8.509	0.224
VGG19	0.004	1.000	9.816	0.200

En la Tabla 4.3 se reportan los valores de error y precisión alcanzados por cada modelo individual en la tercera ronda de experimentos, tanto para el conjunto de entrenamiento como el conjunto de validación.

4.1.4. Entrenamiento general, cuarta ronda

La cuarta ronda del entrenamiento general consistió en entrenar los modelos por 150 épocas cada uno. Se utilizó el algoritmo de Descenso de Gradiente Estocástico, con los valores siguientes: la tasa de aprendizaje (*learning rate*) se estableció en 0.01, impulso (*momentum*) en 0.9 y decaída de peso (*weight decay*) en 0.01. No se agregaron capas de desactivaciones. Los valores fueron ligeramente modificados de manera aleatoria con respecto de los valores usados en la primera ronda, con el propósito de buscar hiperparámetros que funcionen suficientemente bien.

En la Tabla 4.4 se reportan los valores de error y precisión alcanzados por cada modelo individual en la cuarta ronda de experimentos, tanto para el conjunto de entrenamiento como el conjunto de validación.

Tabla 4.4: Resultados finales de entrenamiento general, cuarta ronda.

	Entrenamiento		Precisión	
Arquitectura	Error	Precisión	Error	Precisión
XCEPTION	2.829	0.439	3.895	0.297
INCEPTIONRESNET	1.916	0.549	2.791	0.360
INCEPTIONV3	2.876	0.398	4.218	0.212
RESNET50	4.152	0.268	4.862	0.215
DENSENET121	3.112	0.321	3.668	0.251
DENSENET169	3.594	0.317	4.181	0.228
DENSENET201	4.149	0.287	4.598	0.227
MOBILENETV2	2.787	0.345	3.743	0.215
VGG16	0.000	1.000	8.411	0.281
VGG19	0.000	1.000	9.574	0.205

4.1.5. Entrenamiento por órgano

El entrenamiento por órgano consistió en entrenar cuatro grupos de modelos por 300 épocas cada uno, donde cada grupo está orientado a la clasificación de un órgano en específico. Así, tenemos cuatro grupos de 10 modelos (40 modelos en total). Se utilizó el algoritmo de Descenso de Gradiente Estocástico, con los valores siguientes: la tasa e aprendizaje (*learning rate*) se estableció en 0.01, impulso (*momentum*) en 0.9 y decaída de peso (*weight decay*) en 0.005. Adicionalmente, se incluyó una capa de desactivaciones (*dropout*) antes de la capa de clasificación final en cada modelo, con probabilidad de 0.7. Los valores fueron ligeramente modificados de manera aleatoria con respecto de los valores usados en la primera ronda, con el propósito de buscar hiperparámetros que funcionen suficientemente bien.

Así mismo, el conjunto de datos general fue segmentado por órgano. Se asignó un 70 % del conjunto de datos al conjunto de entrenamiento, 15 % para el de pruebas y 15 % para el de validación. En la Tabla 4.5 se describe las cantidades exactas de imágenes que se asignaron a cada uno de los subconjuntos (entrenamiento, pruebas y validación) para los cuatro órganos diferentes. Las Tablas D.1, D.2, D.3 y D.4 presentan los desempeños durante los entrenamientos con los conjuntos de datos de flores, frutos, hojas y tallos, respectivamente.

Tabla 4.5: Cardinalidad de los subconjuntos de datos para cada órgano individual

Órgano	Clases	Entrenamiento	Pruebas	Validación
Flores	62	2160	493	432
Frutos	7	245	56	49
Hojas	38	1314	297	265
Tallos	5	174	39	36

4.2. Análisis de resultados

En general, se realizaron cinco rondas de experimentos: cuatro rondas de entrenamientos generales y una ronda de entrenamientos por órgano. Los resultados finales se han concentrado en gráficas de barras, las cuales se presentan a continuación. Cada arquitectura presenta dos puntajes; Top-1, representado por la barra de color rojo, y Top-5, representada por la barra de color azul.

4.2.1. Entrenamientos generales

En esta sección se discuten los resultados de los entrenamientos generales. Los resultados de la primera ronda se pueden visualizar en la Figura 4.1, los de la segunda ronda en la Figura 4.2, los de la tercera ronda en la Figura 4.3, y finalmente los de la cuarta ronda en la Figura 4.4. Inicialmente, podemos observar que a pesar de usar técnicas muy distintas, muchas de las arquitecturas de desempeñan de manera similar. Creemos que esto se debe a que ninguna de las técnicas utilizadas por sí mismas es suficiente para permitir a una arquitectura sobresalir. En contraste, podemos observar que arquitecturas como Inception-ResNet, las cuales combinan múltiples técnicas de otras arquitecturas, lograron alcanzar precisiones más altas que el resto.

VGG16 y VGG19

Las redes VGG obtuvieron resultados decentes durante las rondas de entrenamiento general, superando incluso a algunos de los modelos más complejos. Sin embargo, y a pesar de poseer una gran cantidad de parámetros, creemos que estos modelos se vieron

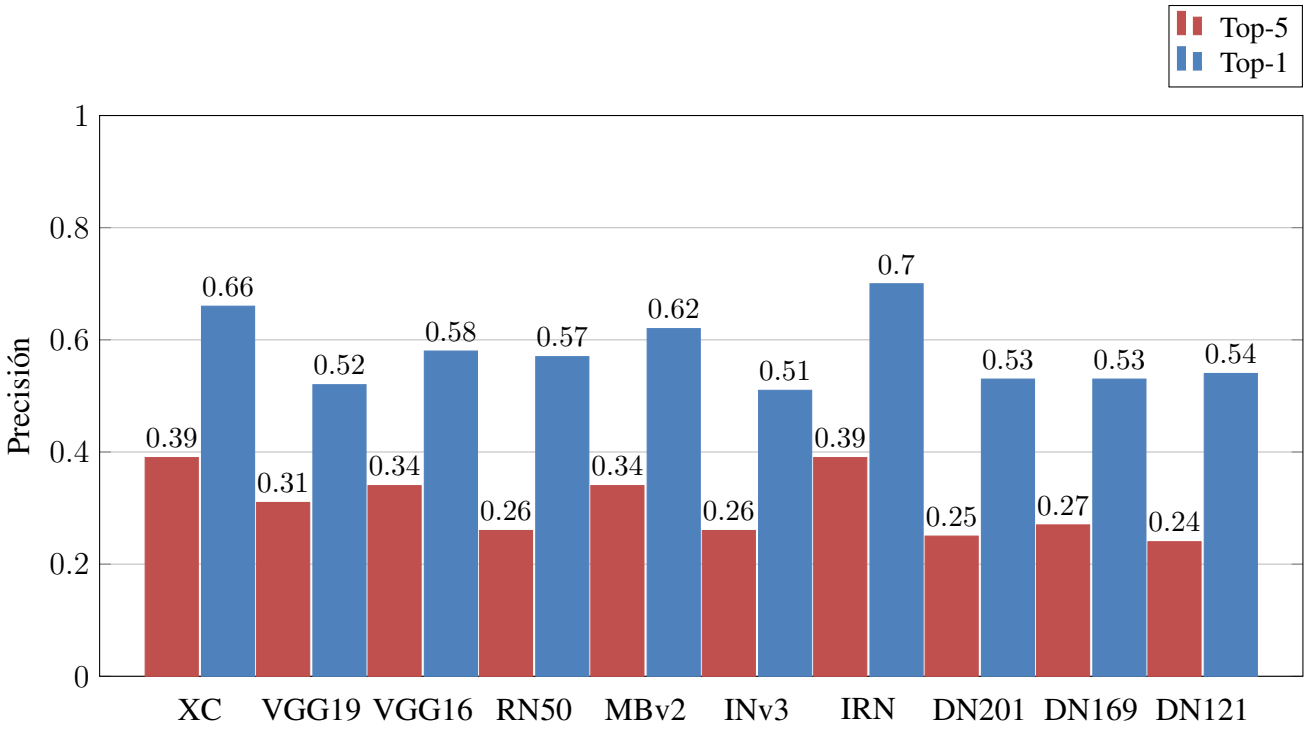

Figura 4.1: Ranking de puntajes Top-1 y Top-5 de arquitecturas, primera ronda.

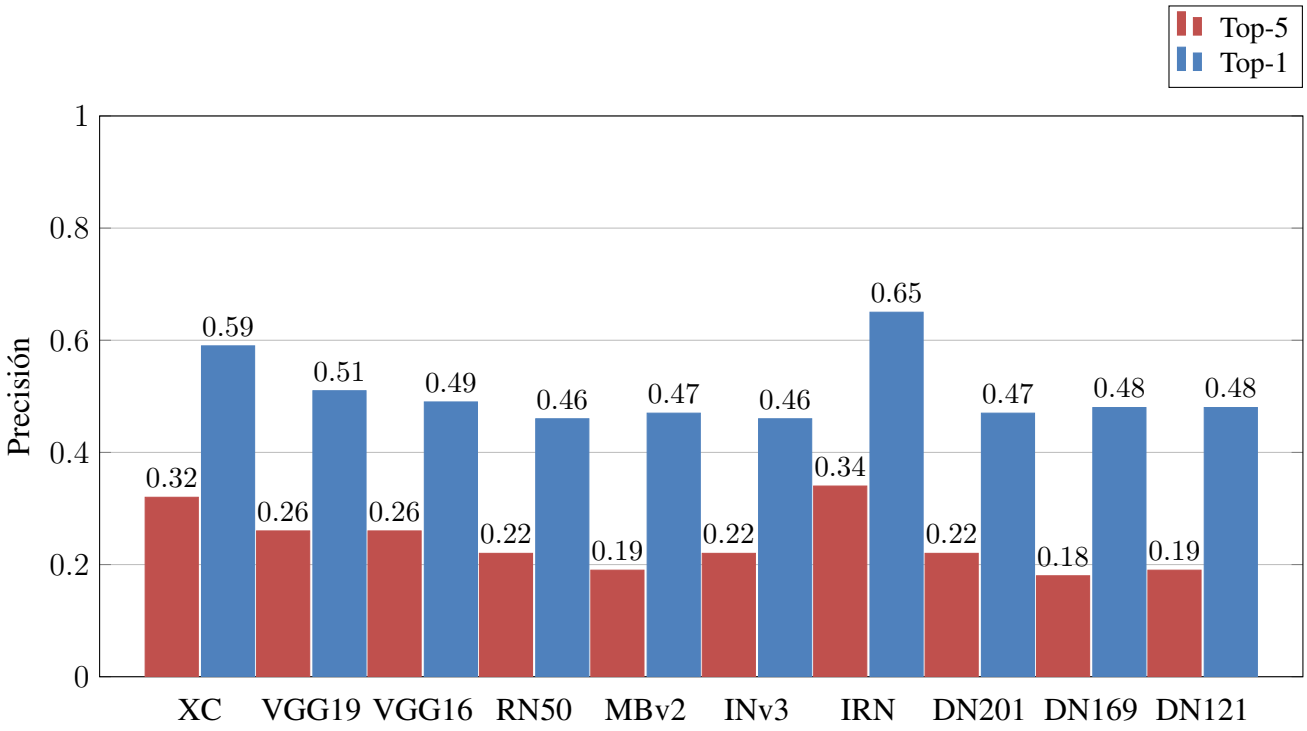

Figura 4.2: Ranking de puntajes Top-1 y Top-5 de arquitecturas, segunda ronda.

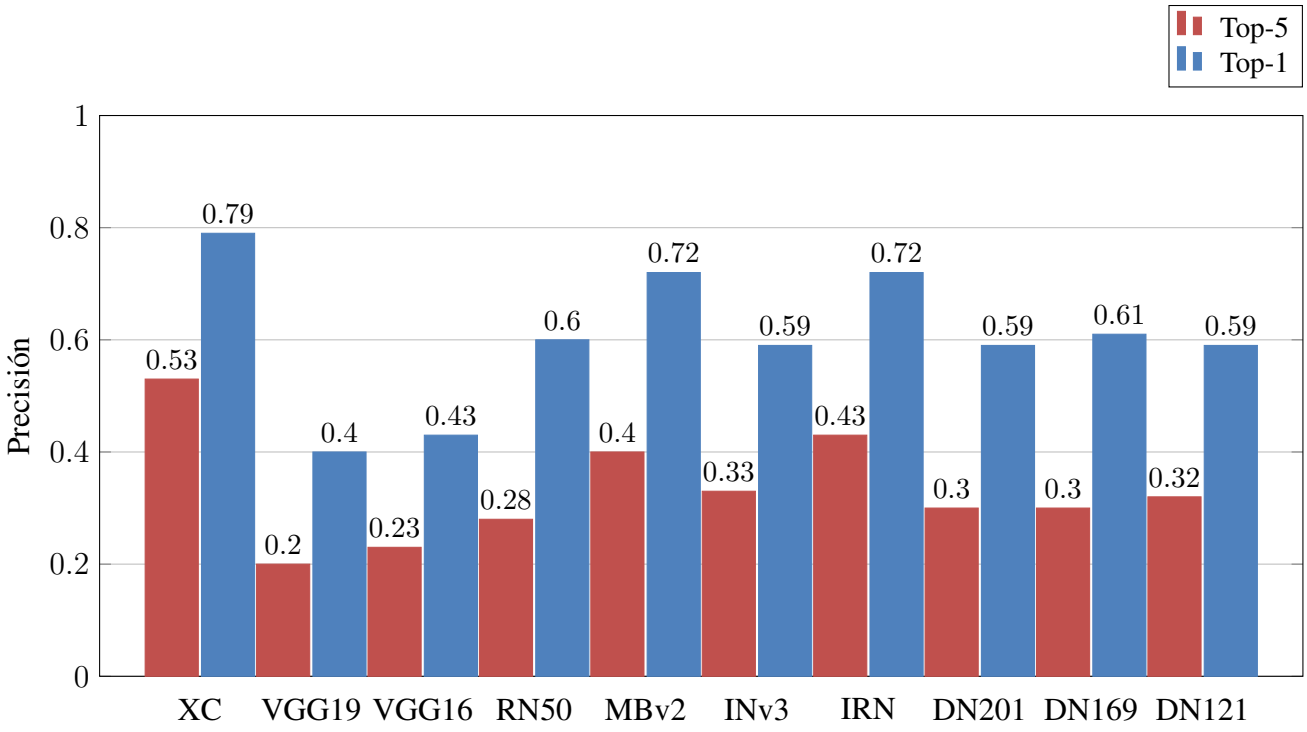

Figura 4.3: Ranking de puntajes Top-1 y Top-5 de arquitecturas, tercera ronda.

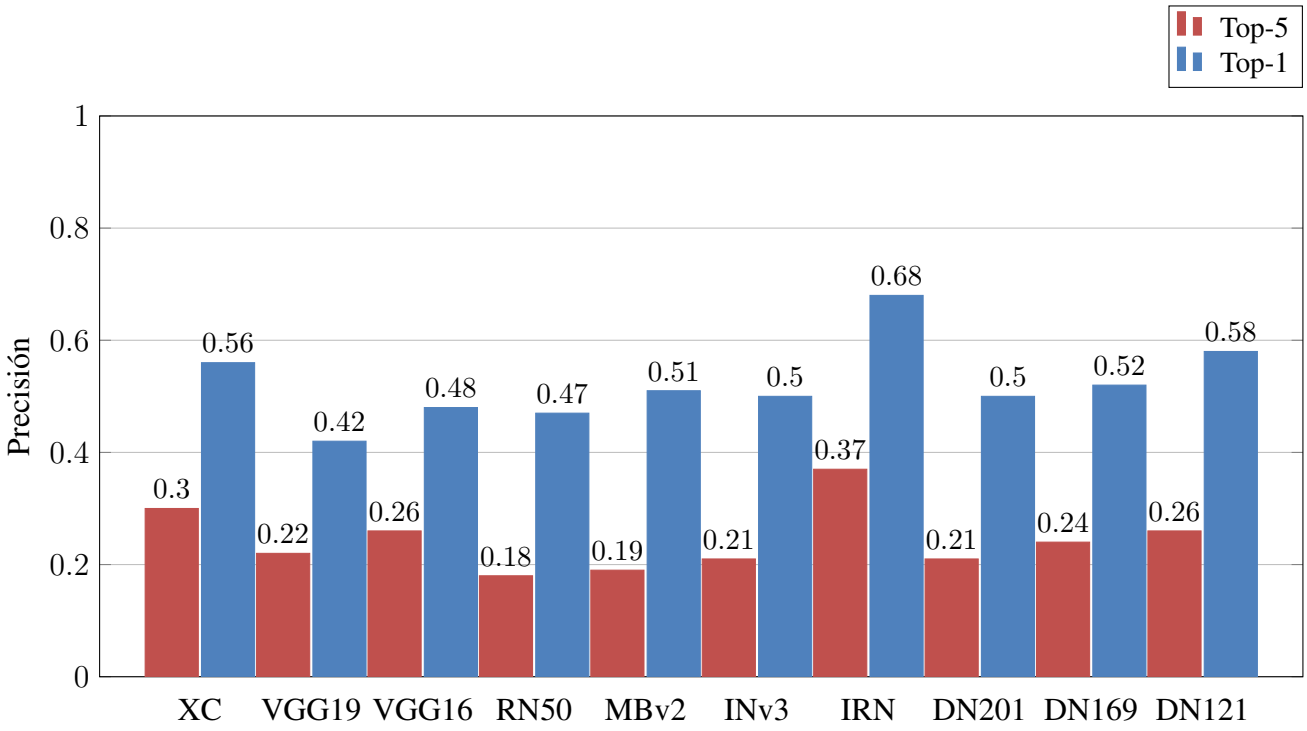

Figura 4.4: Ranking de puntajes Top-1 y Top-5 de arquitecturas, cuarta ronda.

superados debido a que carecen de las técnicas más sofisticadas con las que cuentan los demás modelos analizados.

DenseNet-121, -169 y -201

En el caso de la arquitectura DenseNet, las tres variantes obtuvieron resultados notablemente similares en ambas rondas de los entrenamientos generales. Algo que cabe destacar es el hecho de que las variantes más profundas no se desempeñaron mejor que las demás necesariamente. Durante la primera ronda, DenseNet-121 – la variante menos profunda – alcanzó una precisión Top-5 marginalmente mayor que las dos versiones restantes (0.01). Sin embargo, la precisión Top-1 fue un 0.01 menor que DenseNet-201 y 0.02 menor que DenseNet-169. En ambos casos, DenseNet-201 fue superada por sus dos variantes menos profundas. Esto apoya la noción de que aumentar la profundidad de una red no siempre traerá beneficios. En el caso de DenseNet, creemos que las variantes más profundas son afectadas por el problema del gradiente que se desvanece, lo cual afecta su desempeño en nuestro conjunto de datos.

Xception

La arquitectura Xception fue una de las mejores arquitecturas durante las pruebas preliminares, por lo cual esperábamos verla sobresalir durante las pruebas generales. Si bien obtuvo un desempeño decente (0.32 y 0.66 de precisión Top-1 y Top-5 durante la primera ronda respectivamente, así como 0.32 y 0.59 durante la segunda), en ambos casos se vio superada por InceptionResNet. Creemos que esto se debe a la ventaja que tiene InceptionResNet debido a las conexiones residuales que posee. Estas le permiten incrementar la profundidad de la red de manera substancial sin reducir su desempeño.

ResNet50

La arquitectura ResNet50 fue una de las arquitecturas que no logró sobresalir durante los experimentos preliminares, y aparentemente mantuvo el status quo durante las rondas de entrenamientos generales. La arquitectura alcanzó un desempeño similar a las demás

durante la primera ronda (0.26 de precisión Top-1 y 0.57 de Top-5), sin embargo fue superada por Xception, Mobilenet v2 e InceptionResNet en Top-5 y por casi todas las demás en Top-5, salvo por DenseNet-201, -121 y ambas arquitecturas VGG. La segunda ronda vio una reducción en desempeño para todas las arquitecturas, sin embargo cabe mencionar que de igual manera ResNet50 se vio superada por la gran mayoría de las otras arquitecturas, incluyendo ambas versiones de VGG. Creemos que la complejidad de este modelo fue un factor en su bajo desempeño contra VGG, dado que el riesgo de que un modelo se sobreajuste a los datos de entrenamiento se incrementa con su complejidad. Por otro lado, este modelo pudo verse superado por modelos tales como InceptionResNet y Xception debido a que no implementa las técnicas sofisticadas que caracterizan a estos últimos, como bloques de capas especializados (módulos Inception) o convoluciones separables.

MobileNet v2

Durante la primera ronda, MobileNet v2 alcanzó un desempeño decente (0.34 y 0.62), terminando debajo únicamente de Xception (0.39 y 0.66) e InceptionResNet (0.39 y 0.70). En contraste con las demás arquitecturas, MobileNet v2 presentó una mejora substancial tanto en Top-1 como en Top-5. Sin embargo, la caída general en desempeño de la segunda ronda la posicionó por debajo de varias arquitecturas, más notablemente VGG16 y 19, tanto en Top-1 como Top-5. El desempeño poco sobresaliente de esta arquitectura puede deberse de igual forma a su relativa simplicidad. Las arquitecturas MobileNet fueron diseñadas con dispositivos móviles en mente, por lo que poseen un número relativamente reducido de parámetros en contraste con las demás arquitecturas.

Inception v3

Inception v3 fue otra arquitectura que no logró sobresalir durante ninguno de los entrenamientos. Durante la primera ronda terminó como la quinta mejor arquitectura tanto en Top-1 como en Top-5 (Top-5 empate con ResNet50). La segunda ronda terminó con resultados idénticos a ResNet50, de igual forma debajo de múltiples arquitecturas. En este caso, los módulos Inception resultaron ser insuficientes para darle a esta arquitectura una ventaja sobre las demás. Creemos que esto se debe al relativamente alto número de capas que

posee la red (complejidad), dado que las arquitecturas más sencillas lograron resultados equiparables a pesar de no implementar técnicas tan sofisticadas.

InceptionResNet

En la mayoría de experimentos generales, InceptionResNet se mantuvo como la arquitectura de mejor desempeño, alcanzando puntajes Top-1 y Top-5 de 0.39 y 0.70 durante la primera ronda y 0.34 y 0.65 durante la segunda. Por este motivo, consideramos InceptionResNet como la arquitectura individual más apta para nuestro problema de clasificación. Creemos que el buen desempeño de esta arquitectura es atribuible a la combinación de técnicas que utiliza – esta arquitectura hace uso de módulos Inception, los cuales permiten a una red utilizar distintos tamaños de kernel para detección de características, así como conexiones residuales, las cuales le permiten incrementar su profundidad sin sacrificar desempeño.

4.2.2. Entrenamiento por órgano

Los entrenamientos por órgano produjeron resultados interesantes. Era de esperarse que el desempeño final de cada modelo se incrementara, debido a que el número de clases que cada modelo debía identificar se redujo de forma drástica en algunos casos (ver Tabla 4.5). Sin embargo, no solo vimos arquitecturas más simples superar a las más complejas – la diferencia en precisiones fue bastante acentuada en algunos casos. La discusión de estos resultados se llevará a cabo por tipo de órgano, en lugar de por arquitectura.

Flores

VGG16 obtuvo un desempeño excelente en el dataset de flores, e incluso superó a la mayoría de las demás arquitecturas con puntajes de 0.88 y 0.93, quedando en segundo lugar después de InceptionResNet para Top-5 y siendo la mejor en Top-1. Creemos que la relativamente baja variabilidad en el desempeño de las arquitecturas con este conjunto de datos se debe a que las flores son un órgano bastante distintivo de muchas plantas, puesto que poseen formas y colores muy reconocibles. Los resultados de este entrenamiento se

presentan en la Figura 4.5.

Hojas

El entrenamiento con hojas es interesante considerando que el conjunto de datos contiene menos clases que el conjunto de datos de flores (38 en contraste con 62), y sin embargo las precisiones generales fueron menores que los modelos entrenados con flores. Esto demuestra la dificultad que conlleva identificar especies basándose únicamente en imágenes de hojas. En este entrenamiento, de nueva cuenta VGG16 y VGG19 se desempeñaron mejor que las arquitecturas más complejas. Una posible explicación de esto es que la simplicidad relativa de estas arquitecturas en contraste con las demás les ayuda a mitigar el problema de sobreajuste (*overfitting*). Los resultados de este entrenamiento se presentan en la Figura 4.7.

Frutos

Como era de esperarse debido al reducido número de clases en el conjunto de datos (solo 7), todos los modelos alcanzaron precisiones bastante altas al identificar frutos. Sin embargo, solo cuatro alcanzaron precisión de 1.0 en Top-5: Xception, VGG16, MobileNet v2 e InceptionResNet. Finalmente, VGG16 superó al resto en Top-1, alcanzando una precisión de 0.9. La arquitectura con peor desempeño tanto en Top-1 como Top-5 fue ResNet50, con precisiones de 0.55 y 0.92 respectivamente. Debido al reducido tamaño de este conjunto de datos, es probable que el alto desempeño general de todas las arquitecturas se deba a un caso de sobreajuste. Por este motivo, no es posible extraer conclusiones contundentes en la tarea de clasificación de frutos. Los resultados de este entrenamiento se presentan en la Figura 4.6.

Tallos

Finalmente, el conjunto de datos de tallos fue el más pequeño de todos con solamente 5 clases. Evidentemente el valor de Top-5 es inútil en este caso. En cuanto al Top-1, el mejor desempeño fue obtenido por InceptionResNet con 0.97 de precisión, mientras que la peor

arquitectura fue DenseNet-201 con 0.78. Al igual que con el conjunto de frutos, es altamente probable que los modelos se hayan sobreajustado al conjunto de datos utilizado. Por este motivo tampoco es posible extraer conclusiones contundentes en la tarea de clasificación de tallos. Los resultados de este entrenamiento se presentan en la Figura 4.8.

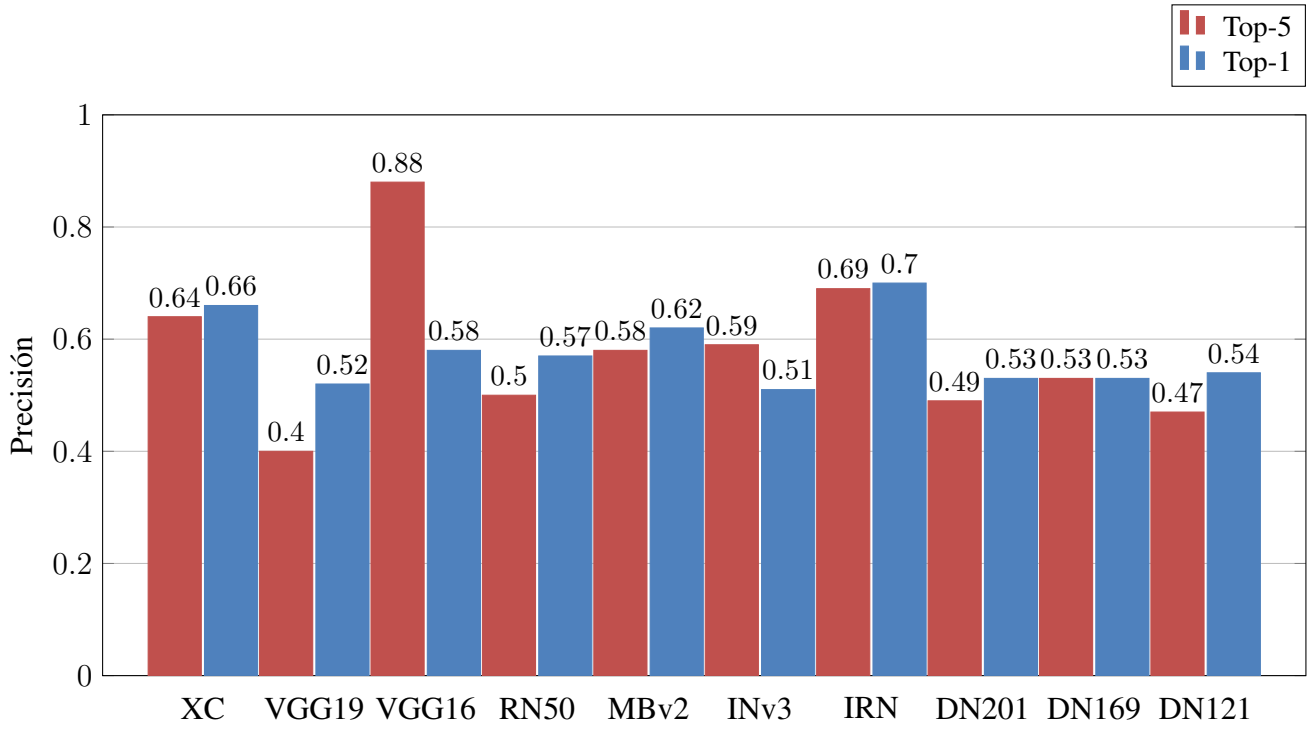

Figura 4.5: Ranking de puntajes Top-1 y Top-5 de arquitecturas por órgano, flores.

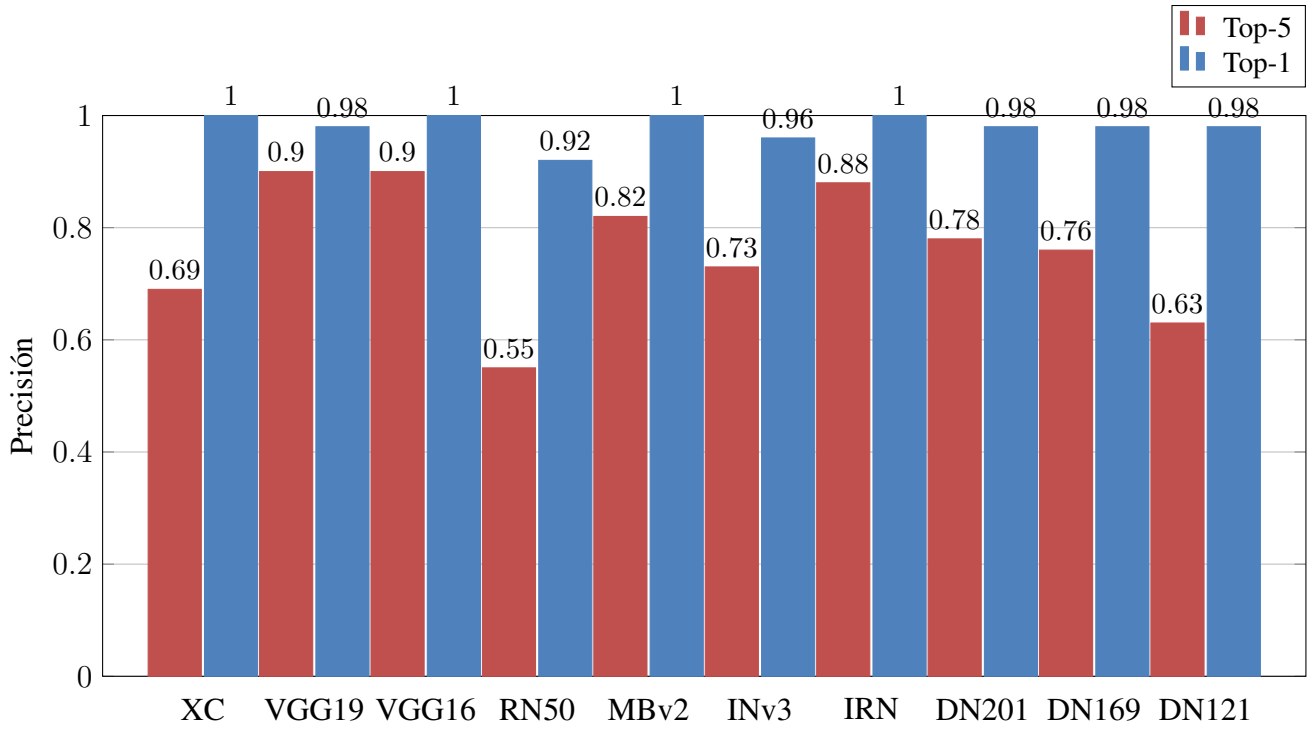

Figura 4.6: Ranking de puntajes Top-1 y Top-5 de arquitecturas por órgano, frutos.

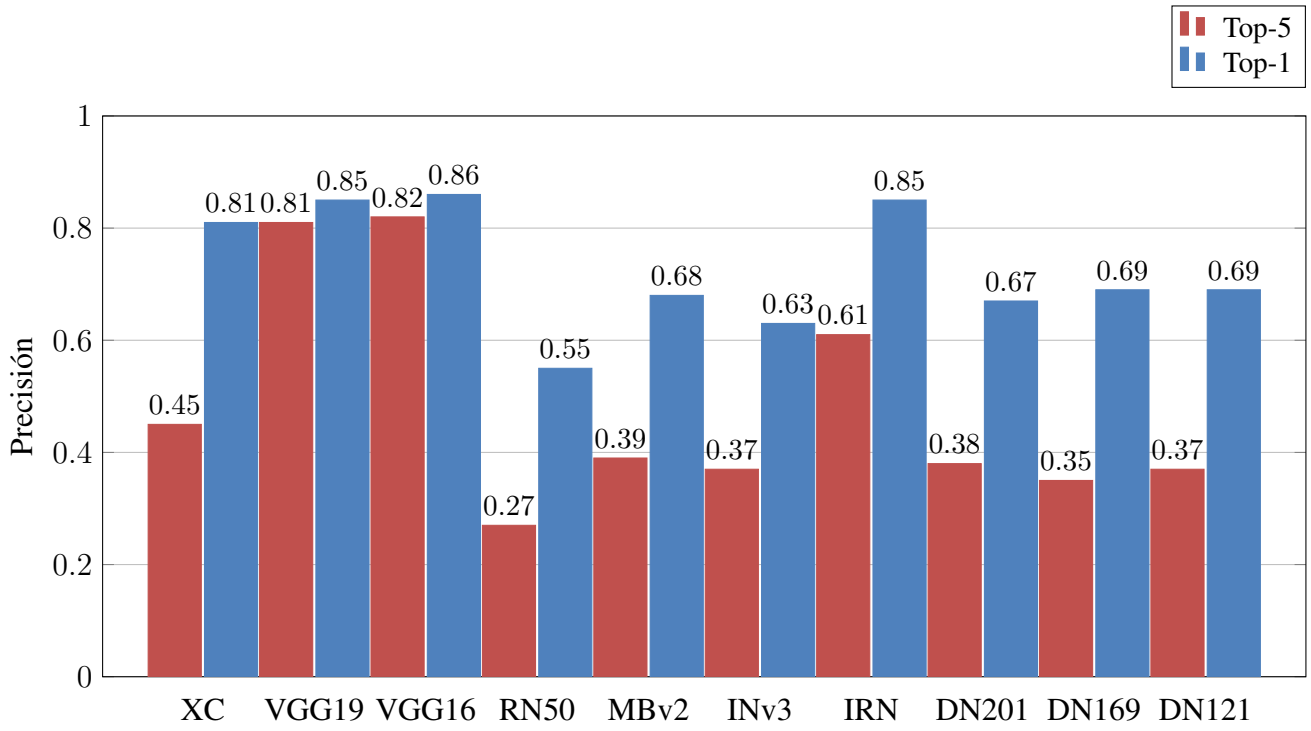

Figura 4.7: Ranking de puntajes Top-1 y Top-5 de arquitecturas por órgano, hojas.

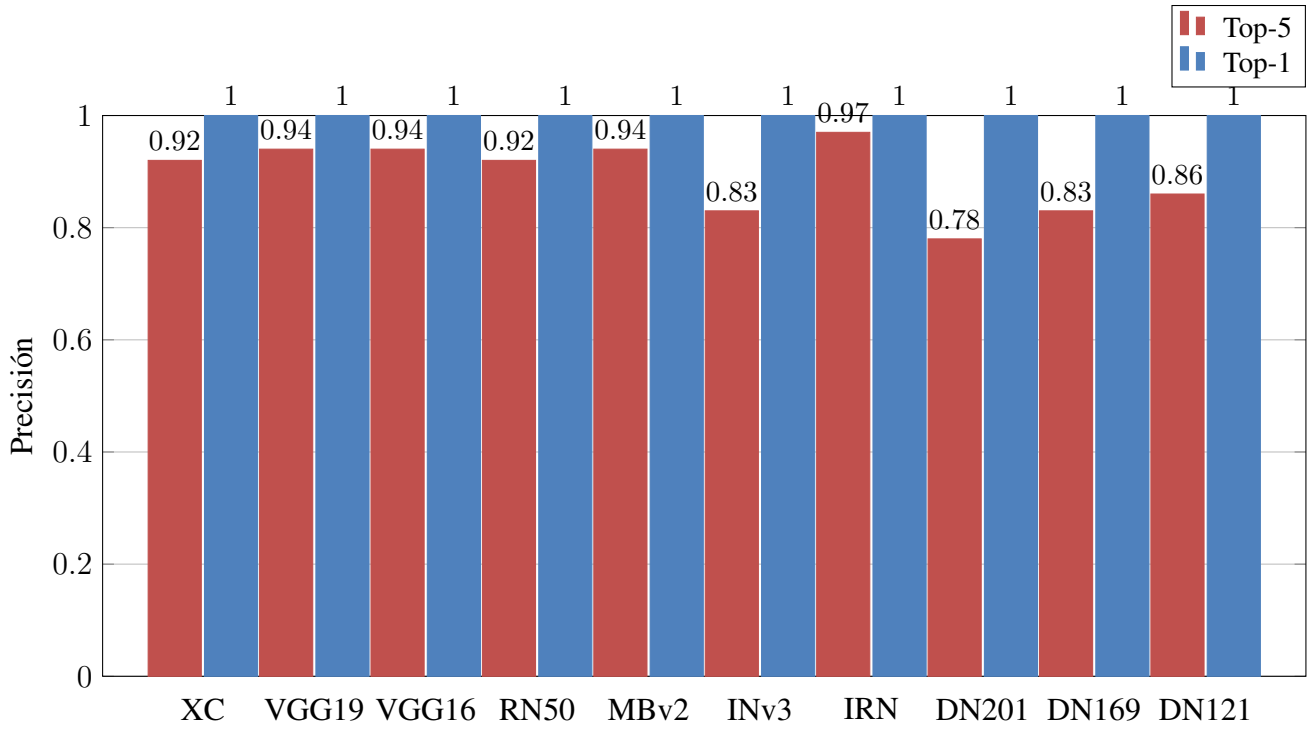

Figura 4.8: Ranking de puntajes Top-1 y Top-5 de arquitecturas por órgano, tallos.

Capítulo 5

Conclusiones

En esta tesis se realizó un análisis de diez arquitecturas de CNN distintas para clasificación de especies vegetales nativas de México. El objetivo principal fue atacar el problema de la clasificación automática de plantas, el cual es significativamente más complejo que los problemas de clasificación comunes debido no solo a la gran cantidad de especies diferentes que existen, si no también al alto grado de variabilidad entre miembros de una misma especie, así como las similitudes que se presentan entre especies distintas. En concreto:

- Se desarrolló una aplicación con C++ y el framework Qt para facilitar el proceso de recorte y etiquetado de imágenes.
- Se desarrollaron scripts en Python para automatizar la ejecución de experimentos.
- Se ejecutaron tres rondas de experimentos: dos entrenamientos generales con el conjunto de datos completo (112 clases) y una en donde se segmentó el conjunto de datos por órgano.

Dado que el número de clases en el conjunto de datos es relativamente pequeño, no es posible declarar una arquitectura como superior a las demás. Sin embargo, es posible observar ciertas arquitecturas que se desempeñan mejor de forma consistente, como es el caso de InceptionResNet. Esto nos permitirá enfocarnos en las mejores arquitecturas conforme el conjunto de datos crezca, y omitir aquellas que fueron incapaces de sobresalir, ahorrando tiempo.

Tomando en cuenta la complejidad de las arquitecturas que lograron sobresalir, es posible concluir que el número de parámetros de una arquitectura de RNC, si bien es un factor que influye en la capacidad de una red para resolver un problema, no es la cualidad más importante en un modelo de RNC cuando se utilizan para realizar clasificación de especies de plantas. Esto es evidente cuando se contrastan los resultados de las arquitecturas VGG19 y Xception en, por ejemplo, la tercera ronda: con tan solo una fracción del número de parámetros que posee VGG19 (143,667,240 de VGG19 contra 22,910,480 de Xception), Xception alcanzó un desempeño significativamente mejor en el mismo conjunto de datos. Al mismo tiempo, si bien es verdad que Xception es una arquitectura mucho más profunda que VGG19 (126 capas en contraste con 26), la profundidad por sí sola no es factor decisivo. Esto se puede observar en los casos de las arquitecturas DenseNet, donde la variante más profunda, DenseNet-201, se desempeñó igual y en algunos casos peor que las variantes menos profundas (DenseNet-121 y -169).

Como líneas de trabajo futuro, se identificaron los siguientes puntos:

- Realizar entrenamientos de las arquitecturas con conjuntos de datos más extensos. Debido a la falta de recursos, el conjunto de datos utilizado en este trabajo resultó relativamente limitado.

- Atacar el problema de desbalance de clases en el conjunto de datos utilizado. Existe un alto grado de desbalance de clases en el conjunto de datos utilizado, producto también de los recursos limitados con los que se contaba durante la realización de este trabajo.

- Utilizar las arquitecturas de RNC que hayan sido propuestas posteriormente al desarrollo de este trabajo. Nuevas técnicas y arquitecturas se proponen constantemente, y puede resultar potencialmente útil aplicar estos nuevos desarrollos al problema de clasificación de plantas.

Bibliografía

[1] S. H. Lee, Y. L. Chang, C. S. Chan y P. Remagnino, "Plant Identification System based on a Convolutional Neural Network for the LifeClef 2016 Plant Classification Task," en *CLEF*, 2016, págs. 502-510.

[2] B. Yanikoglu, E. Aptoula y C. Tirkaz, "Automatic Plant Identification from Photographs," *Machine vision and applications*, vol. 25, n.º 6, págs. 1369-1383, 2014.

[3] S. T. Hang, A. Tatsuma y M. Aono, "Bluefield (KDE TUT) at LifeCLEF 2016 Plant Identification Task," en *CLEF*, 2016.

[4] A. Krizhevsky, I. Sutskever y G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," en *Advances in Neural Information Processing Systems*, F. Pereira, C. J. C. Burges, L. Bottou y K. Q. Weinberger, eds., vol. 25, Curran Associates, Inc., 2012, págs. 1097-1105.

[5] *Biodiversidad: Conocer para Conservar*. Secretaría de Medio Ambiente y Recursos Naturales, 2011.

[6] J. Krause, G. Sugita, K. Baek y L. Lim, "WTPlant (What's That Plant?): A Deep Learning System for Identifying Plants in Natural Images," en *Proceedings of the 2018 ACM on International Conference on Multimedia Retrieval*, ACM, 2018, págs. 517-520.

[7] N. Kumar, P. N. Belhumeur, A. Biswas, D. W. Jacobs, W. J. Kress, I. C. Lopez y J. V. B. Soares, "Leafsnap: A Computer Vision System for Automatic Plant Species Identification," en *Proceedings of the European Conference on Computer Vision*, A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato y C. Schmid, eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, págs. 502-516, ISBN: 978-3-642-33709-3.

[8] Z.-Q. Zhao, L.-H. Ma, Y.-m. Cheung, X. Wu, Y. Tang y C. L. P. Chen, "ApLeaf: An Efficient Android-based Plant Leaf Identification System," *Neurocomputing*, vol. 151, págs. 1112-1119, 2015.

[9] J. Wäldchen y P. Mäder, "Plant Species Identification using Computer Vision Techniques: A Systematic Literature Review," *Archives of Computational Methods in Engineering*, vol. 25, n.º 2, págs. 507-543, 2018.

[10] J. Wäldchen, M. Rzanny, M. Seeland y P. Mäder, "Automated Plant Species Identification—Trends and Future Directions," *PLOS Computational Biology*, vol. 14, n.º 4, págs. 1-19, abr. de 2018. DOI: 10.1371/journal.pcbi.1005993. dirección: https://doi.org/10.1371/journal.pcbi.1005993.

[11] W. S. McCulloch y W. Pitts, "A Logical Calculus of the Ideas Immanent in Nervous Activity," *The bulletin of mathematical biophysics*, vol. 5, n.º 4, págs. 115-133, 1943.

[12] P. J. Werbos, *The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting (Adaptive and Cognitive Dynamic Systems: Signal Processing, Learning, Communications and Control)*. Wiley-Interscience, 1994, ISBN: 0471598976.

[13] F. Rosenblatt, "The Perceptron - A Perceiving and Recognizing Automaton," Cornell Aeronautical Laboratory, Ithaca, New York, inf. téc. 85-460-1, ene. de 1957.

[14] D. P. Kingma y J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[15] Y. LeCun, "Generalization and Network Design Strategies," en *Connectionism in Perspective*, R. Pfeifer, Z. Schreter, F. Fogelman y L. Steels, eds., Zurich, Switzerland: Elsevier, 1989.

[16] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard y L. Jackel, "Handwritten Digit Recognition with a Back-Propagation Network," en *Advances in Neural Information Processing Systems*, D. Touretzky, ed., vol. 2, Morgan-Kaufmann, 1990, págs. 396-404.

[17] A. Rakitianskaia y A. Engelbrecht, "Measuring Saturation in Neural Networks," en *Proceedings of the IEEE Symposium Series on Computational Intelligence*, IEEE, 2015, págs. 1423-1430.

[18] S. H. Lee, C. S. Chan, P. Wilkin y P. Remagnino, "Deep-plant: Plant Identification with Convolutional Neural Networks," en *Proceedings of the IEEE International Conference on Image Processing*, 2015, págs. 452-456. DOI: 10.1109/ICIP.2015.7350839.

[19] A. K. Reyes, J. C. Caicedo y J. E. Camargo, "Fine-tuning Deep Convolutional Networks for Plant Recognition.," en *CLEF*, 2015.

[20] M. Mehdipour Ghazi, B. Yanıkoğlu y E. Aptoula, "Open-set Plant Identification using an Ensemble of Deep Convolutional Neural Networks," CLEF, 2016.

[21] A. Toma, L. Stefan y B. Ionescu, "UPB HES SO @ PlantCLEF 2017: Automatic Plant Image Identification using Transfer Learning via Convolutional Neural Networks," *Working Notes of CLEF*, vol. 2017, 2017.

[22] P. Pawara, E. Okafor, L. Schomaker y M. Wiering, "Data Augmentation for Plant Classification," en *Proceedings of the International Conference on Advanced Concepts for Intelligent Vision Systems*, Springer, 2017, págs. 615-626.

[23] T. Munisami, M. Ramsurn, S. Kishnah y S. Pudaruth, "Plant Leaf Recognition Using Shape Features and Colour Histogram with K-nearest Neighbour Classifiers," *Procedia Computer Science*, vol. 58, págs. 740-747, 2015, Second International Symposium on Computer Vision and the Internet (VisionNet'15), ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2015.08.095.

[24] P. Pawara., E. Okafor., O. Surinta., L. Schomaker. y M. Wiering., "Comparing Local Descriptors and Bags of Visual Words to Deep Convolutional Neural Networks for Plant Recognition," en *Proceedings of the* 6_{th} *International Conference on Pattern Recognition Applications and Methods*, INSTICC, vol. 1, SciTePress, 2017, págs. 479-486, ISBN: 978-989-758-222-6. DOI: 10.5220/0006196204790486.

[25] *Swedish Leaf Dataset*, http://www.cvl.isy.liu.se/en/research/datasets/swedish-leaf/, (Accessed on 02/14/2021).

[26] P. Barré, B. C. Stöver, K. F. Müller y V. Steinhage, "LeafNet: A Computer Vision System for Automatic Plant Species Identification," *Ecological Informatics*, vol. 40, págs. 50-56, 2017.

[27] M. Carpentier, P. Giguère y J. Gaudreault, "Tree Species Identification from Bark Images Using Convolutional Neural Networks," en *Proceedings of the International Conference on Intelligent Robots and Systems*, 2018, págs. 1075-1081. DOI: 10.1109/IROS.2018.8593514.

[28] S. H. Lee, C. S. Chan y P. Remagnino, "Multi-Organ Plant Classification Based on Convolutional and Recurrent Neural Networks," *IEEE Transactions on Image Processing*, vol. 27, n.º 9, págs. 4287-4301, 2018.

[29] O. Söderkvist, "Computer Vision Classification of Leaves from Swedish Trees," Tesis doct., 2001. dirección: http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-54366.

[30] X.-Y. Xiao, R. Hu, S.-W. Zhang y X.-F. Wang, "HOG-based Approach for Leaf Classification," en *Proceedings of the International Conference on Intelligent Computing*, Springer, 2010, págs. 149-155.

[31] S. G. Wu, F. S. Bao, E. Y. Xu, Y.-X. Wang, Y.-F. Chang y Q.-L. Xiang, "A Leaf Recognition Algorithm for Plant Classification using Probabilistic Neural Network," en *Proceedings of the IEEE International Symposium on Signal Processing and Information Technology*, IEEE, 2007, págs. 11-16.

[32] Q.-K. Nguyen, T.-L. Le y N.-H. Pham, "Leaf based Plant Identification System for Android using SURF Features in Combination with Bag of Words Model and Supervised Learning," en *Proceedings of the International Conference on Advanced Technologies for Communications*, IEEE, 2013, págs. 404-407.

[33] Y. Sun, Y. Liu, G. Wang y H. Zhang, "Deep Learning for Plant Identification in Natural Environment," *Computational Intelligence and Neuroscience*, vol. 2017, págs. 1-6, mayo de 2017. DOI: 10.1155/2017/7361042.

[34] Z. Wang, B. Lu, Z. Chi y D. Feng, "Leaf Image Classification with Shape Context and SIFT Descriptors," en *Proceedings of the International Conference on Digital Image Computing: Techniques and Applications, pages=650–654*, IEEE, 2011.

[35] N. Liu y J.-m. Kan, "Improved Deep Belief Networks and Multi-Feature Fusion for Leaf Identification," *Neurocomputing*, vol. 216, págs. 460-467, 2016.

[36] M. Seeland, M. Rzanny, N. Alaqraa, J. Wäldchen y P. Mäder, "Plant Species Classification using Flower Images—A Comparative Study of Local Feature Representations," *PLOS ONE*, vol. 12, n.º 2, págs. 1-29, feb. de 2017. DOI: 10.1371/journal.pone.0170629.

[37] G.-S. Xie, X.-Y. Zhang, S. Yan y C.-L. Liu, "SDE: A Novel Selective, Discriminative and Equalizing Feature Representation for Visual Recognition," *International Journal of Computer Vision*, vol. 124, n.º 2, págs. 145-168, 2017.

[38] P. Koniusz, F. Yan, P.-H. Gosselin y K. Mikolajczyk, "Higher-order Occurrence Pooling for Bags-of-words: Visual Concept Detection," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, n.º 2, págs. 313-326, 2016.

[39] G.-S. Xie, X.-Y. Zhang, W. Yang, M. Xu, S. Yan y C.-L. Liu, "LG-CNN: From Local Parts to Global Discrimination for Fine-grained Recognition," *Pattern Recognition*, vol. 71, págs. 118-131, 2017.

[40] K. Simonyan y A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," en *Proceedings of the* 3_{rd} *International Conference on Learning Representations, Conference Track*, Y. Bengio e Y. LeCun, eds., San Diego, CA, USA, mayo de 2015. dirección: http://arxiv.org/abs/1409.1556.

[41] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke y A. Rabinovich, "Going Deeper with Convolutions," en *Proceedings of the IEEE*

Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, oct. de 2015, págs. 1-9.

[42] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens y Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," en *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, págs. 2818-2826.

[43] K. He, X. Zhang, S. Ren y J. Sun, "Deep Residual Learning for Image Recognition," en *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Las Vegas, NV, USA, jun. de 2016, págs. 770-778. DOI: 10.1109/CVPR.2016.90.

[44] C. Szegedy, S. Ioffe, V. Vanhoucke y A. A. Alemi, "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning," en *Proceedings of the* 31_{st} *AAAI Conference on Artificial Intelligence*, ép. AAAI'17, San Francisco, California, USA: AAAI Press, 2017, págs. 4278-4284.

[45] F. Chollet, "Xception: Deep Learning with Depthwise Separable Convolutions," en *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, págs. 1251-1258.

[46] G. Huang, Z. Liu, L. Van Der Maaten y K. Q. Weinberger, "Densely Connected Convolutional Networks," en *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, págs. 4700-4708.

[47] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto y H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *arXiv preprint arXiv:1704.04861*, abr. de 2017.

Anexo A

Código

A.0.1. Script de entrenamiento principal

```
#!/usr/bin/env python
def paramDictFormat(d: dict):
    'Formats dictionary into a comma-separated key=value style string'
    return ', '.join([f'{key}={val:.5f}' if isinstance(val, float) else f'{key}={val}' for key, val in d.items()])

def main():
    'Main'
    from argparse import ArgumentParser
    import os

    # available architectures
    models_list = [
        'vgg16',
        'vgg19',
        'inceptionv3',
        'resnet50',
        'custom',
        'xception',
        'inceptionresnet',
        'mobilenet',
        'densenet121',
        'densenet169',
        'densenet201',
        'nasnet',
        'mobilenetv2'
    ]

    # available optimizers
    optimizers_list = ['sgd', 'adam']

    losses_list = [
        'categorical_crossentropy',
        'sparse_categorical_crossentropy',
        'binary_crossentropy',
        'mean_squared_error',
        'mean_absolute_error',
        'mean_absolute_percentage_error',
        'mean_squared_logarithmic_error',
        'squared_hinge',
        'hinge',
        'categorical_hinge',
        'logcosh',
        'kullback_leibler_divergence',
        'poisson',
        'cosine_proximity'
    ]
    # print these names for loss functions
    losses_dict = {
        'mean_squared_error': 'MSE',
        'mean_absolute_error': 'MAE',
        'mean_absolute_percentage_error': 'MAPE',
        'mean_squared_logarithmic_error': 'MSLE',
        'squared_hinge': 'Squared Hinge',
        'hinge': 'Hinge',
        'categorical_hinge': 'Categorical Hinge',
        'logcosh': 'Log-Cosh',
        'categorical_crossentropy': 'Categorial Cross-entropy',
        'sparse_categorical_crossentropy': 'Sparse Categorical Cross-entropy',
        'binary_crossentropy': 'Binary Cross-entropy',
        'kullback_leibler_divergence': 'Kullback-Leibler Divergence',
        'poisson': 'Poisson',
        'cosine_proximity': 'Cosine Proximity'
    }

    parser = ArgumentParser()
    parser.add_argument('model', help='which model to use',
                        type=str, choices=models_list)
    parser.add_argument('path', help='path to data', type=str)
    parser.add_argument('--loadfrom', help='load previous model', type=str)
    parser.add_argument(
        '-e', '--epochs', help='epochs to train for', type=int, default=30)
    parser.add_argument(
        '-b', '--batch', help='training batch size', type=int, default=8)
```

```
parser.add_argument('-o', '--optimizer', help='optimizer to use',
                    type=str, choices=optimizers_list, default='sgd')
parser.add_argument(
    '-s', '--split', help='test split size', default=0.2, type=float)
parser.add_argument('-t', '--testset', help='path to test data', type=str)
parser.add_argument('--loss', help='loss function to use', type=str,
                    choices=losses_list, default='categorical_crossentropy')
parser.add_argument('--nogpu', help='disable GPU',
                    action='store_true', dest='no_gpu')
parser.add_argument('--usemp', help='enable multiprocessing for sequences',
                    action='store_true', dest='use_mp')
parser.add_argument(
    '--pretrained', help='load pre-trained weights', action='store_true')
parser.add_argument('--output', help='output file', type=str)
parser.add_argument('--extsum', help='print extended summary',
                    action='store_true', dest='print_extsum')
parser.add_argument('--sum', help='print summary',
                    action='store_true', dest='print_sum')
parser.add_argument('--json', help='save model as JSON file', type=str)
parser.add_argument('--log', help='test log filename',
                    type=str, default='tests_log.log')
parser.add_argument(
    '--shape', help='input shape in (height:width:depth) format', type=str)
parser.add_argument(
    '--dropout', help='dropout probability (default=0)', type=float)
parser.add_argument('--pfldir', help='put .pfl files here',
                    type=str, default='.', dest='pfldir')
parser.add_argument('--decay', help='weight decay',
                    default=0.005, type=float, dest='weight_decay')
parser.add_argument('-K', help='apply kernel regularization',
                    action='store_true', dest='regularize_kernel')
parser.add_argument('-B', help='apply bias regularization',
                    action='store_true', dest='regularize_bias')
parser.add_argument('-A', help='apply activity regularization',
                    action='store_true', dest='regularize_activity')
parser.add_argument(
    '-a', '--augment', help='apply perform data augmentation', action='store_true')
parser.add_argument(
    '--seed', help='random seed for train-test split', type=int, default=7)
args = parser.parse_args()

from keras.callbacks import Callback

if args.pfldir:
    os.makedirs(args.pfldir, exist_ok=True)
if args.output:
    os.makedirs(args.output, exist_ok=True)

class PerformanceHistory(Callback):
    def __init__(self, output_file: str):
        super().__init__()
        self.output_file = open(output_file, 'wt')
        self.csv_file = csv.writer(self.output_file)
        self.csv_file.writerow(
            ['ACCURACY', 'LOSS', 'VALIDATION_ACCURACY', 'VALIDATION_LOSS'])

    def on_epoch_end(self, batch, logs={}):
        self.csv_file.writerow([logs.get('accuracy'), logs.get(
            'loss'), logs.get('val_accuracy'), logs.get('val_loss')])
        self.output_file.flush()

    def __del__(self):
        self.output_file.close()

if args.no_gpu:
    os.environ['CUDA_DEVICE_ORDER'] = 'PCI_BUS_ID'
    os.environ['CUDA_VISIBLE_DEVICES'] = ''

image_dim = 224 if args.model in ['vgg16', 'vgg19', 'custom'] else 299
input_shape = (image_dim, image_dim, 3)
if args.shape is not None:
    input_shape = [(int(y) if y != '' else 1)
                   for y in args.shape.split(':')]

import keras.applications as apps

# preprocessing functions dictonary
input_preprocessing = {
    'vgg16': apps.vgg16.preprocess_input,
    'vgg19': apps.vgg19.preprocess_input,
    'inceptionv3': apps.inception_v3.preprocess_input,
    'resnet50': apps.resnet50.preprocess_input,
    'custom': apps.vgg16.preprocess_input,
    'xception': apps.xception.preprocess_input,
    'inceptionresnet': apps.inception_resnet_v2.preprocess_input,
    'mobilenet': apps.mobilenet.preprocess_input,
    'densenet121': apps.densenet.preprocess_input,
    'densenet169': apps.densenet.preprocess_input,
    'densenet201': apps.densenet.preprocess_input,
    'nasnet': apps.nasnet.preprocess_input,
    'mobilenetv2': apps.mobilenet_v2.preprocess_input
}

from keras.layers import Dropout
from keras.regularizers import l2
from keras.models import load_model
from keras import Model
from optimizers import getOptimizer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelBinarizer
from generator import AugmentedBatchGenerator, BatchGenerator
from extsum import extended_summary
import numpy as np
```

```
import datetime as dt
import pickle
import csv
import subprocess
from random import shuffle, seed

# load default specified dataset
data_path = os.path.abspath(args.path)
X_raw = []
y_raw = []

for d in os.listdir(data_path):
    for f in os.listdir(data_path + '/' + d):
        X_raw.append('/'.join([data_path, d, f]))
        y_raw.append(d)

lb = LabelBinarizer()
lb.fit(y_raw)
seed(args.seed)

if args.testset:
    # if a test set was specified, load it
    print(f'Using data at {args.testset} as test set.')

    # shuffle training data
    training_shuffled = list(zip(X_raw, y_raw))
    shuffle(training_shuffled)
    X_data, y_data = zip(*training_shuffled)

    X_data, y_data = np.asarray(X_data), lb.transform(y_data)
    X_train, y_train = X_data, y_data
    data_path = os.path.abspath(args.testset)

    X_raw = []
    y_raw = []

    for d in os.listdir(data_path):
        for f in os.listdir(data_path + '/' + d):
            X_raw.append('/'.join([data_path, d, f]))
            y_raw.append(d)

    # shuffle test
    test_shuffled = list(zip(X_raw, y_raw))
    shuffle(test_shuffled)
    X_raw, y_raw = zip(*test_shuffled)

    X_test, y_test = np.asarray(X_raw), lb.transform(y_raw)
else:
    # otherwise split default dataset
    print(f'Using {args.split} of the provided dataset as test set.')

    X_data, y_data = np.asarray(X_raw), lb.transform(y_raw)
    X_train, X_test, y_train, y_test = train_test_split(
        X_data, y_data, test_size=args.split, random_state=args.seed)

del X_raw
del y_raw

n_classes = y_data.shape[1]

models_dict = {
    'xception': apps.Xception,
    'vgg16': apps.VGG16,
    'vgg19': apps.VGG19,
    'resnet50': apps.ResNet50,
    'inceptionv3': apps.InceptionV3,
    'inceptionresnet': apps.InceptionResNetV2,
    'mobilenet': apps.MobileNet,
    'mobilenetv2': apps.MobileNetV2,
    'densenet121': apps.DenseNet121,
    'densenet169': apps.DenseNet169,
    'densenet201': apps.DenseNet201,
    'nasnet': apps.NASNetLarge
}

# load vanilla model with specified parameters
model = models_dict[args.model](
    classes=n_classes, input_shape=input_shape, weights='imagenet'
        if args.pretrained else None)

if args.dropout is not None:
    print('Adding weight decay')
    # insert dropout layer and regularization
    preds = model.layers[-1]
    dp = Dropout(args.dropout)(model.layers[-2].output)
    preds = preds(dp)
    model = Model(inputs=model.inputs, outputs=preds)

    for layer in model.layers:
        if args.regularize_kernel:
            layer.kernel_regularizer = l2(args.weight_decay)
        if args.regularize_bias:
            layer.bias_regularizer = l2(args.weight_decay)
        if args.regularize_activity:
            layer.activity_regularizer = l2(args.weight_decay)

opt = getOptimizer(args.optimizer)

model.compile(loss=args.loss, optimizer=opt, metrics=['accuracy'])
if args.loadfrom:
    print('Loading', args.loadfrom)
    model = load_model(os.path.abspath(args.loadfrom))

# iteratively rename performance file
pfldir = os.path.abspath(args.pfldir)
performance_file = os.path.join(
```

```
    pfldir, f'{args.model}_b{args.batch}_e{args.epochs}.pfl')
fnum = 1
while os.path.isfile(performance_file):
    performance_file = os.path.join(
        pfldir, f'{args.model}_b{args.batch}_e{args.epochs}_{fnum}.pfl')
    fnum += 1
os.makedirs(pfldir, exist_ok=True)

if args.print_extsum:
    extended_summary(model)
elif args.print_sum:
    model.summary()

perf_log = PerformanceHistory(performance_file)
# print test parameters to screen
print('\n{:<20}{}'.format('Model', args.model))
print('{:<20}{}'.format('Input_shape', input_shape))
print('{:<20}{}'.format('Epochs', args.epochs))
print('{:<20}{}'.format('Batch_size', args.batch))
print('{:<20}{}'.format('Optimizer', type(opt).__name__))
print('{:<20}{}'.format('Optimizer_params',
                        paramDictFormat(opt.get_config())))
print('{:<20}{}'.format('Loss', args.loss))
print('{:<20}{}'.format('Multiprocessing', 'On' if args.use_mp else 'Off'))
print('{:<20}{}'.format('Performance_log', performance_file))
print('{:<20}{}'.format('Test_log', args.log))
print('{:<20}{}'.format('Dataset', args.path))
reg = []
if args.regularize_kernel:
    reg.append('kernel')
if args.regularize_activity:
    reg.append('activity')
if args.regularize_bias:
    reg.append('bias')
print('{:<20}{}\n'.format('Regularization',
                          'None' if not reg else ', '.join(reg)))

opt = getOptimizer(args.optimizer)
model.compile(loss=args.loss, optimizer=opt, metrics=['accuracy'])

# create training batch generator
if args.augment:
    print('Data_augmentation_enabled.')
    train_gen = AugmentedBatchGenerator(X_train, y_train, args.batch,
            shape=input_shape, ops=[input_preprocessing[args.model]], pad=False)
else:
    print('Data_augmentation_disabled.')
    train_gen = BatchGenerator(X_train, y_train, args.batch, shape=input_shape, ops=[
                               input_preprocessing[args.model]], pad=False)
# create testing batch  generator
test_gen = BatchGenerator(X_test, y_test, args.batch, shape=input_shape, ops=[
                          input_preprocessing[args.model]], pad=False)

# train model
train_start = dt.datetime.now()
model.fit_generator(train_gen, epochs=args.epochs, use_multiprocessing=args.use_mp,
                    validation_data=test_gen, callbacks=[perf_log])
train_end = dt.datetime.now()

# evaluate final model on train set
train_score = model.evaluate_generator(train_gen)
print('Train_loss:', train_score[0])
print('Train_accuracy:', train_score[1])

# evaluate final model on test set
test_score = model.evaluate_generator(test_gen)
print('Test_loss:', test_score[0])
print('Test_accuracy:', test_score[1])

# update tests log with current test data
date_format = '{:%Y-%m-%d_%H:%M}'
log_format = '{:<20}{:<20}{:<20}{:<10}{:<10}{:<15}{:<15.5}{:<15.5}' \
    '{:<15.5}{:<15.5}{:<30}{:<30}{:<70}{:<15}{:<15}{:<15}{:<15.5}{:<15.5}\n'
header_format = '{:<20}{:<20}{:<20}{:<10}{:<10}{:<15}{:<15}{:<15}{:<15}' \
    '{:<15}{:<30}{:<30}{:<70}{:<15}{:<15}{:<15}{:<15}{:<15}\n'
with open(args.log, 'a+t') as test_log:
    if test_log.tell() <= 0:
        test_log.write(header_format.format(
            'BEGIN', 'END', 'ARCHITECTURE', 'BATCH', 'EPOCHS',
            'OPTIMIZER', 'TRAIN_LOSS', 'TRAIN_ACC', 'TEST_LOSS',
            'TEST_ACC', 'DATA_FOLDER', 'LOSS_FUNCTION', 'OPTIMIZER_PARAMS',
            'KERNEL_REG', 'BIAS_REG', 'ACTIV._REG', 'DECAY', 'DROPOUT'))
    start_str = date_format.format(train_start)
    end_str = date_format.format(train_end)
    data_folder = args.path.split('/')[-1 if args.path[-1] != '/' else -2]

    test_log.write(log_format.format(start_str, end_str, args.model.upper(),
                                     args.batch, args.epochs, args.optimizer.upper(
    ), train_score[0], train_score[1],
        test_score[0], test_score[1], data_folder, losses_dict[args.loss], paramDictFormat(
        opt.get_config()),
        'YES' if args.regularize_kernel else 'NO',
        'YES' if args.regularize_bias else 'NO',
        'YES' if args.regularize_activity else 'NO',
        args.weight_decay,
        args.dropout if args.dropout else 0.0))

# save the model and class file if an output filename was specified
if args.output is not None:
    print(f'Saving_model_as_{args.output}.h5')
    os.makedirs('/'.join(args.output.split('/')[:-1]), exist_ok=True)
    model.save(f'{args.output}.h5')
```

```
        with open(f'{args.output}.bin', 'wb') as fout:
            pickle.dump((args.model, lb), fout)

    subprocess.run(['notify-send', 'Entrenamiento completado',
                    f'Se ha completado el entrenamiento del modelo {args.model}.'], check=False)

if __name__ == '__main__':
    main()
```

A.0.2. Generador de lotes

```
#!/usr/bin/env python
import math
from keras.utils import Sequence
from keras.preprocessing.image import load_img, img_to_array
import numpy as np
import cv2
import random
import imaug

def clamp(val, minimum, maximum):
    'Clamp value between a minimum and maximum'
    if val < minimum:
        return minimum
    if val > maximum:
        return maximum
    return val

def loadImage(path: str, target_size=None, grayscale=False):
    color_flag = cv2.IMREAD_COLOR if not grayscale else cv2.IMREAD_GRAYSCALE
    img = cv2.imread(path, color_flag)
    if target_size is not None:
        img = cv2.resize(img, target_size)
    return img

def getPadded(img: np.array, output_size: int):
    'Returns image padded so that height and width are the same'
    if img is None:
        return None
    if len(img.shape) == 3:
        h, w, _ = img.shape
        pixel_val = [0, 0, 0]
    elif len(img.shape) == 2:
        h, w = img.shape
        pixel_val = [0]
    else:
        raise ValueError('Unexpected image shape {}'.format(img.shape))
    top, bottom, left, right = 0, 0, 0, 0
    if h > w:
        left = math.floor((h-w)/2)
        right = math.ceil((h-w)/2)
    elif h < w:
        top = math.floor((w-h)/2)
        bottom = math.ceil((w-h)/2)
    padded = cv2.copyMakeBorder(img, top, bottom, left, right,
                                cv2.BORDER_CONSTANT, value=pixel_val) if h != w else img
    return cv2.resize(padded, (output_size, output_size))

def augment(img: np.array, vflip: bool, hflip: bool, rotation: int, translation_matrix: np.array):
    cols, rows = img.shape[:2]
    img = imaug.flip(img, vflip, hflip)
    img = imaug.rotate(img, rotation)
    if translation_matrix is not None:
        img = cv2.warpAffine(img, translation_matrix, (cols, rows))
    return img

class BatchGenerator(Sequence):
    'Batch generator class'

    def __init__(self, x_set, y_set, batch_size: int, shape: tuple, pad: bool = False,
                 ops: list = [lambda a: a], grayscale: bool = False):
        self.x, self.y = x_set, y_set
        self.batch_size = batch_size
        self.shape = tuple(shape)
        self.pad = pad
        self.ops = ops
        self.grayscale = grayscale

    def __len__(self):
        return int(np.ceil(len(self.x) / float(self.batch_size)))

    def __getitem__(self, idx):
        batch_x = self.x[idx * self.batch_size:(idx + 1) * self.batch_size]
        batch_y = self.y[idx * self.batch_size:(idx + 1) * self.batch_size]

        image_list = [cv2.resize(cv2.imread(f, cv2.IMREAD_COLOR), (224, 224))
                      for f in batch_x if f is not None]
        image_tensor = np.array(image_list).astype(np.float32)

        for op in self.ops:
            image_tensor = op(image_tensor)
        return image_tensor.astype(np.float32), np.array(batch_y)

class AugmentedBatchGenerator(Sequence):
    'Batch generator class with data augmentation'

    def __init__(self, x_set, y_set, batch_size: int, shape: tuple, pad: bool = False,
                 ops: list = [lambda a: a], vflip: bool = True, hflip: bool = True,
                 rotation_range: int = 360, grayscale=False, flip_probability: float = 0.5,
```

```
            x_translation_range: int = 1, y_translation_range: int = 1):
        self.x, self.y = x_set, y_set
        self.batch_size = batch_size
        self.shape = tuple(shape)
        self.pad = pad
        self.ops = ops
        self.grayscale = grayscale
        self.flip_prob = clamp(flip_probability, 0.0, 1.1)
        self.rotation_range = rotation_range
        self.vflip = vflip
        self.hflip = hflip
        self.x_translation_range = x_translation_range
        self.y_translation_range = y_translation_range
        self.i = 0

    def __len__(self):
        return int(np.ceil(len(self.x) / float(self.batch_size)))

    def __getitem__(self, idx):
        batch_x = self.x[idx * self.batch_size:(idx + 1) * self.batch_size]
        batch_y = self.y[idx * self.batch_size:(idx + 1) * self.batch_size]

        if self.pad:
            image_list = [getPadded(loadImage(f, grayscale=self.grayscale), max(
                self.shape[:2])) for f in batch_x if f is not None]
        else:
            image_list = [loadImage(
                f, grayscale=self.grayscale, target_size=self.shape[:2])
                          for f in batch_x if f is not None]
        auglist = []
        for img in image_list:
            # augment
            vflip = (random.uniform(0, 1) >
                     self.flip_prob) if self.vflip else False
            hflip = (random.uniform(0, 1) >
                     self.flip_prob) if self.hflip else False
            rotation_angle = random.randrange(0, self.rotation_range)

            y_trans = random.randint(1, self.y_translation_range)
            x_trans = random.randint(1, self.x_translation_range)

            M = np.float32([[1, 0, y_trans], [0, 1, x_trans]])
            a = augment(img, vflip, hflip, rotation_angle, M)
            auglist.append(a)

        image_tensor = np.array(auglist).astype(np.float32)
        for op in self.ops:
            image_tensor = op(image_tensor)
        return image_tensor.astype(np.float32), np.array(batch_y)
```

A.0.3. Optimizadores

```
import configparser
default_opt_cfg = 'optimizers.cfg'

def writeDefaultConfig():
    'Write_default_values_to_optimizer_configuration_file'
    cfg = configparser.ConfigParser()
    cfg['sgd'] = {'lr': '0.01', 'momentum': '0.0',
                  'decay': '0.0', 'nesterov': 'False'}
    cfg['rmsprop'] = {'lr': '0.001', 'rho': '0.9',
                      'epsilon': 'None', 'decay': '0.0'}
    cfg['adagrad'] = {'lr': '0.01', 'epsilon': 'None', 'decay': '0.0'}
    cfg['adadelta'] = {'lr': '1.0', 'rho': '0.95',
                       'epsilon': 'None', 'decay': '0.0'}
    cfg['adam'] = {'lr': '0.001', 'beta_1': '0.9', 'beta_2': '0.999',
                   'epsilon': 'None', 'decay': '0.0', 'amsgrad': 'False'}
    cfg['adamax'] = {'lr': '0.002', 'beta_1': '0.9',
                     'beta_2': '0.999', 'epsilon': 'None', 'decay': '0'}
    cfg['nadam'] = {'lr': '0.002', 'beta_1': '0.9',
                    'beta_2': '0.999', 'epsilon': 'None', 'schedule_decay': '0.004'}
    with open(default_opt_cfg, 'w') as f:
        cfg.write(f)

def parseParam(target_optimizer: str, param: str, opt_cfg: configparser.ConfigParser):
    'Parse_and_return_parameter_value'
    a = opt_cfg[target_optimizer][param].upper()
    if a == 'NONE':
        return None
    if a == 'FALSE':
        return False
    if a == 'TRUE':
        return True
    return float(a)

def getOptimizer(opt: str):
    'Returns_a_Keras_optimizer_object'
    import os
    from keras import optimizers as k_opt

    if not os.path.isfile(default_opt_cfg):
        writeDefaultConfig()
    opt_cfg = configparser.ConfigParser()
    opt_cfg.read(default_opt_cfg)
    if opt == 'sgd':
        return k_opt.SGD(lr=parseParam(opt, 'lr', opt_cfg), momentum=parseParam(opt, 'momentum', opt_cfg),
            decay=parseParam(opt, 'decay', opt_cfg), nesterov=parseParam(opt, 'nesterov', opt_cfg))
    if opt == 'rmsprop':
        return k_opt.RMSprop(lr=parseParam(opt, 'lr', opt_cfg), rho=parseParam(opt, 'rho', opt_cfg),
```

```
            epsilon=parseParam(opt, 'epsilon', opt_cfg), decay=parseParam(opt, 'decay', opt_cfg))
    if opt == 'adagrad':
        return k_opt.Adagrad(lr=parseParam(opt, 'lr', opt_cfg), epsilon=parseParam(opt, 'epsilon',
            opt_cfg), decay=parseParam(opt, 'decay', opt_cfg))
    if opt == 'adadelta':
        return k_opt.Adadelta(lr=parseParam(opt, 'lr', opt_cfg), rho=parseParam(opt, 'rho', opt_cfg),
            epsilon=parseParam(opt, 'epsilon', opt_cfg), decay=parseParam(opt, 'decay', opt_cfg))
    if opt == 'adam':
        return k_opt.Adam(lr=parseParam(opt, 'lr', opt_cfg), beta_1=parseParam(opt, 'beta_1', opt_cfg),
            beta_2=parseParam(opt, 'beta_2', opt_cfg), epsilon=parseParam(opt, 'epsilon', opt_cfg),
            decay=parseParam(opt, 'decay', opt_cfg), amsgrad=parseParam(opt, 'amsgrad', opt_cfg))
    if opt == 'adamax':
        return k_opt.Adamax(lr=parseParam(opt, 'lr', opt_cfg), beta_1=parseParam(opt, 'beta_1', opt_cfg),
            beta_2=parseParam(opt, 'beta_2', opt_cfg), epsilon=parseParam(opt, 'epsilon', opt_cfg),
            decay=parseParam(opt, 'decay', opt_cfg))
    if opt == 'nadam':
        return k_opt.Nadam(lr=parseParam(opt, 'lr', opt_cfg), beta_1=parseParam(opt, 'beta_1', opt_cfg),
            beta_2=parseParam(opt, 'beta_2', opt_cfg), epsilon=parseParam(opt, 'epsilon', opt_cfg),
            schedule_decay=parseParam(opt, 'schedule_decay', opt_cfg))
    raise ValueError('Unrecognized optimizer ({})'.format(opt))
```

A.0.4. Sumarizador de modelos

```
from keras.models import Model

def extended_summary(model: Model):
    cfg = model.get_config()
    fields = ('#', 'Name', 'Type', 'Activation', 'Trainable', 'Filters',
              'Kernel_size', 'Strides', 'Padding', 'Pool_Size', 'Units')
    print('%-10s %-30s %-30s %-20s %-15s %-15s %-15s %-15s %-15s %-15s %-15s' % fields)
    i = 0
    for l in cfg['layers']:
        values = (i, l['name'], l['class_name'],
        l['config']['activation'] if 'activation' in l['config'] else '-',
        l['config']['trainable'] if 'trainable' in l['config'] else '-',
        l['config']['filters'] if 'filters' in l['config'] else '-',
        l['config']['kernel_size'] if 'kernel_size' in l['config'] else '-',
        l['config']['strides'] if 'strides' in l['config'] else '-',
        l['config']['padding'] if 'padding' in l['config'] else '-',
        l['config']['pool_size'] if 'pool_size' in l['config'] else '-',
        l['config']['units'] if 'units' in l['config'] else '-'
        )
        print('%-10i %-30s %-30s %-20s %-15s %-15s %-15s %-15s %-15s %-15s %-15s' % values)
        i += 1
```

Anexo B
Entrenamiento de modelos

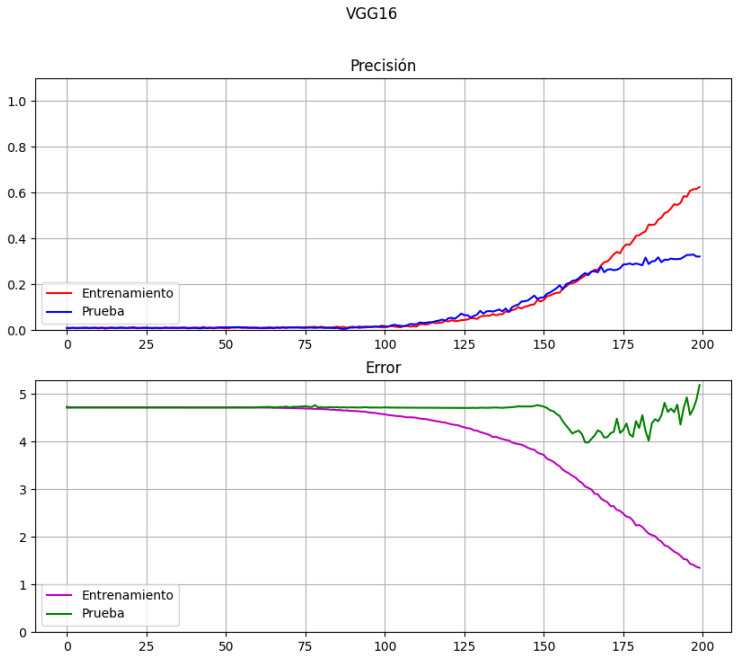

Figura B.1: Desempeño de VGG16 durante el entrenamiento general, primera ronda.

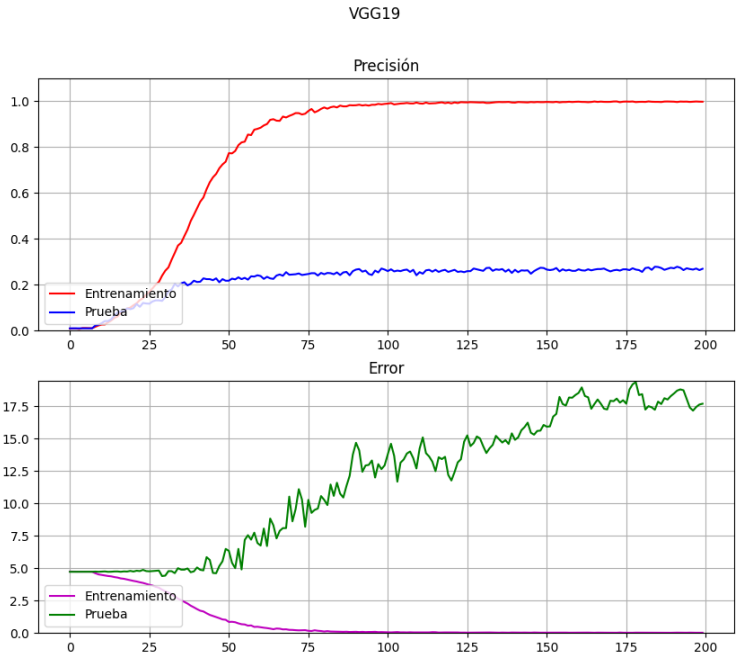

Figura B.2: Desempeño de VGG19 durante el entrenamiento general, primera ronda.

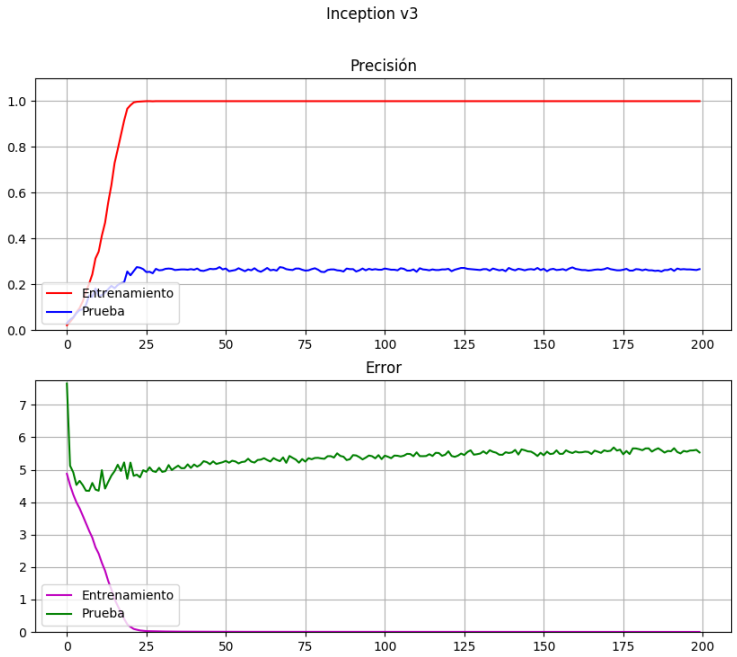

Figura B.3: Desempeño de Inception v3 durante el entrenamiento general, primera ronda.

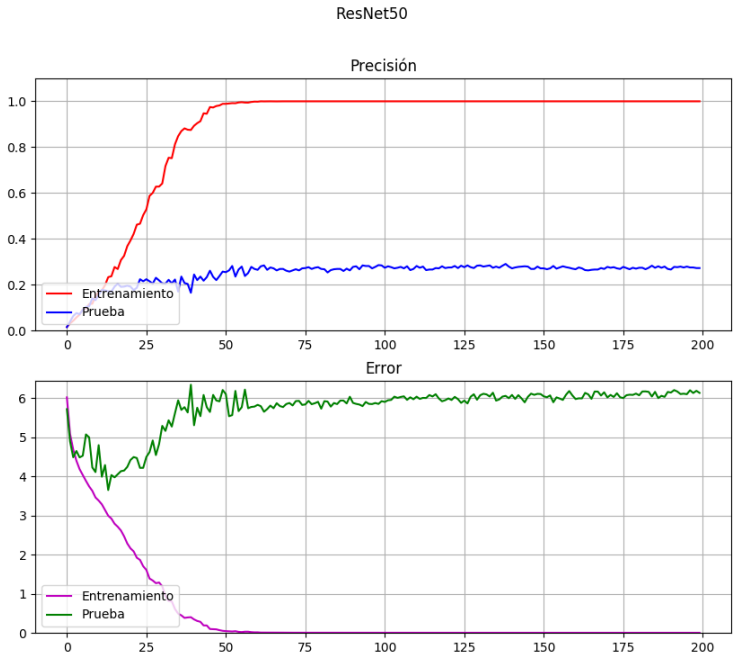

Figura B.4: Desempeño de ResNet50 durante el entrenamiento general, primera ronda.

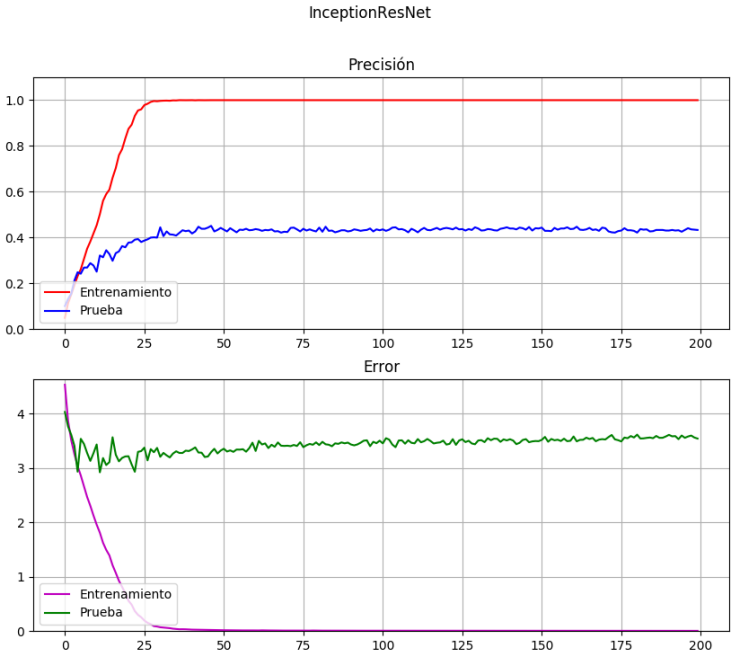

Figura B.5: Desempeño de InceptionResNet durante el entrenamiento general, primera ronda.

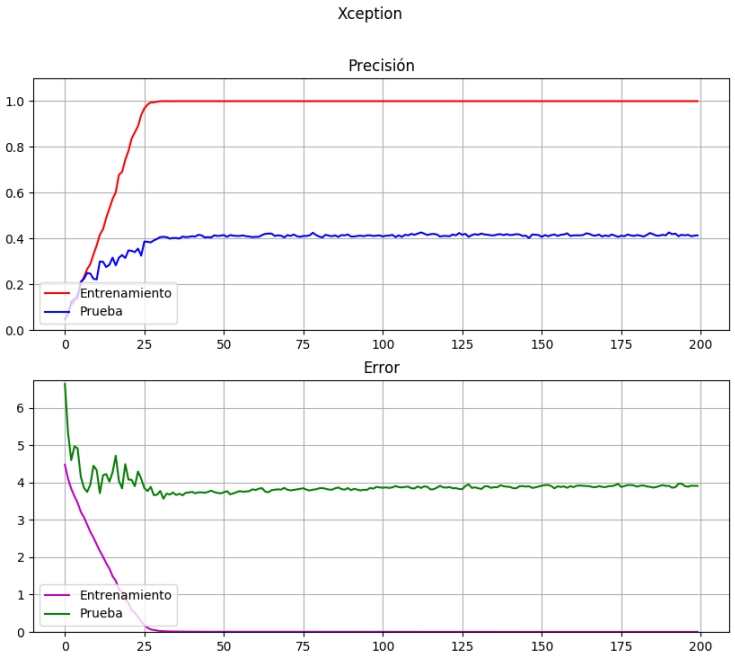

Figura B.6: Desempeño de Xception durante el entrenamiento general, primera ronda.

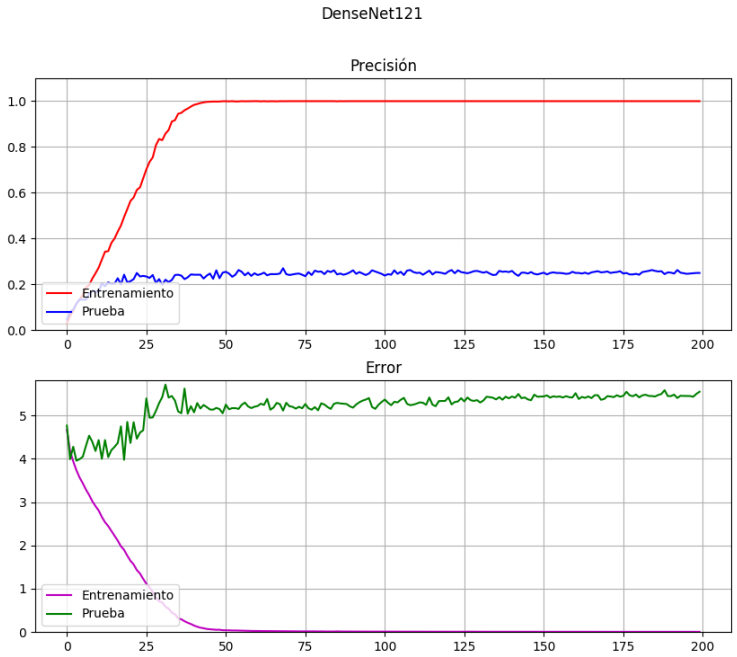

Figura B.7: Desempeño de DenseNet121 durante el entrenamiento general, primera ronda.

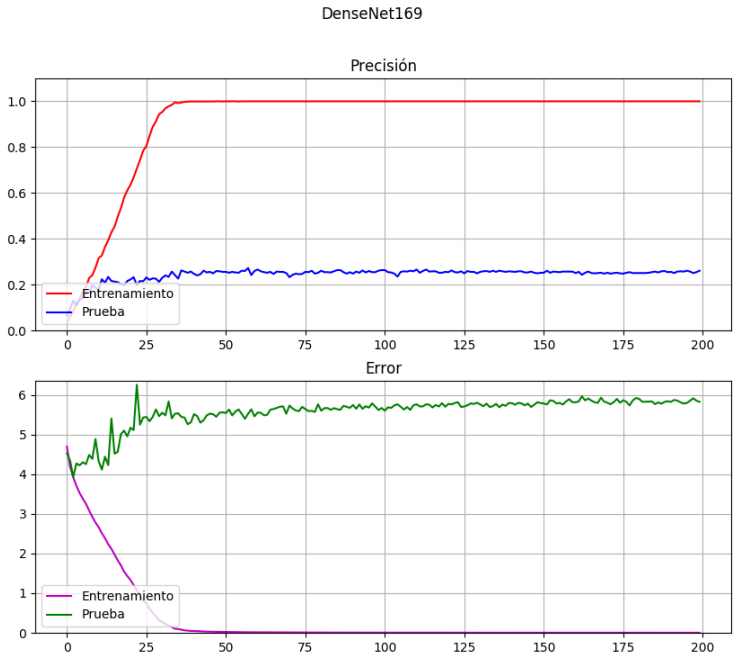

Figura B.8: Desempeño de DenseNet169 durante el entrenamiento general, primera ronda.

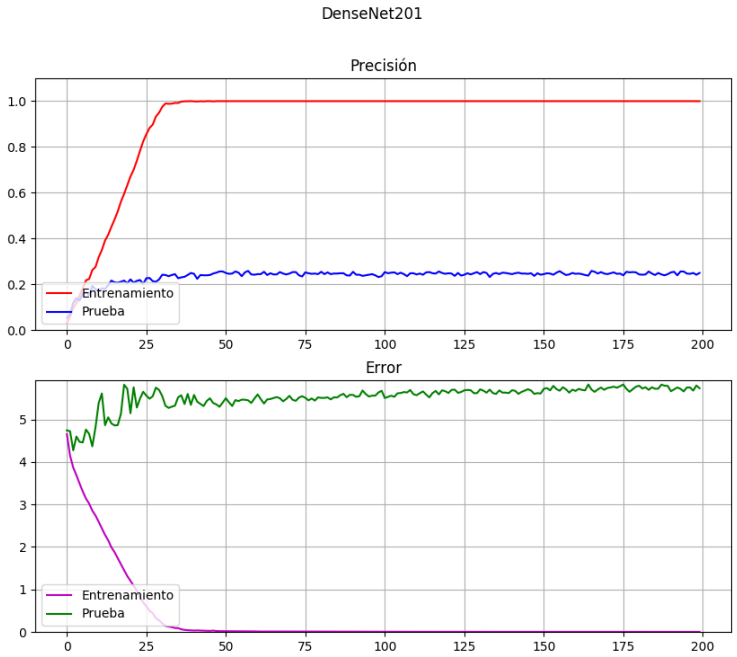

Figura B.9: Desempeño de DenseNet201 durante el entrenamiento general, primera ronda.

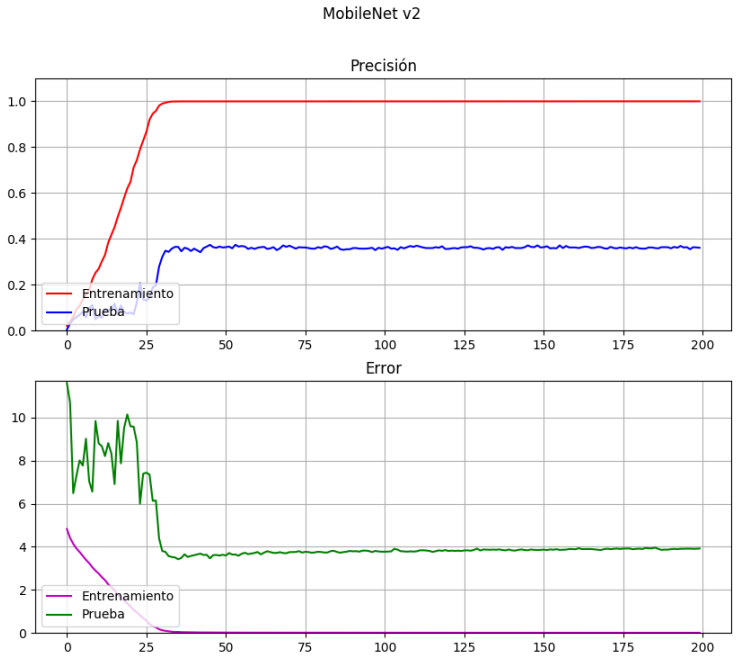

Figura B.10: Desempeño de MobileNetv2 durante el entrenamiento general, primera ronda.

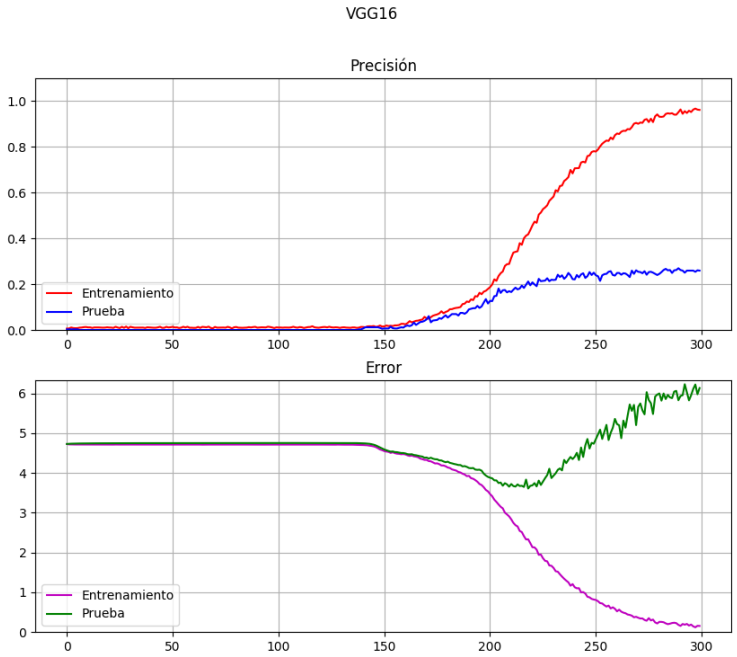

Figura B.11: Desempeño de VGG16 durante el entrenamiento general, segunda ronda.

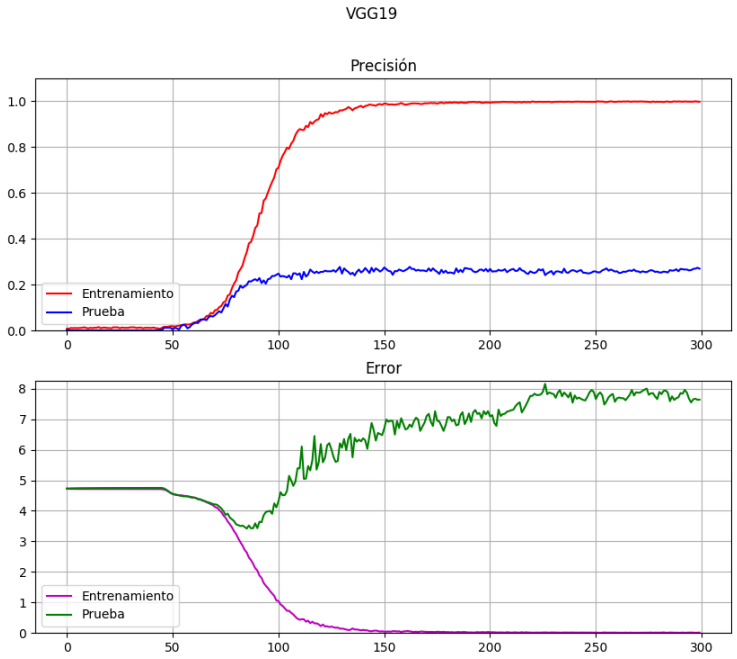

Figura B.12: Desempeño de VGG19 durante el entrenamiento general, segunda ronda.

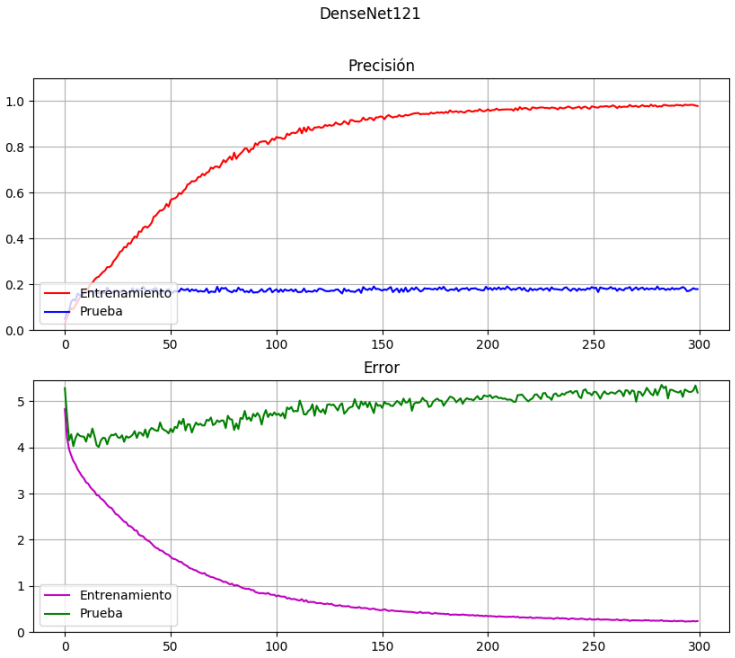

Figura B.13: Desempeño de DenseNet121 durante el entrenamiento general, segunda ronda.

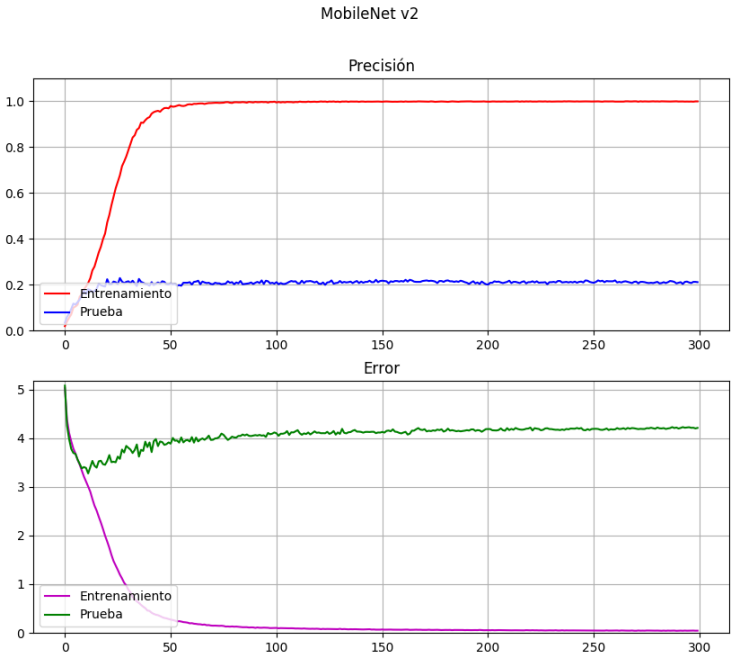

Figura B.14: Desempeño de MobileNet v2 durante el entrenamiento general, segunda ronda.

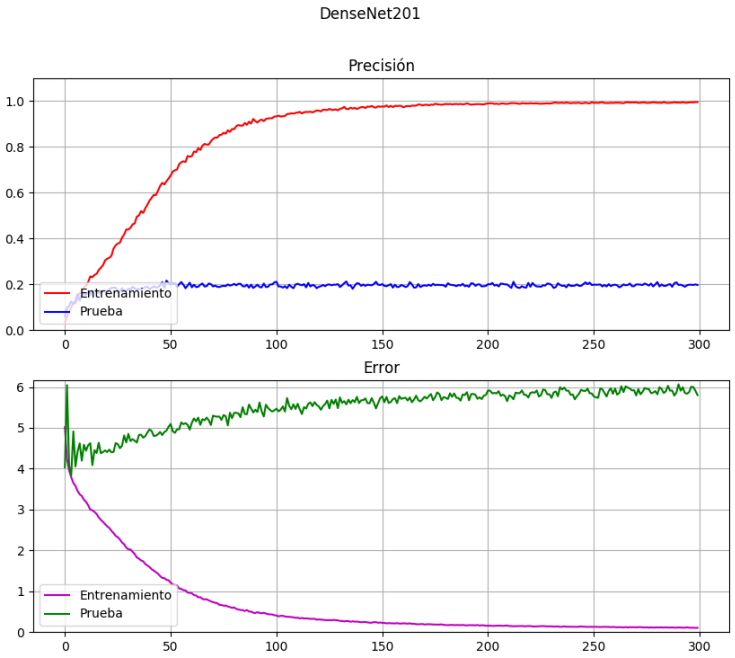

Figura B.15: Desempeño de DenseNet201 durante el entrenamiento general, segunda ronda.

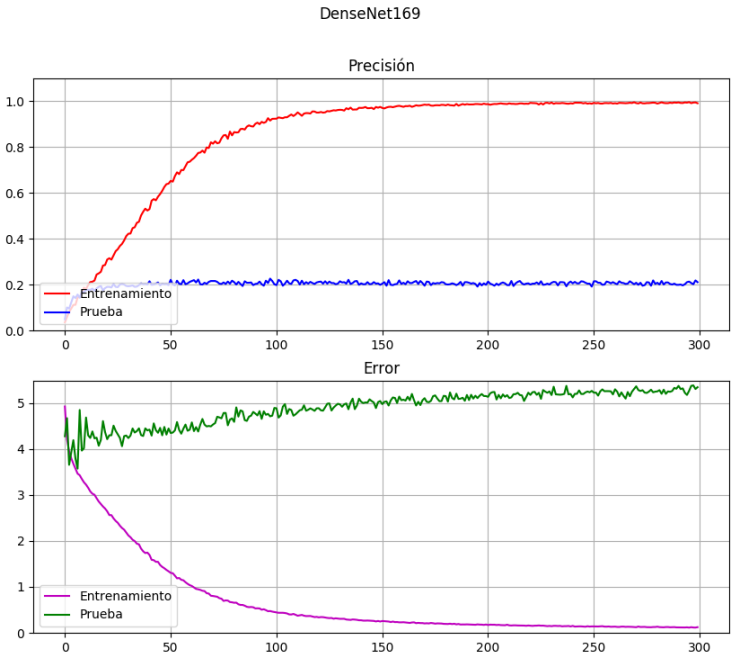

Figura B.16: Desempeño de DenseNet169 durante el entrenamiento general, segunda ronda.

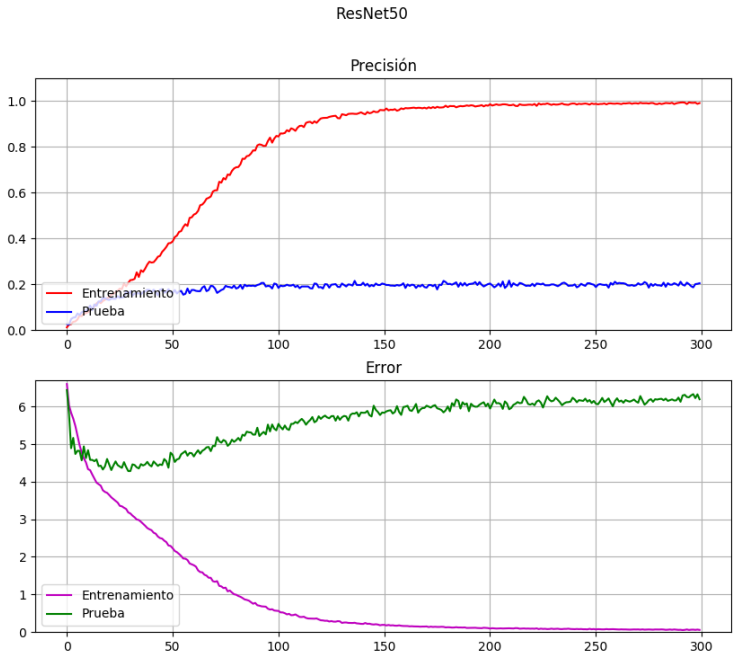

Figura B.17: Desempeño de ResNet50 durante el entrenamiento general, segunda ronda.

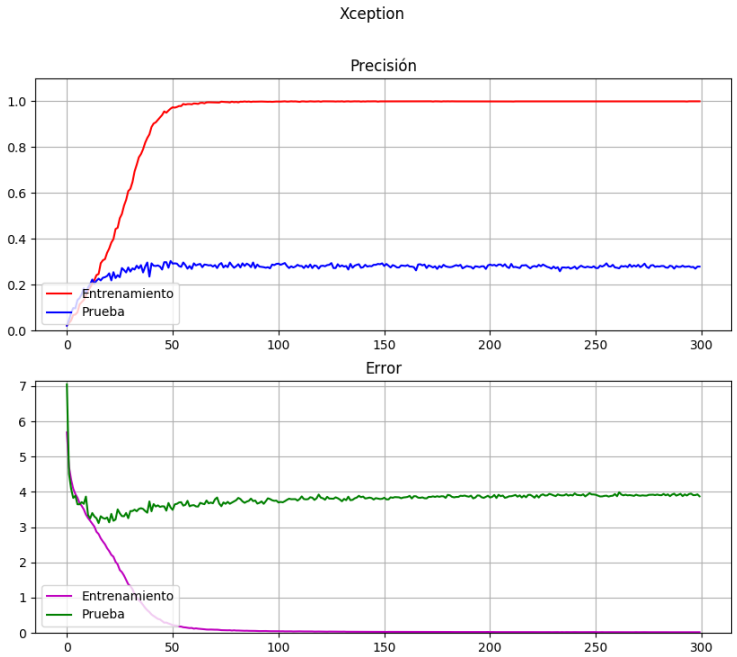

Figura B.18: Desempeño de Xception durante el entrenamiento general, segunda ronda.

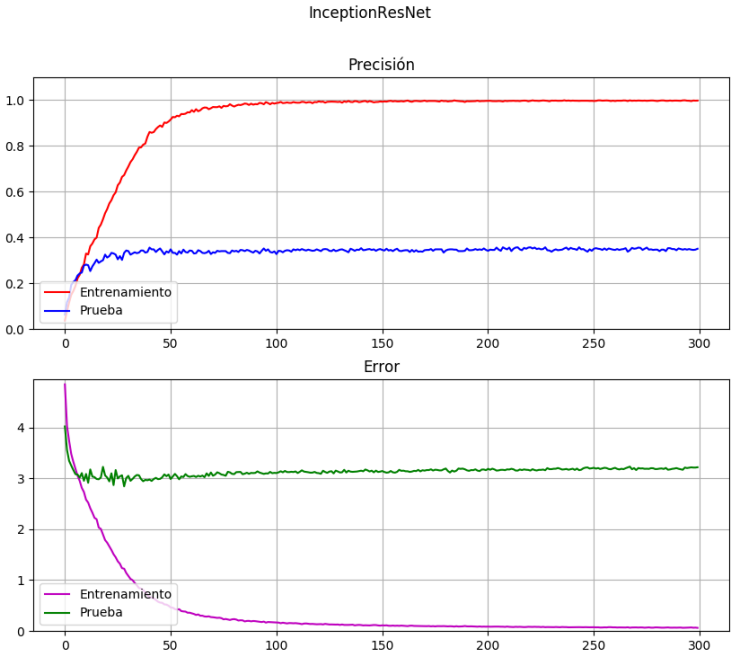

Figura B.19: Desempeño de InceptionResNet durante el entrenamiento general, segunda ronda.

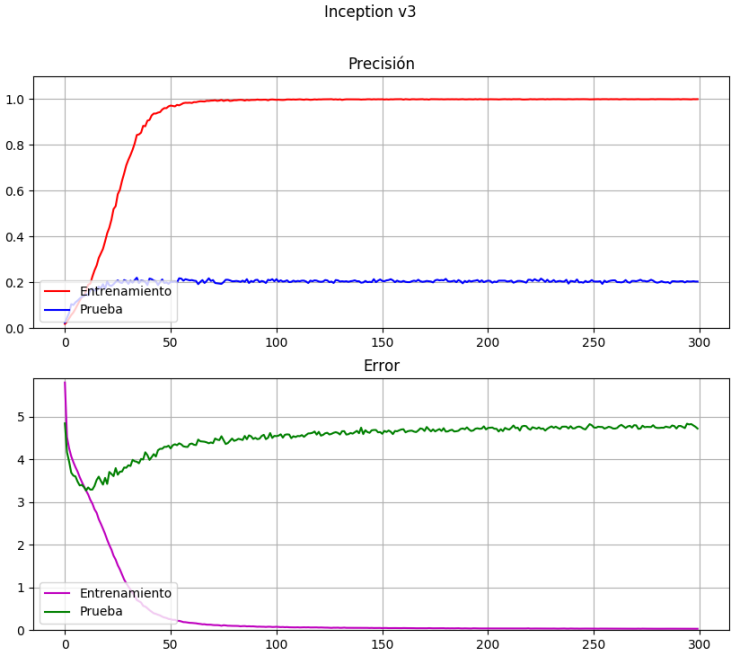

Figura B.20: Desempeño de Inception v3 durante el entrenamiento general, segunda ronda.

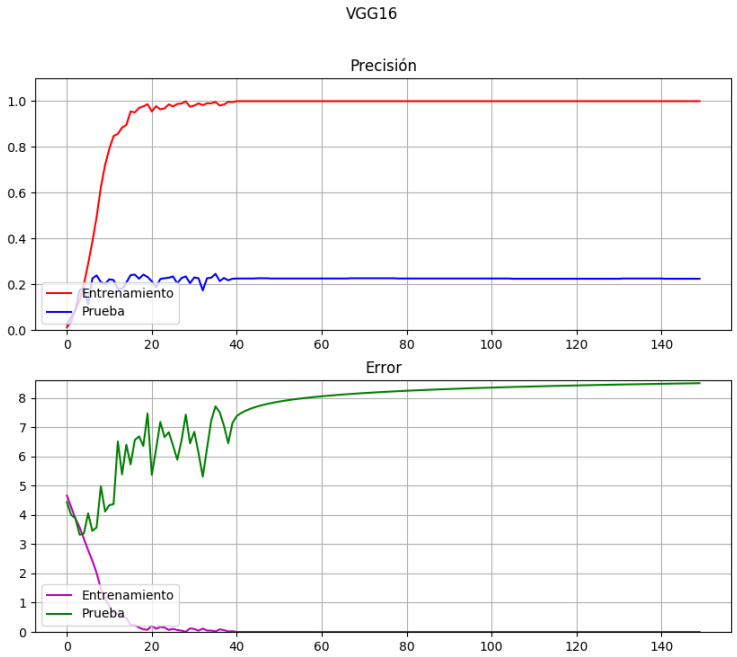

Figura B.21: Desempeño de VGG16 durante el entrenamiento general, tercera ronda.

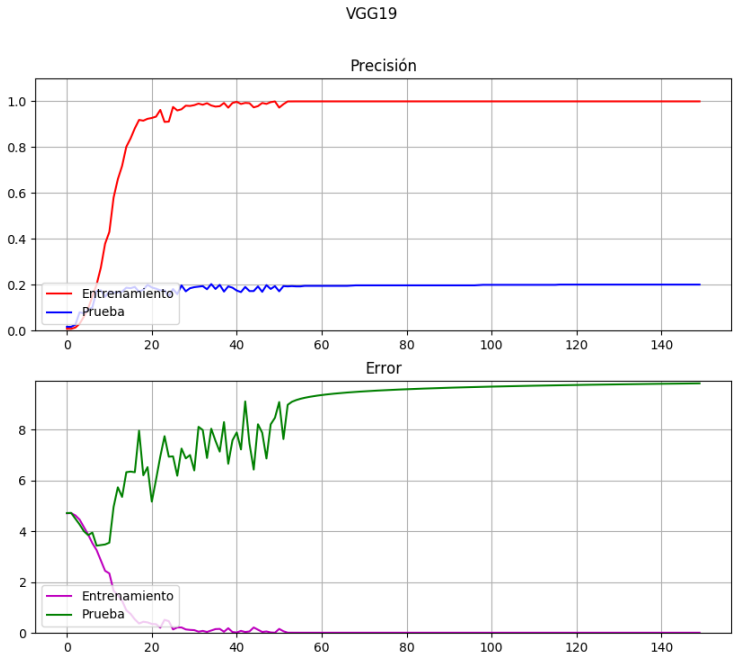

Figura B.22: Desempeño de VGG19 durante el entrenamiento general, tercera ronda.

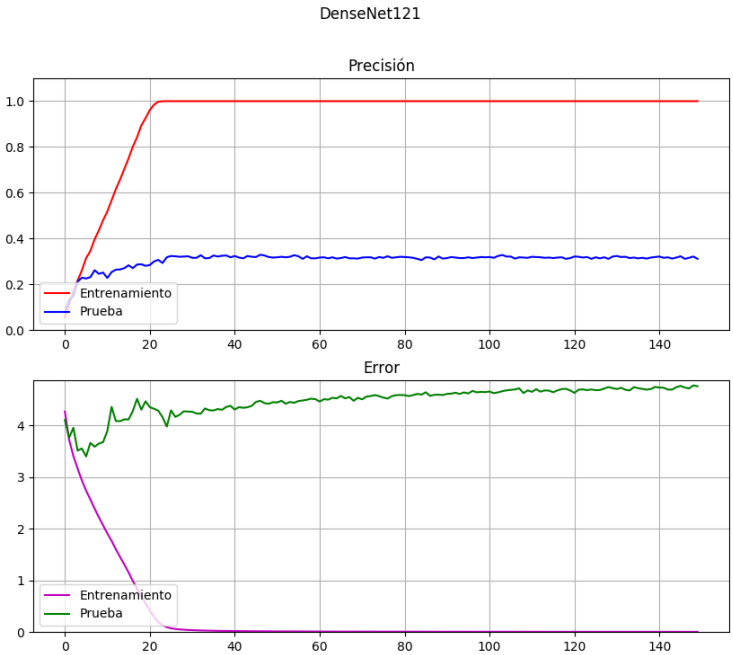

Figura B.23: Desempeño de DenseNet121 durante el entrenamiento general, tercera ronda.

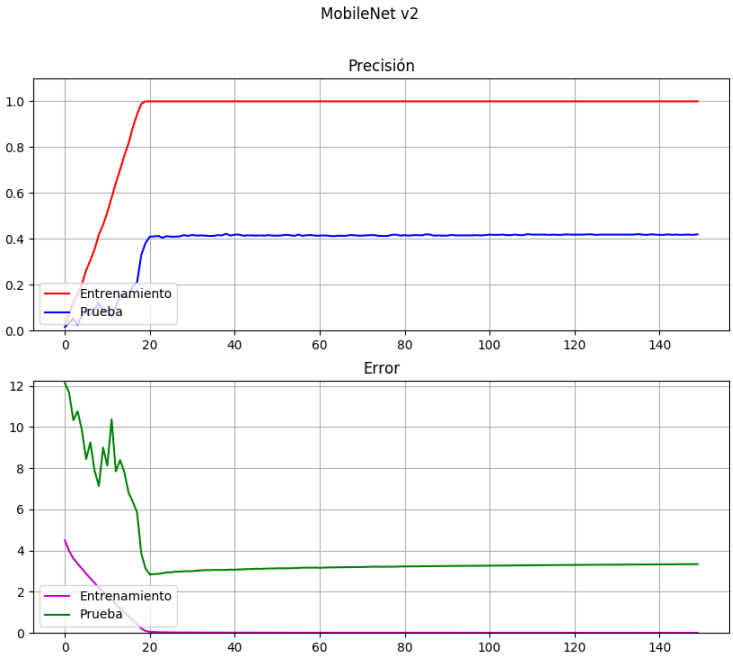

Figura B.24: Desempeño de MobileNet v2 durante el entrenamiento general, tercera ronda.

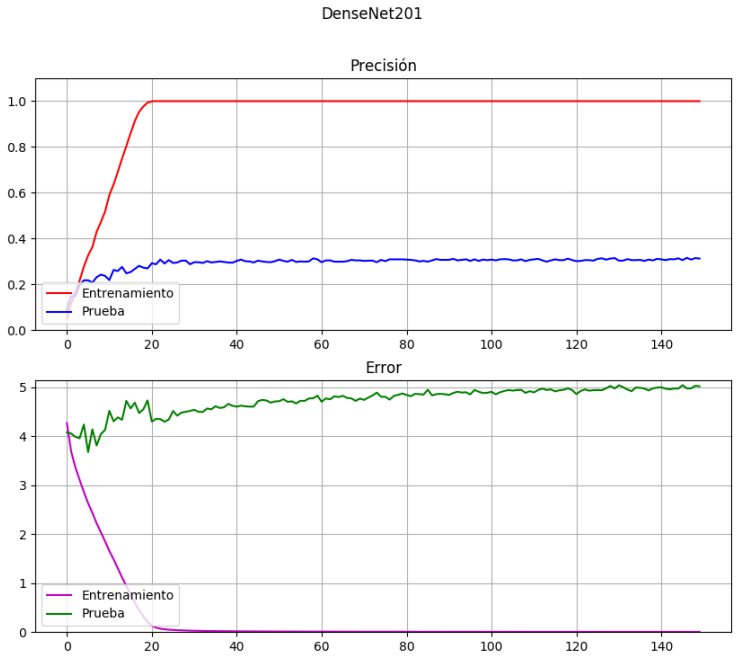

Figura B.25: Desempeño de DenseNet201 durante el entrenamiento general, tercera ronda.

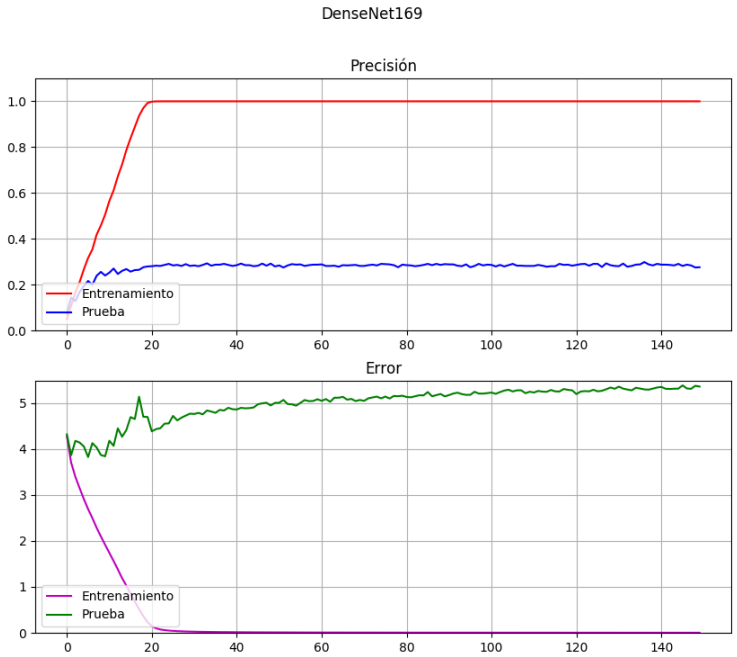

Figura B.26: Desempeño de DenseNet169 durante el entrenamiento general, tercera ronda.

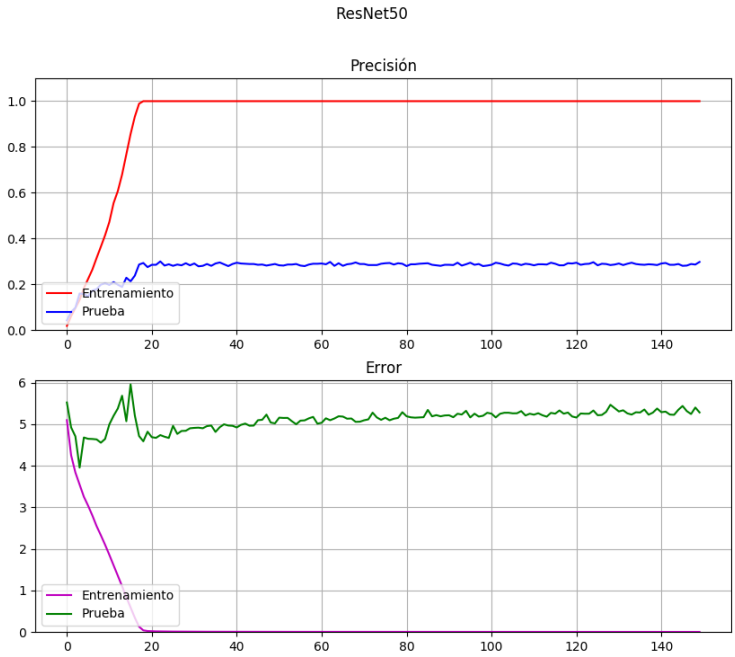

Figura B.27: Desempeño de ResNet50 durante el entrenamiento general, tercera ronda.

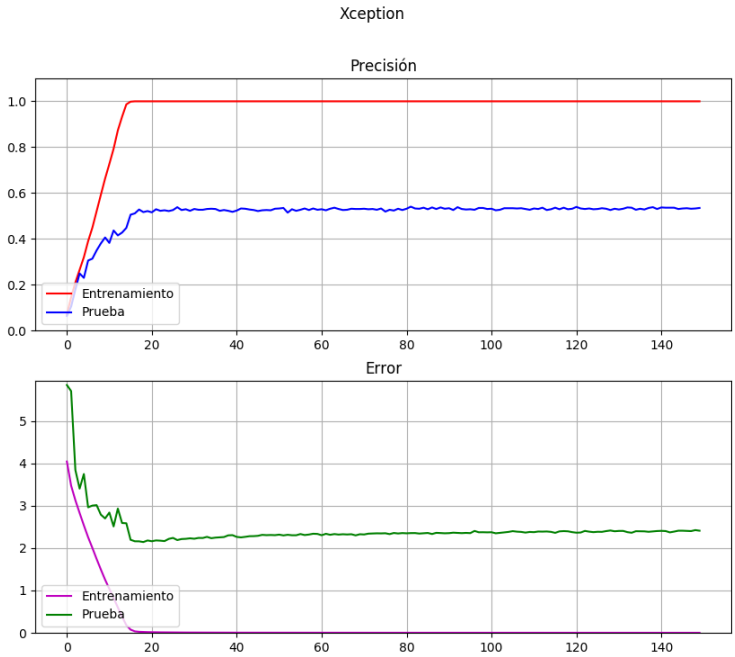

Figura B.28: Desempeño de Xception durante el entrenamiento general, tercera ronda.

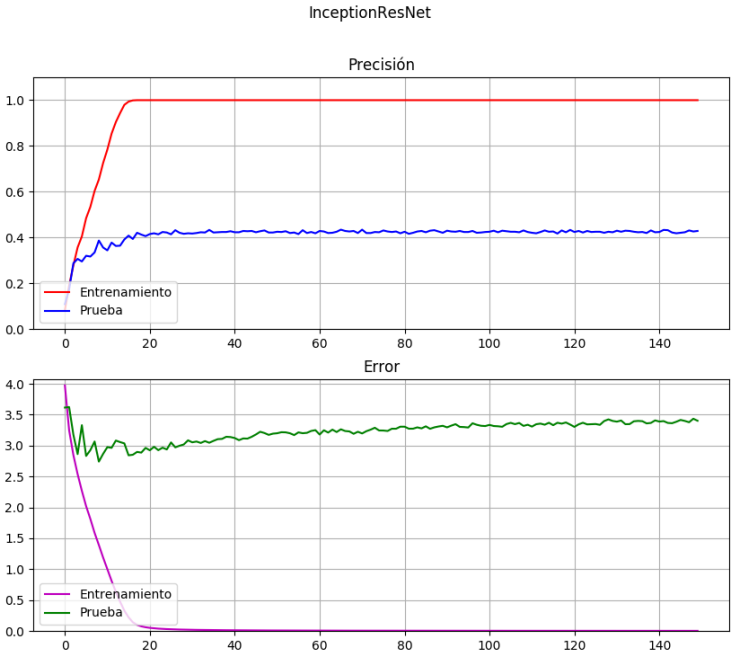

Figura B.29: Desempeño de InceptionResNet durante el entrenamiento general, tercera ronda.

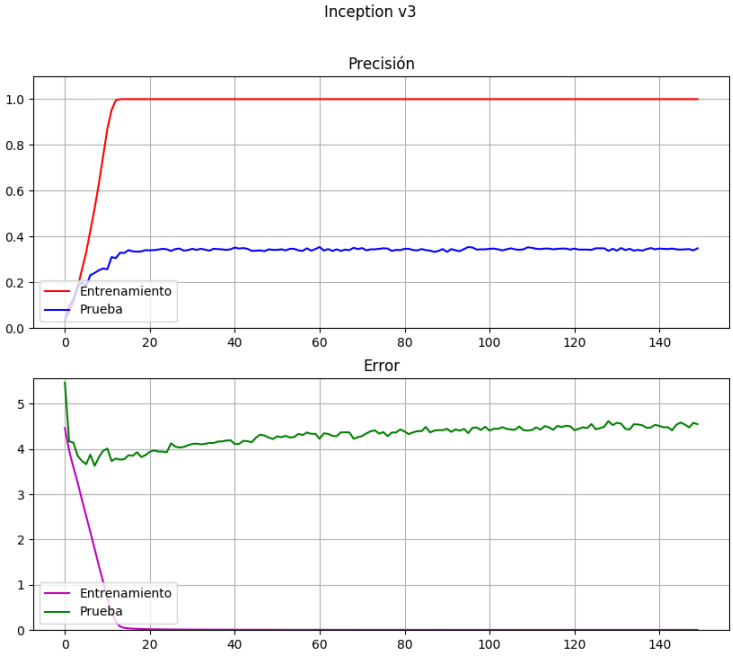

Figura B.30: Desempeño de Inception v3 durante el entrenamiento general, qtercera ronda.

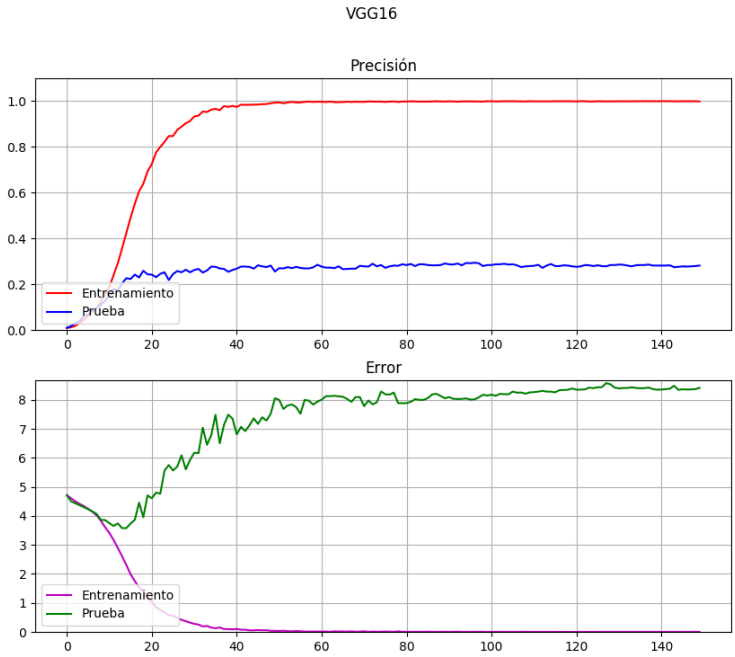

Figura B.31: Desempeño de VGG16 durante el entrenamiento general, cuarta ronda.

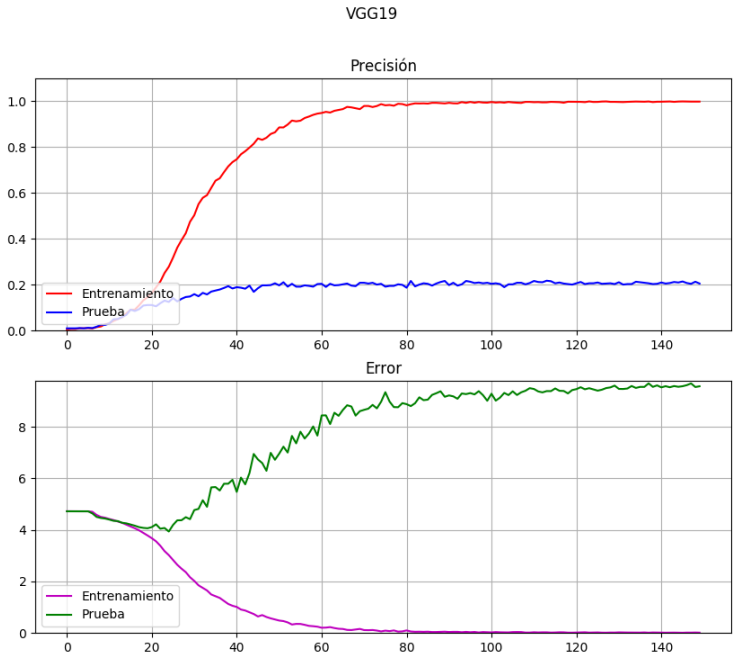

Figura B.32: Desempeño de VGG19 durante el entrenamiento general, cuarta ronda.

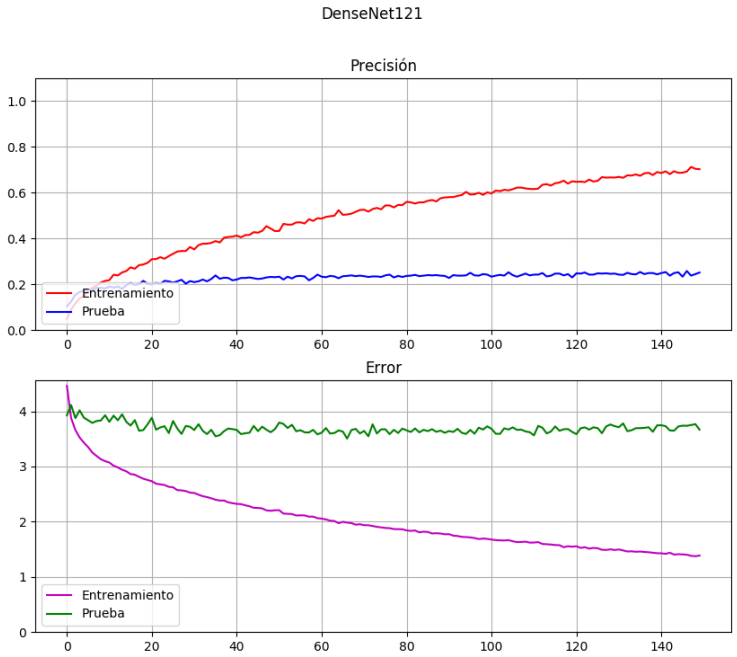

Figura B.33: Desempeño de DenseNet121 durante el entrenamiento general, cuarta ronda.

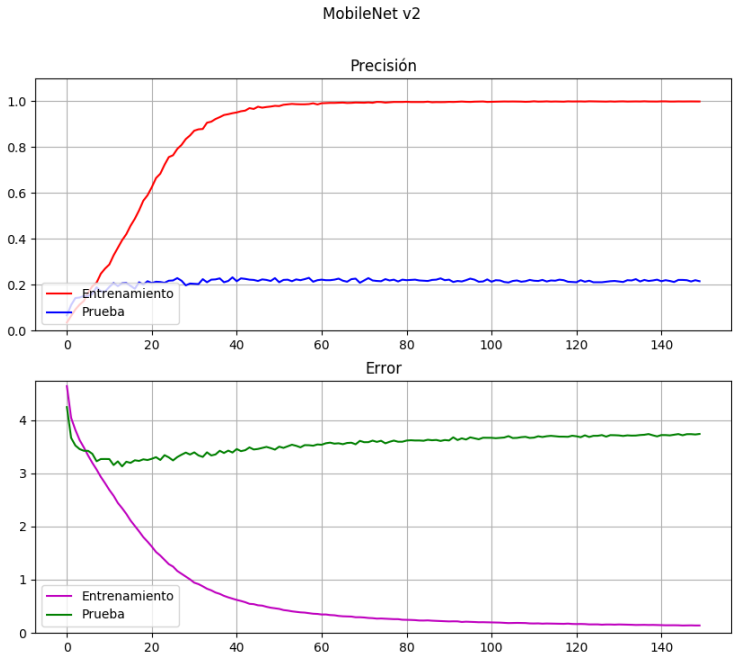

Figura B.34: Desempeño de MobileNet v2 durante el entrenamiento general, cuarta ronda.

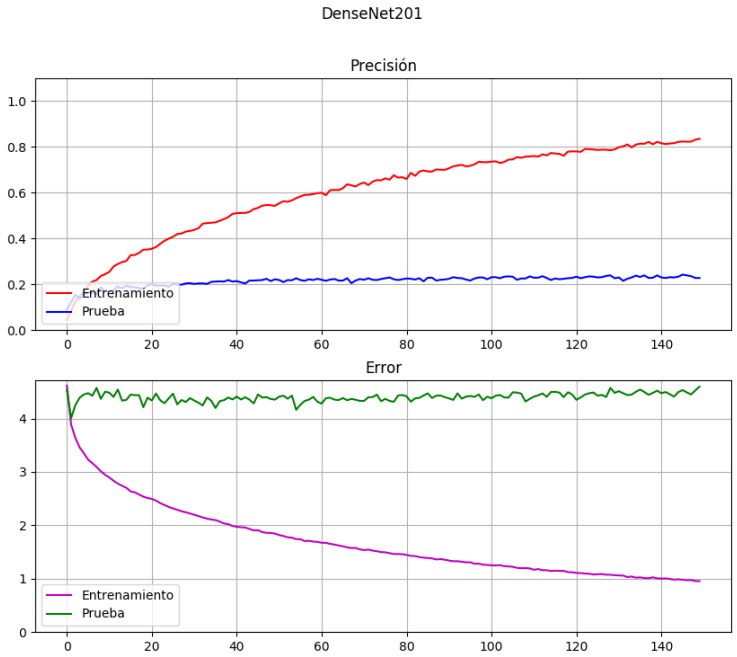

Figura B.35: Desempeño de DenseNet201 durante el entrenamiento general, cuarta ronda.

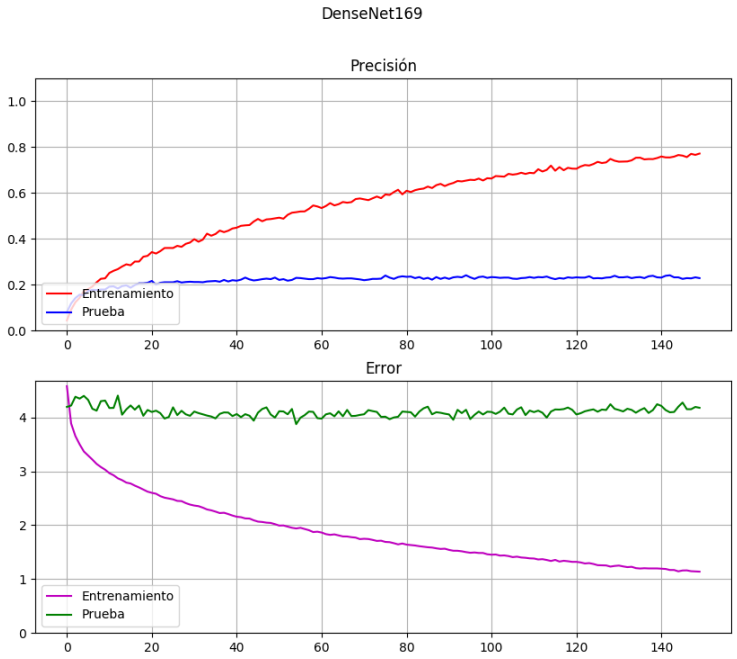

Figura B.36: Desempeño de DenseNet169 durante el entrenamiento general, cuarta ronda.

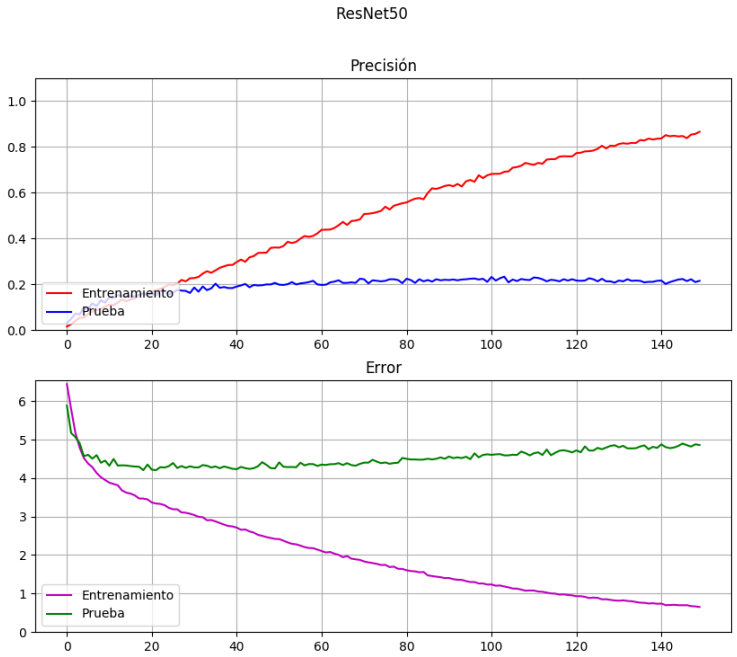

Figura B.37: Desempeño de ResNet50 durante el entrenamiento general, cuarta ronda.

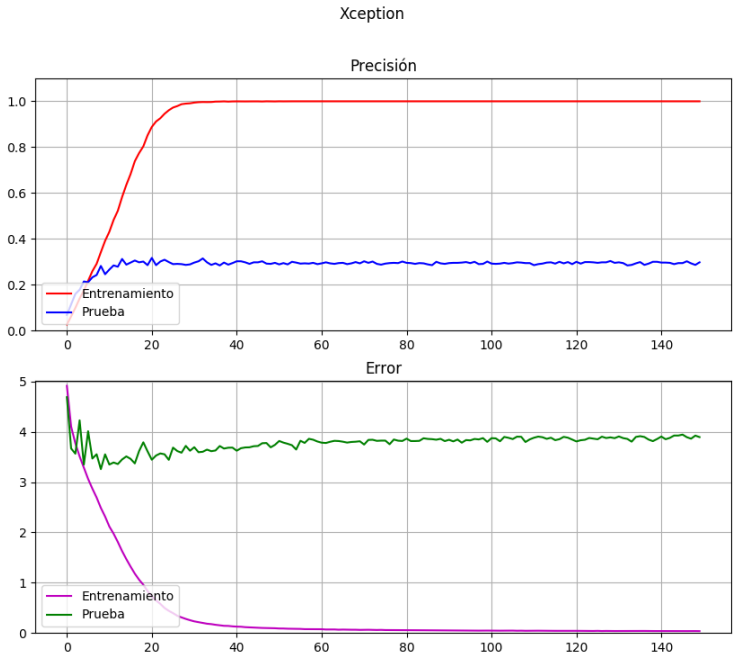

Figura B.38: Desempeño de Xception durante el entrenamiento general, cuarta ronda.

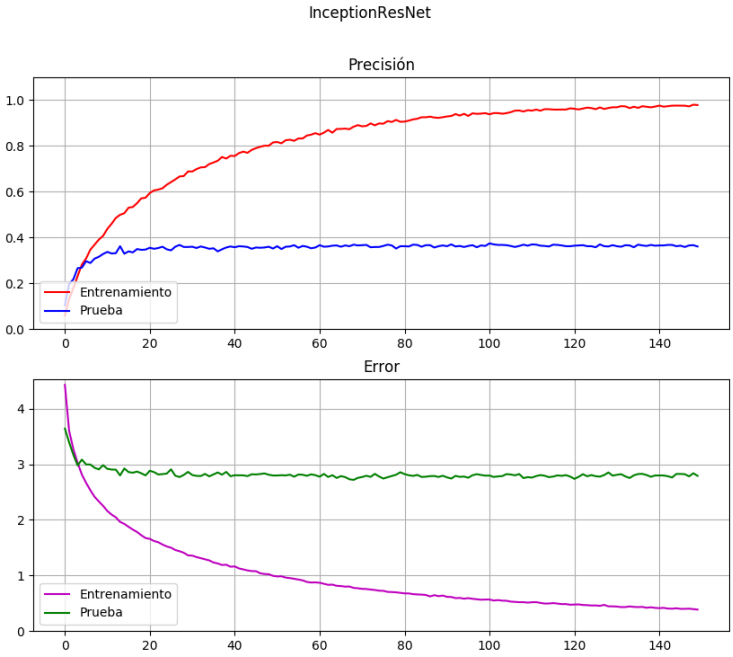

Figura B.39: Desempeño de InceptionResNet durante el entrenamiento general, cuarta ronda.

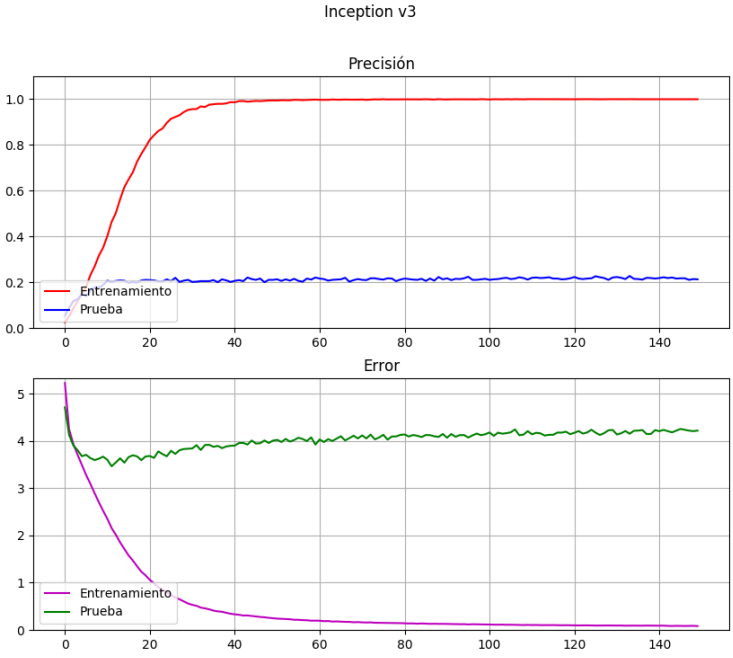

Figura B.40: Desempeño de Inception v3 durante el entrenamiento general, cuarta ronda.

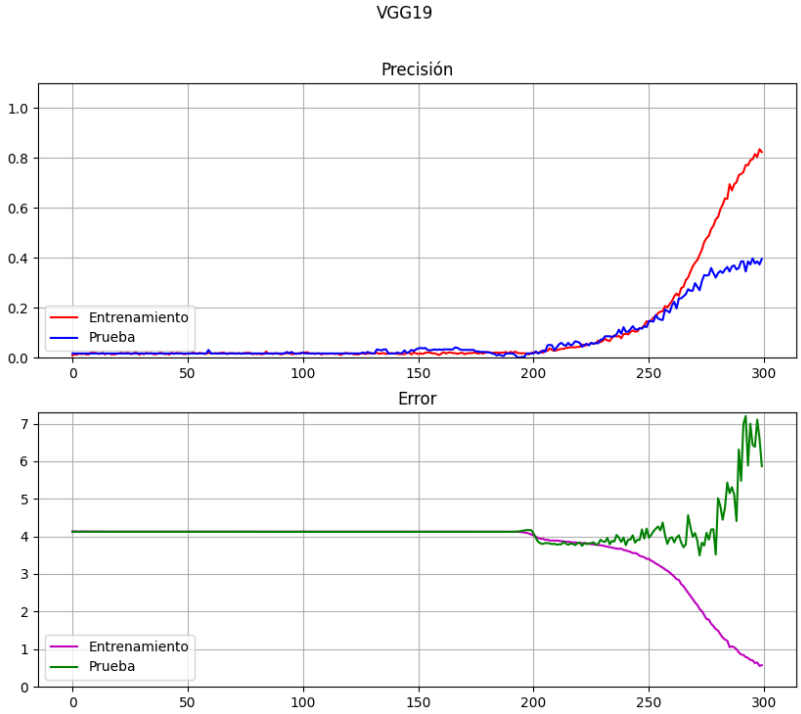

Figura B.41: Desempeño de VGG19 durante el entrenamiento por órgano, primera ronda, flores.

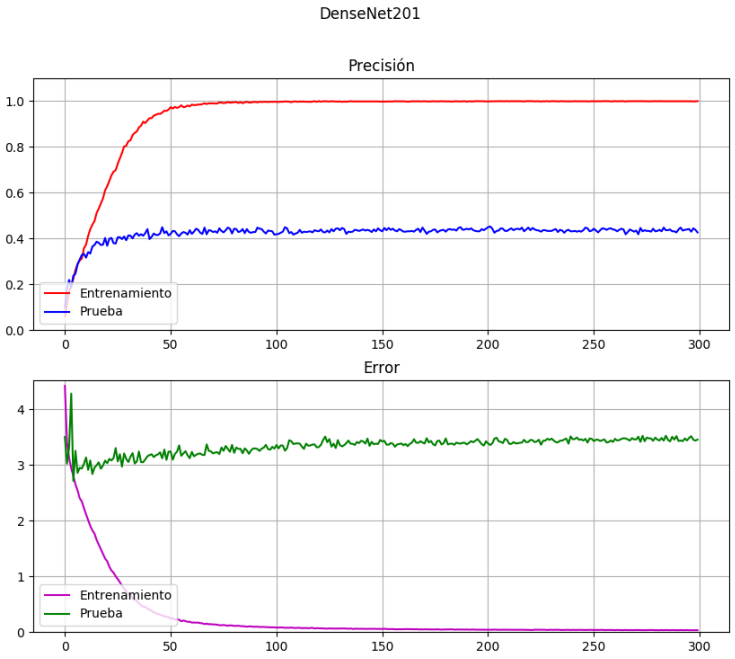

Figura B.42: Desempeño de DenseNet201 durante el entrenamiento por órgano, primera ronda, flores.

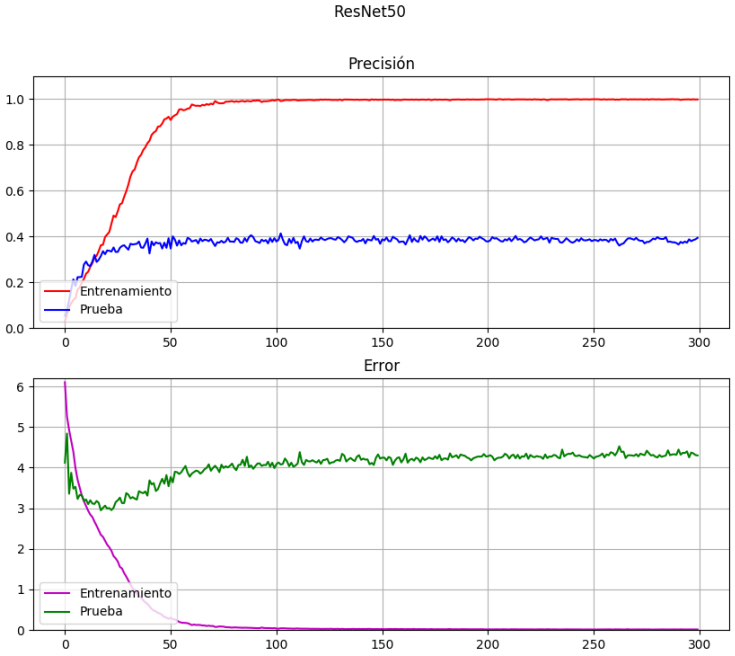

Figura B.43: Desempeño de ResNet50 durante el entrenamiento por órgano, primera ronda, flores.

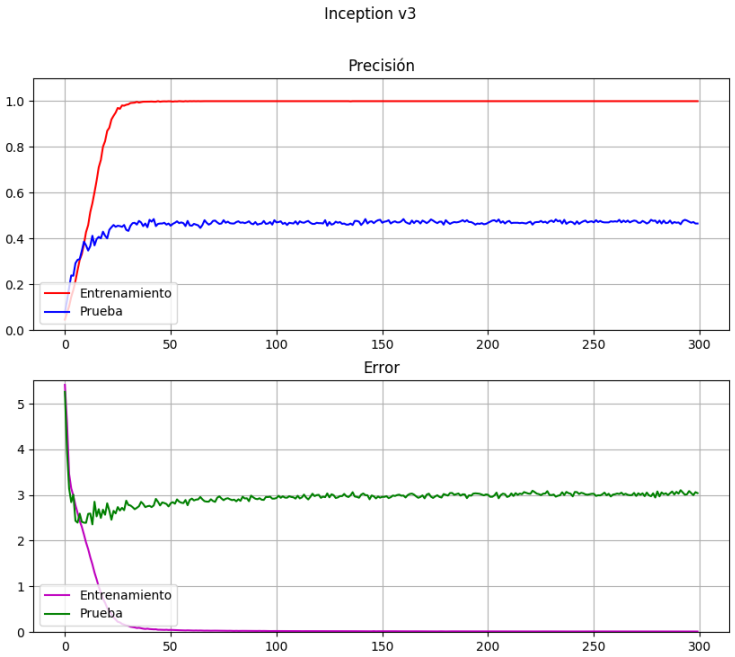

Figura B.44: Desempeño de Inception v3 durante el entrenamiento por órgano, primera ronda, flores.

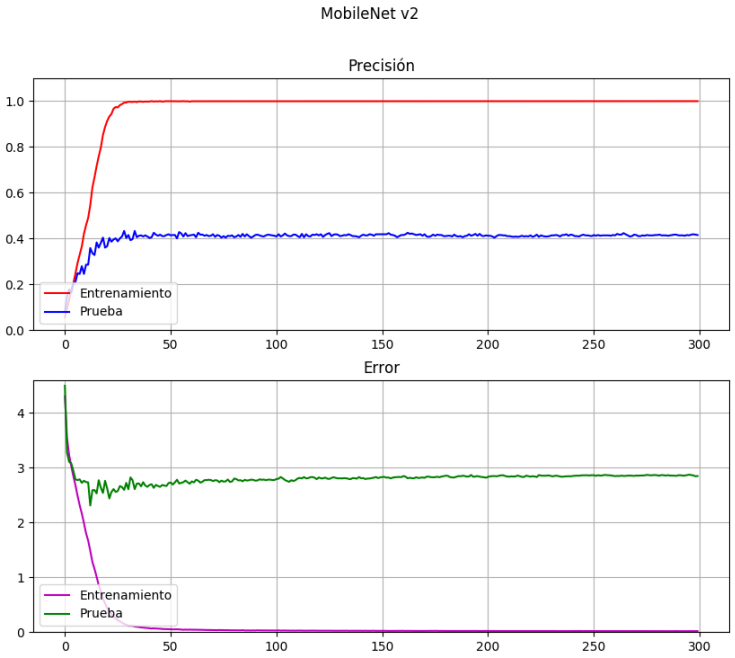

Figura B.45: Desempeño de MobileNet v2 durante el entrenamiento por órgano, primera ronda, flores.

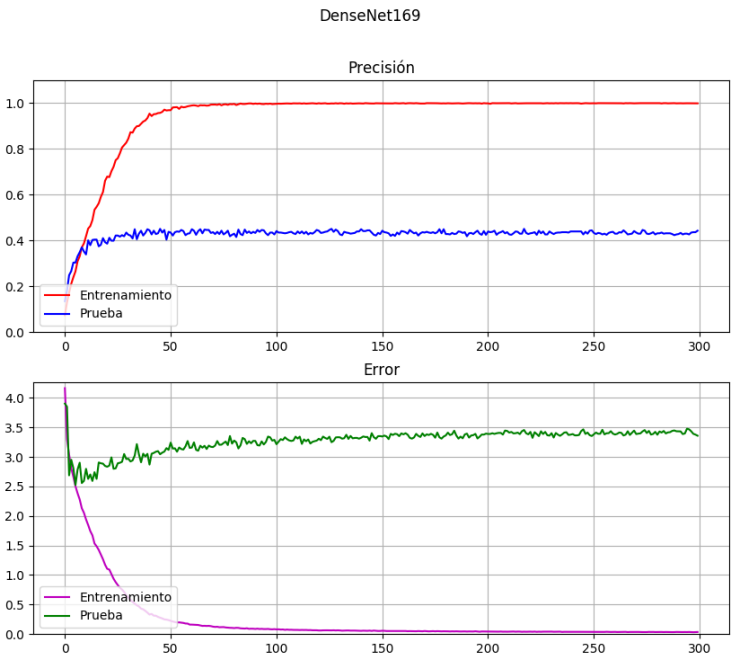

Figura B.46: Desempeño de DenseNet169 durante el entrenamiento por órgano, primera ronda, flores.

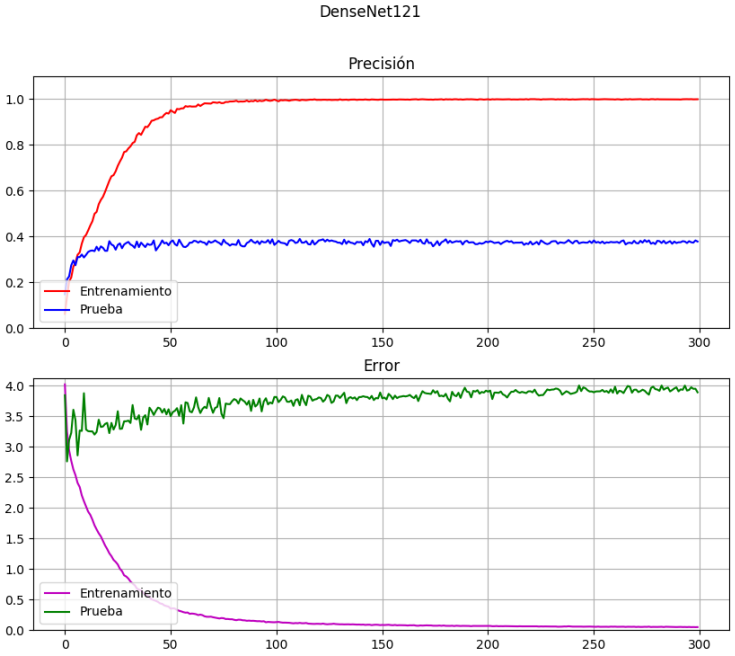

Figura B.47: Desempeño de DenseNet121 durante el entrenamiento por órgano, primera ronda, flores.

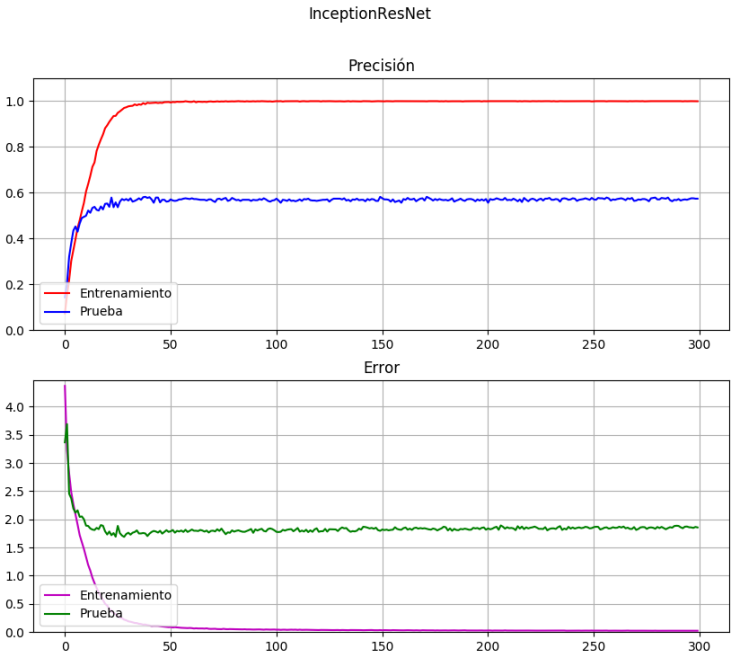

Figura B.48: Desempeño de InceptionResNet durante el entrenamiento por órgano, primera ronda, flores.

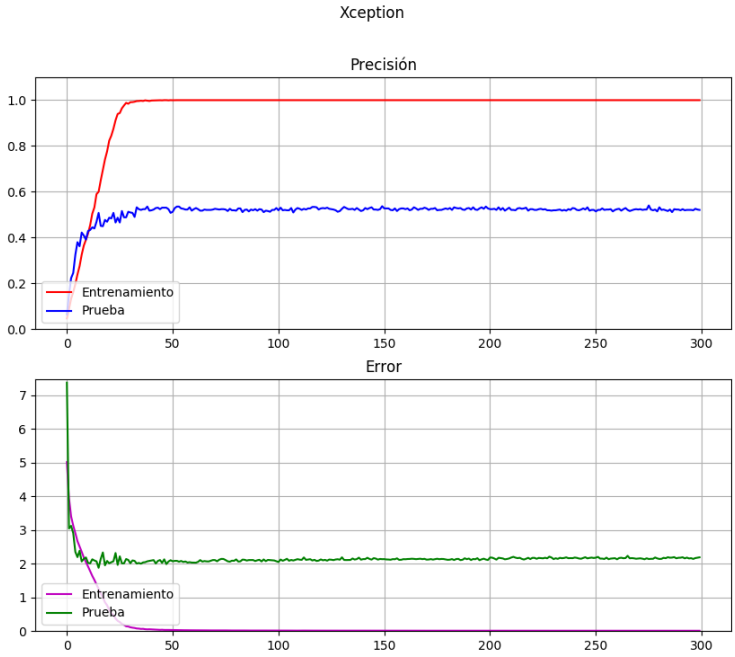

Figura B.49: Desempeño de Xception durante el entrenamiento por órgano, primera ronda, flores.

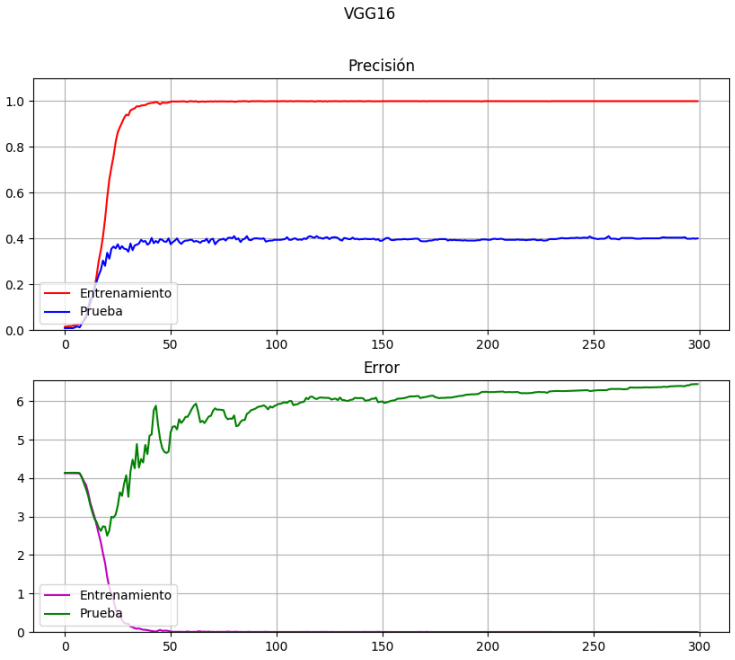

Figura B.50: Desempeño de VGG16 durante el entrenamiento por órgano, primera ronda, flores.

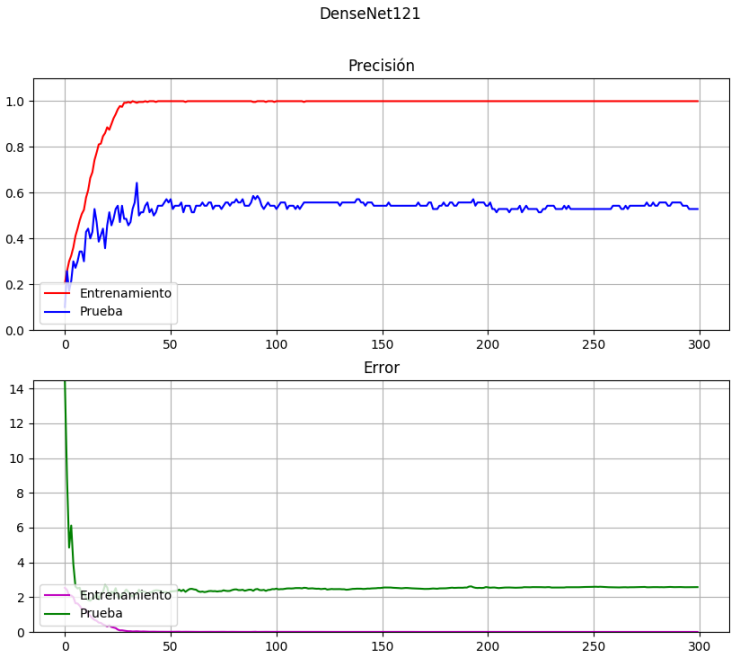

Figura B.51: Desempeño de DenseNet121 durante el entrenamiento por órgano, primera ronda, frutos.

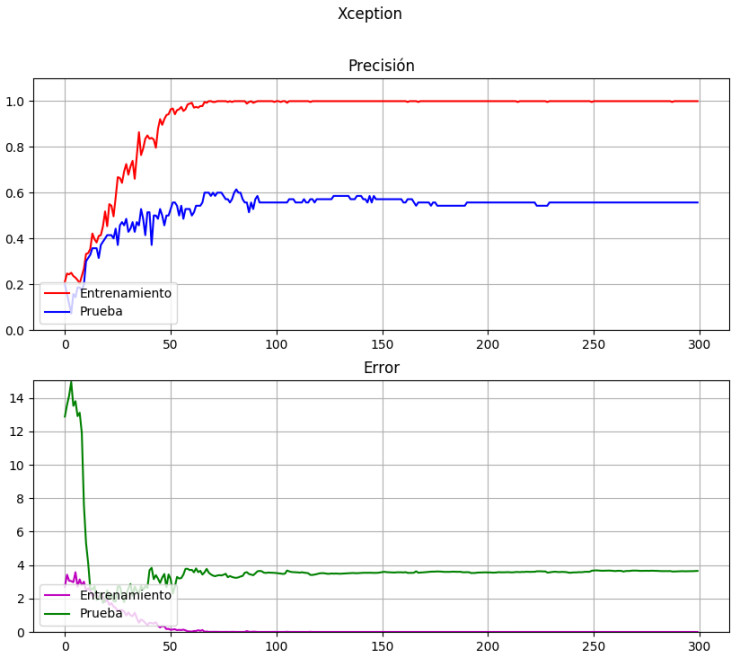

Figura B.52: Desempeño de Xception durante el entrenamiento por órgano, primera ronda, frutos.

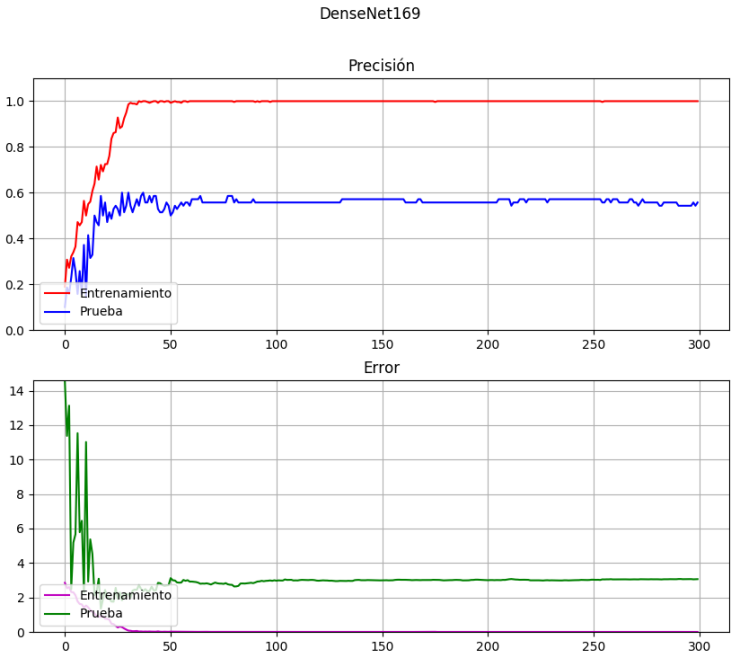

Figura B.53: Desempeño de DenseNet169 durante el entrenamiento por órgano, primera ronda, frutos.

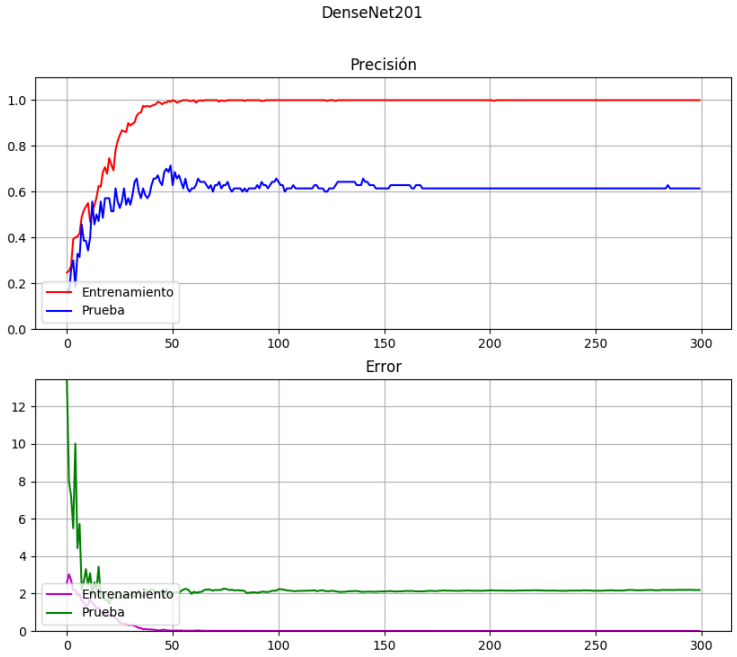

Figura B.54: Desempeño de DenseNet201 durante el entrenamiento por órgano, primera ronda, frutos.

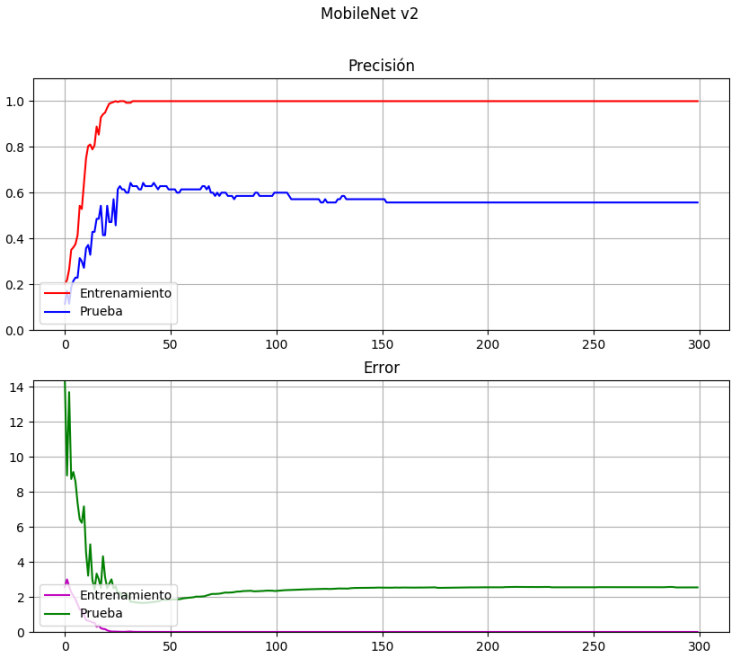

Figura B.55: Desempeño de MobileNet v2 durante el entrenamiento por órgano, primera ronda, frutos.

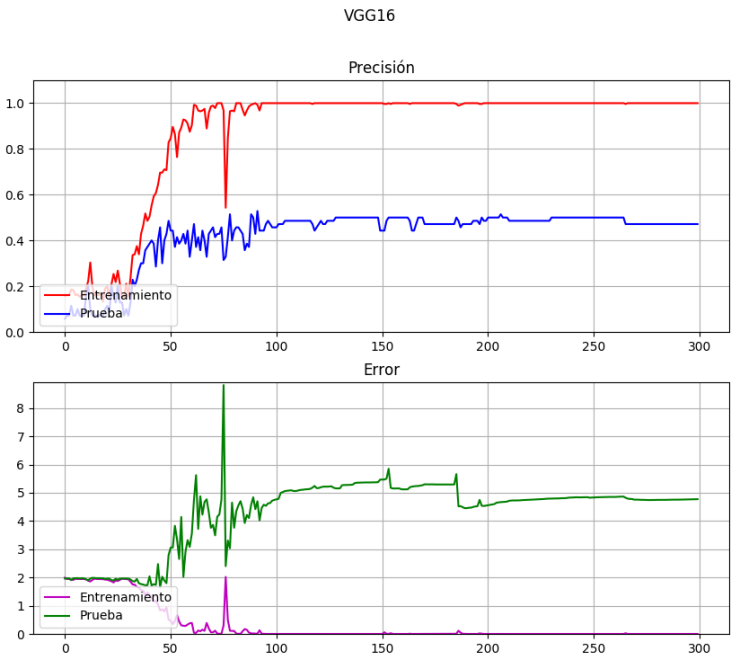

Figura B.56: Desempeño de VGG16 durante el entrenamiento por órgano, primera ronda, frutos.

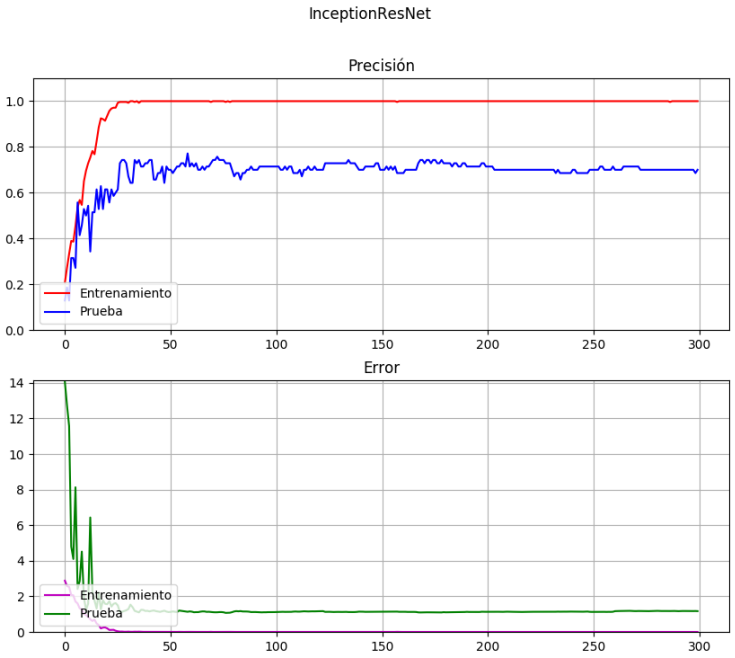

Figura B.57: Desempeño de InceptionResNet durante el entrenamiento por órgano, primera ronda, frutos.

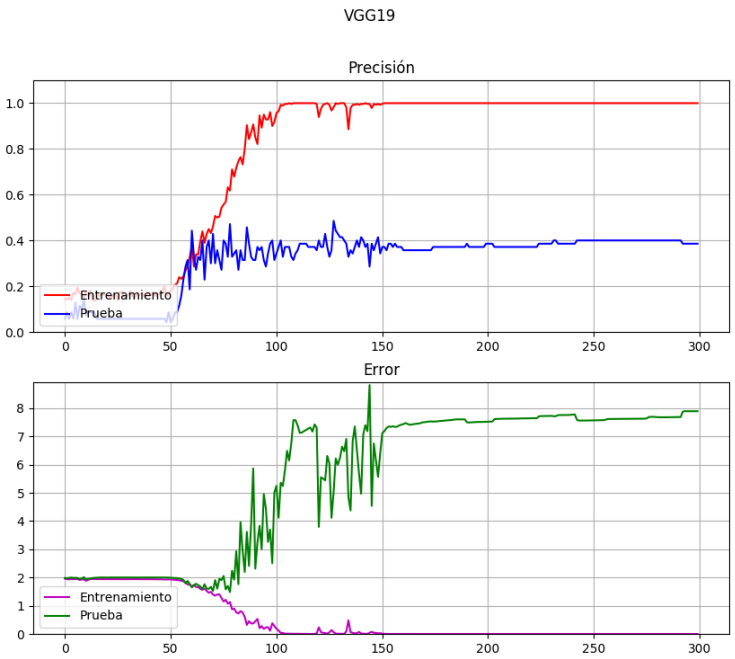

Figura B.58: Desempeño de VGG19 durante el entrenamiento por órgano, primera ronda, frutos.

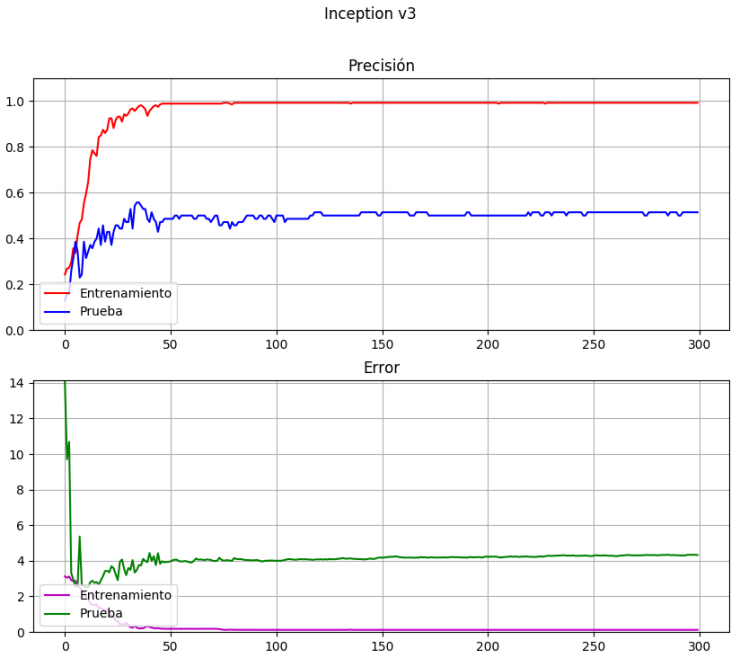

Figura B.59: Desempeño de Inception v3 durante el entrenamiento por órgano, primera ronda, frutos.

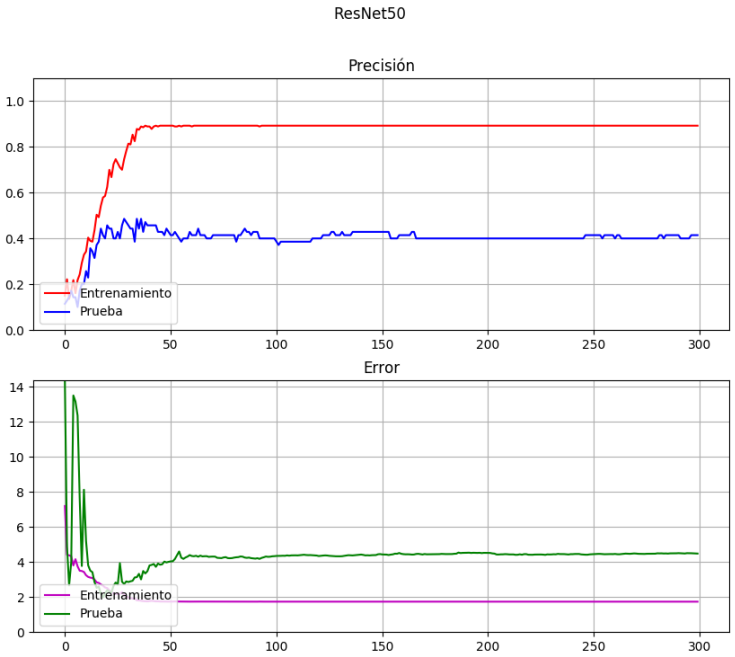

Figura B.60: Desempeño de ResNet50 durante el entrenamiento por órgano, primera ronda, frutos.

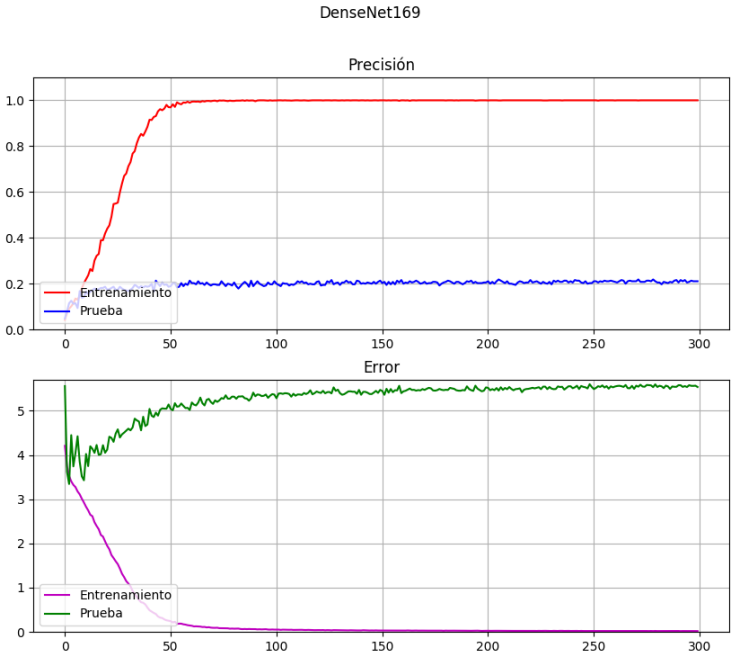

Figura B.61: Desempeño de DenseNet169 durante el entrenamiento por órgano, primera ronda, hojas.

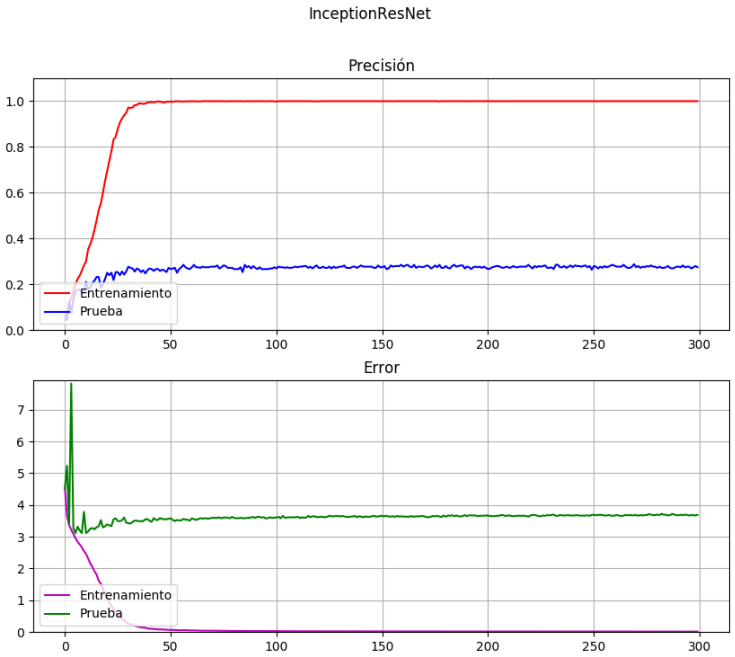

Figura B.62: Desempeño de InceptionResNet durante el entrenamiento por órgano, primera ronda, hojas.

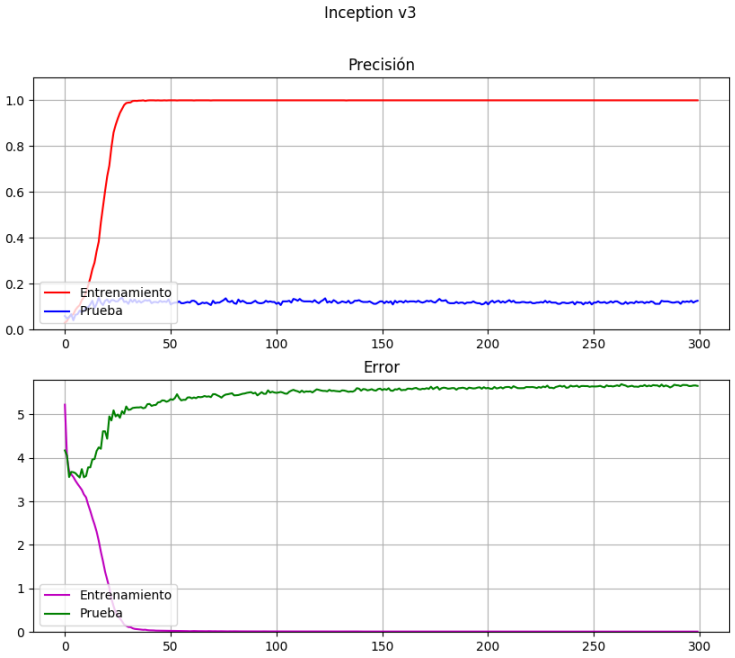

Figura B.63: Desempeño de Inception v3 durante el entrenamiento por órgano, primera ronda, hojas.

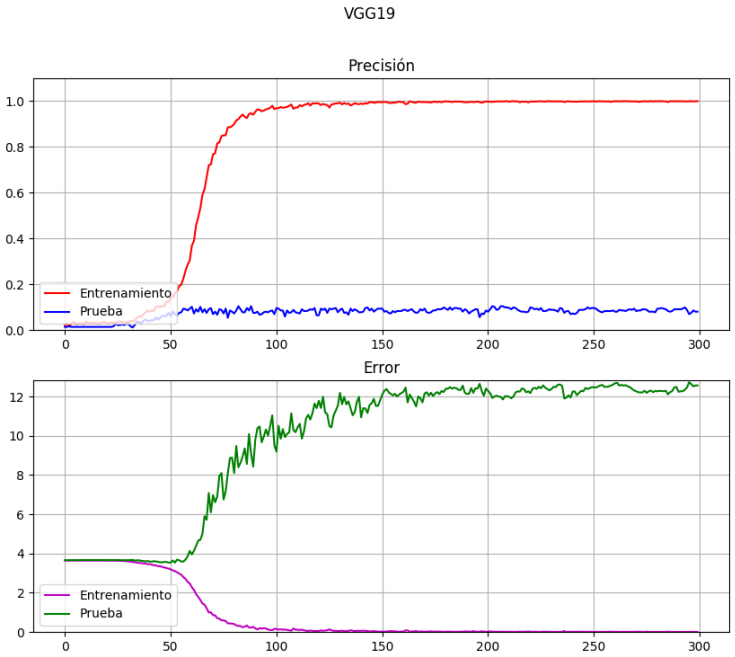

Figura B.64: Desempeño de VGG19 durante el entrenamiento por órgano, primera ronda, hojas.

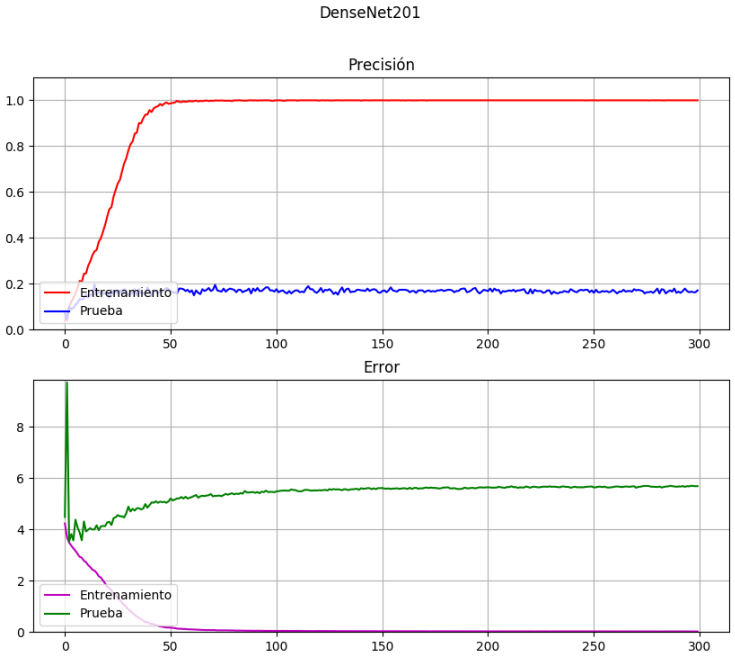

Figura B.65: Desempeño de DenseNet201 durante el entrenamiento por órgano, primera ronda, hojas.

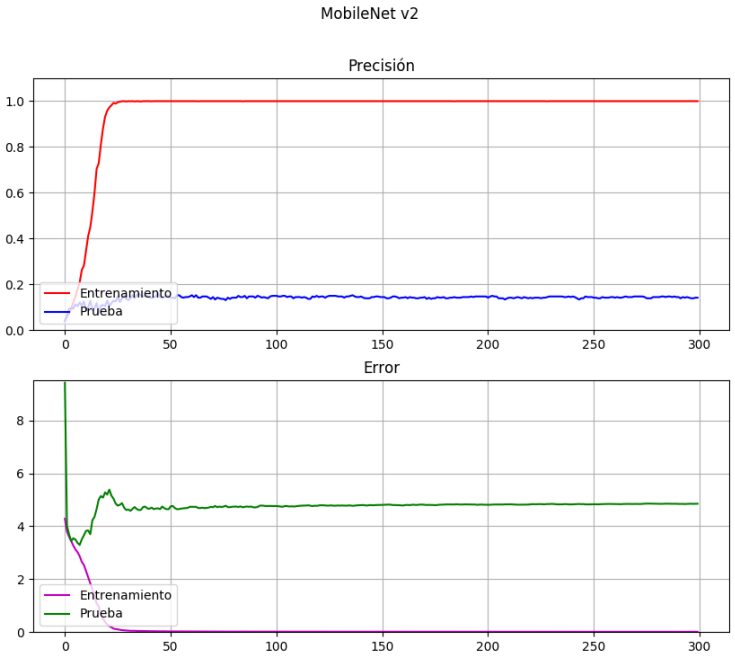

Figura B.66: Desempeño de MobileNet v2 durante el entrenamiento por órgano, primera ronda, hojas.

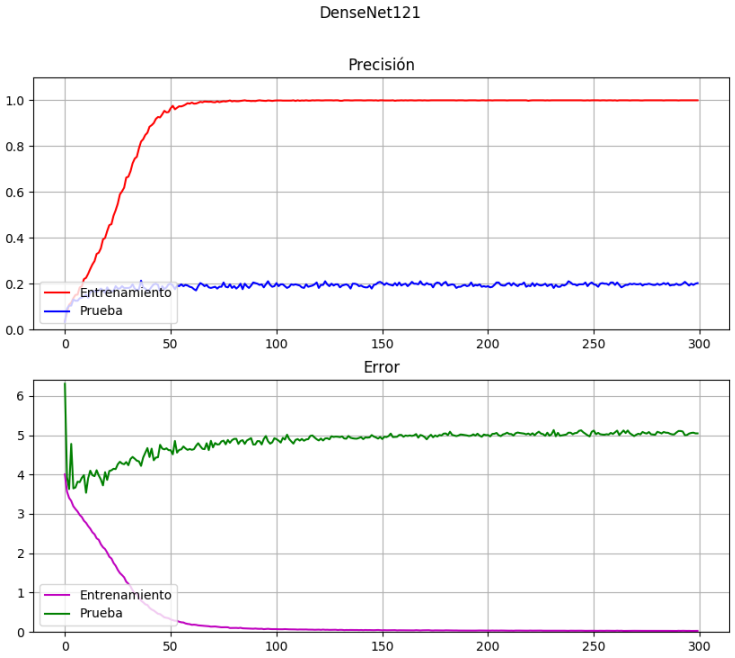

Figura B.67: Desempeño de DenseNet121 durante el entrenamiento por órgano, primera ronda, hojas.

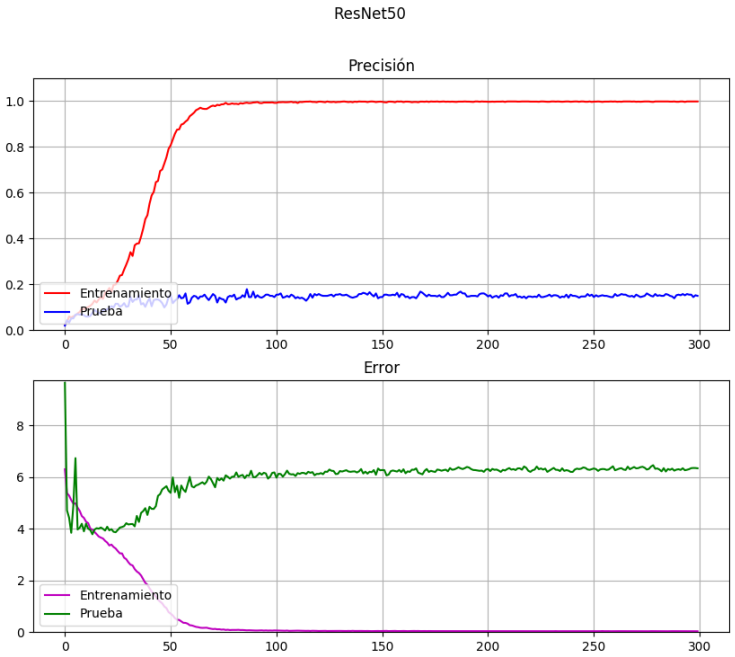

Figura B.68: Desempeño de ResNet50 durante el entrenamiento por órgano, primera ronda, hojas.

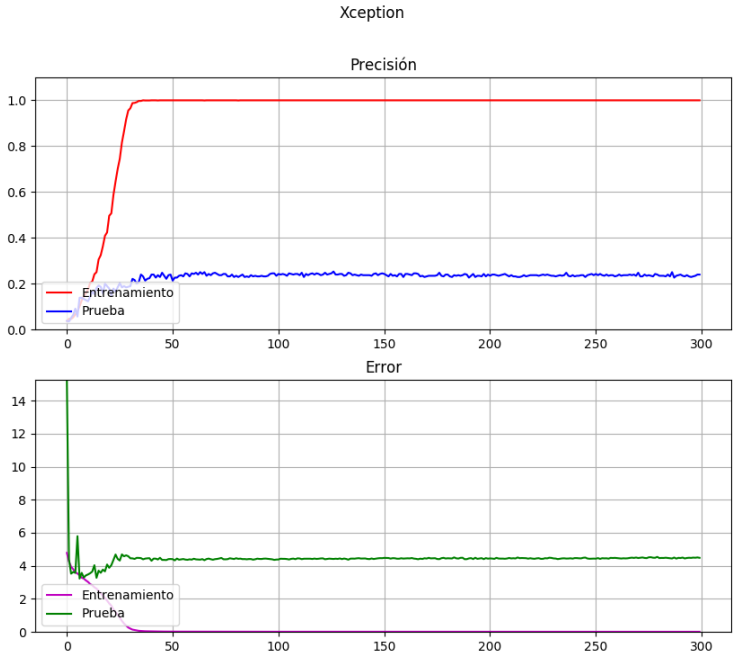

Figura B.69: Desempeño de Xception durante el entrenamiento por órgano, primera ronda, hojas.

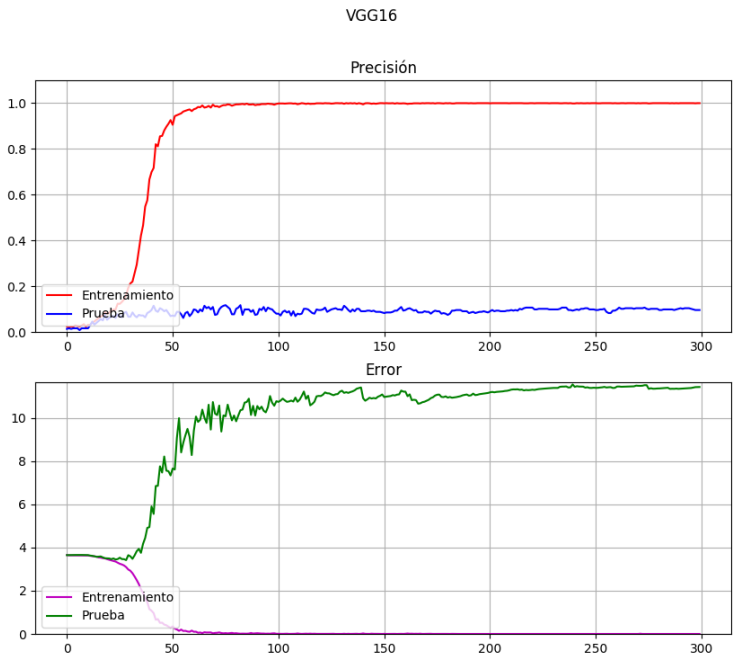

Figura B.70: Desempeño de VGG16 durante el entrenamiento por órgano, primera ronda, hojas.

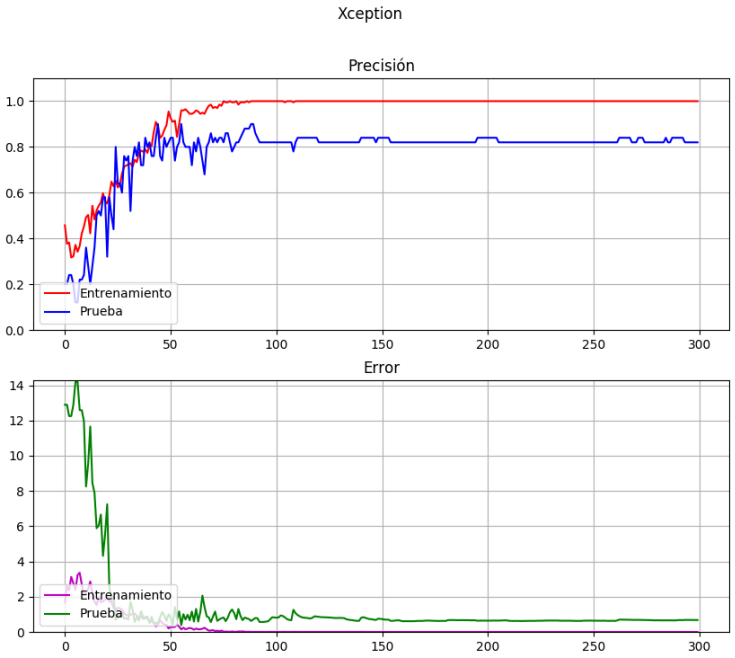

Figura B.71: Desempeño de Xception durante el entrenamiento por órgano, primera ronda, tallos.

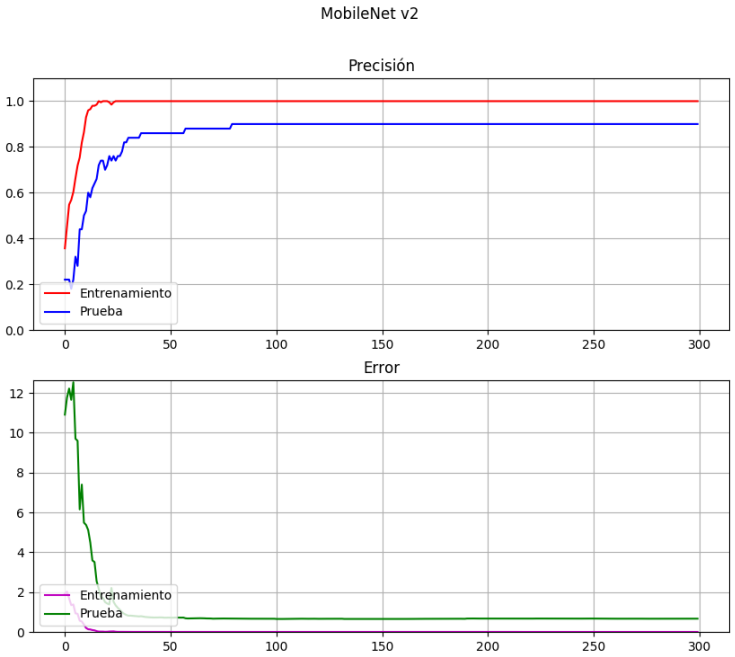

Figura B.72: Desempeño de MobileNet v2 durante el entrenamiento por órgano, primera ronda, tallos.

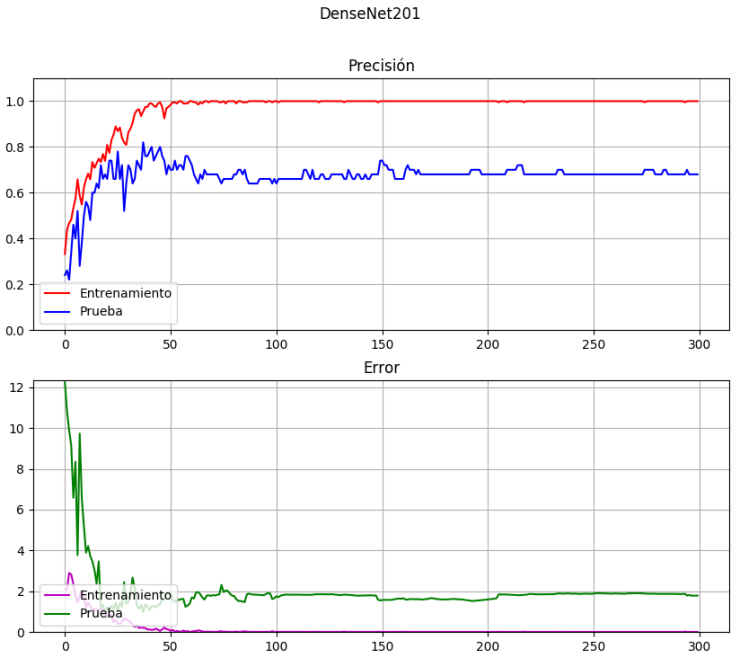

Figura B.73: Desempeño de DenseNet201 durante el entrenamiento por órgano, primera ronda, tallos.

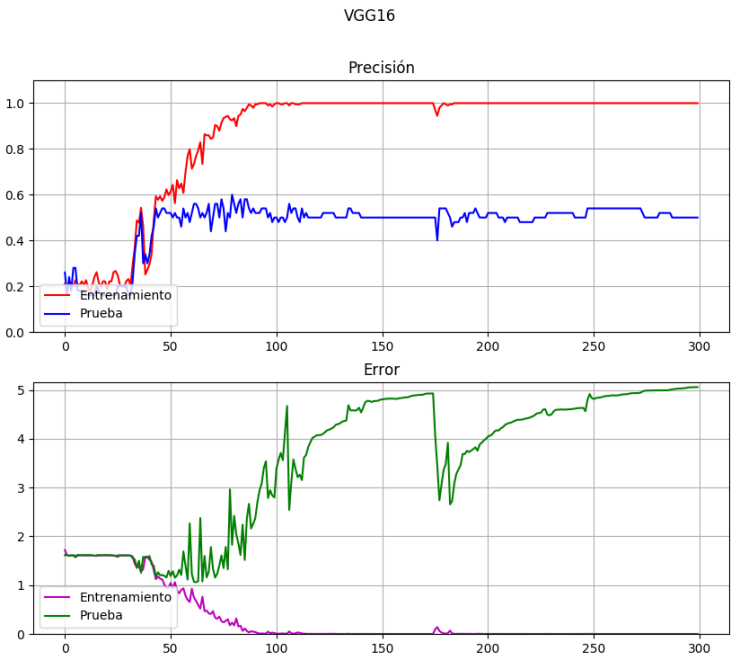

Figura B.74: Desempeño de VGG16 durante el entrenamiento por órgano, primera ronda, tallos.

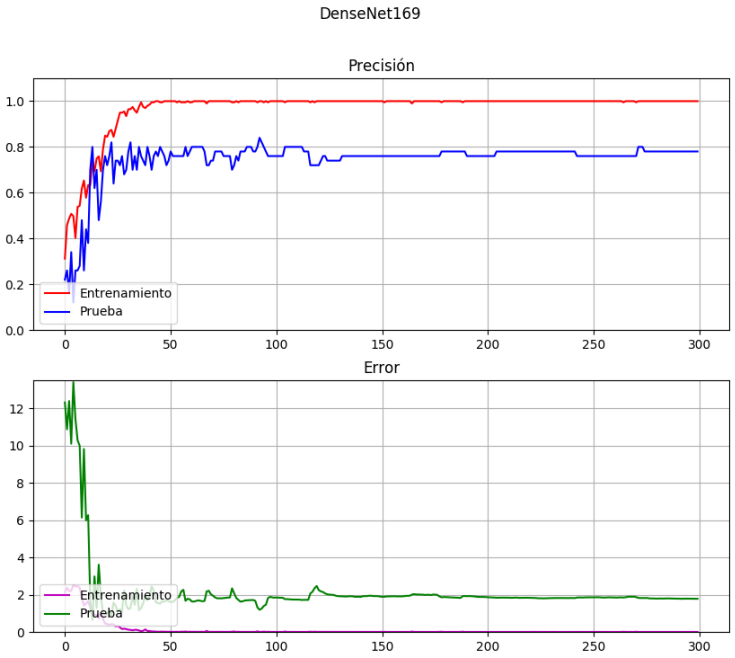

Figura B.75: Desempeño de DenseNet169 durante el entrenamiento por órgano, primera ronda, tallos.

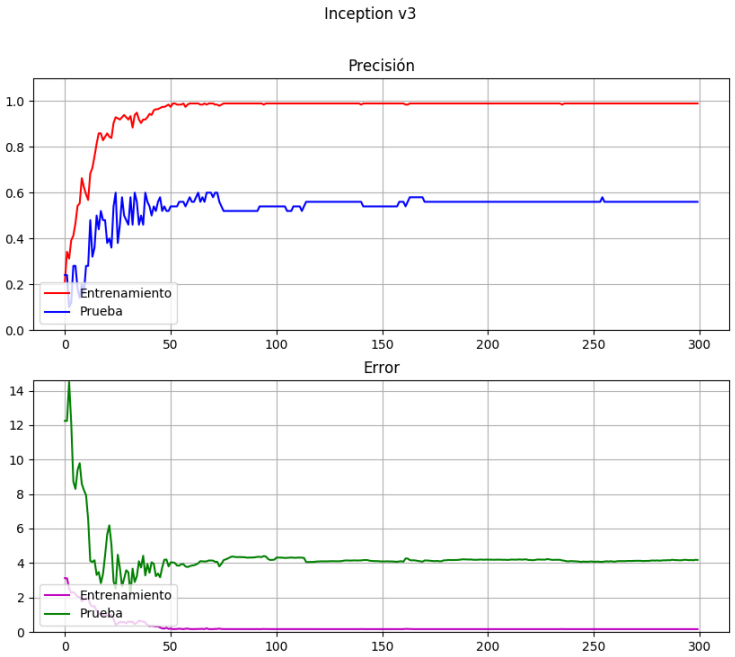

Figura B.76: Desempeño de Inception v3 durante el entrenamiento por órgano, primera ronda, tallos.

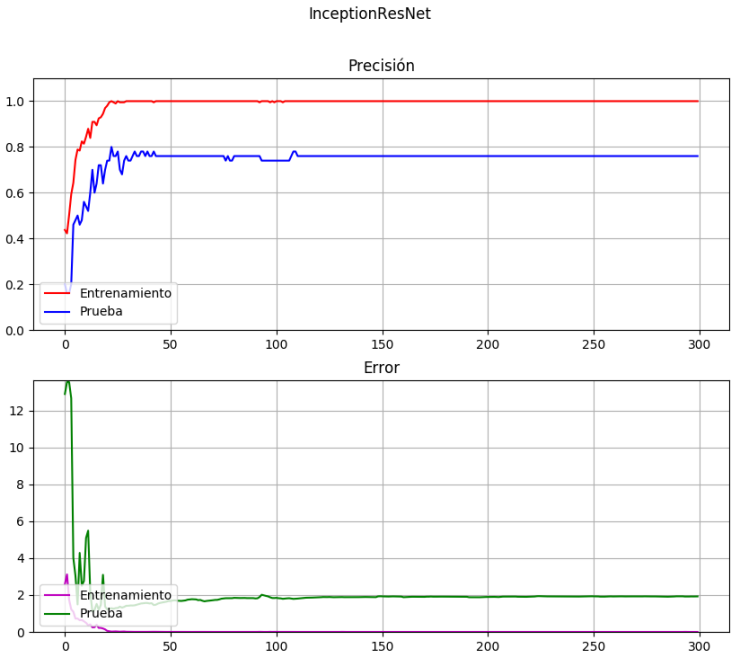

Figura B.77: Desempeño de InceptionResNet durante el entrenamiento por órgano, primera ronda, tallos.

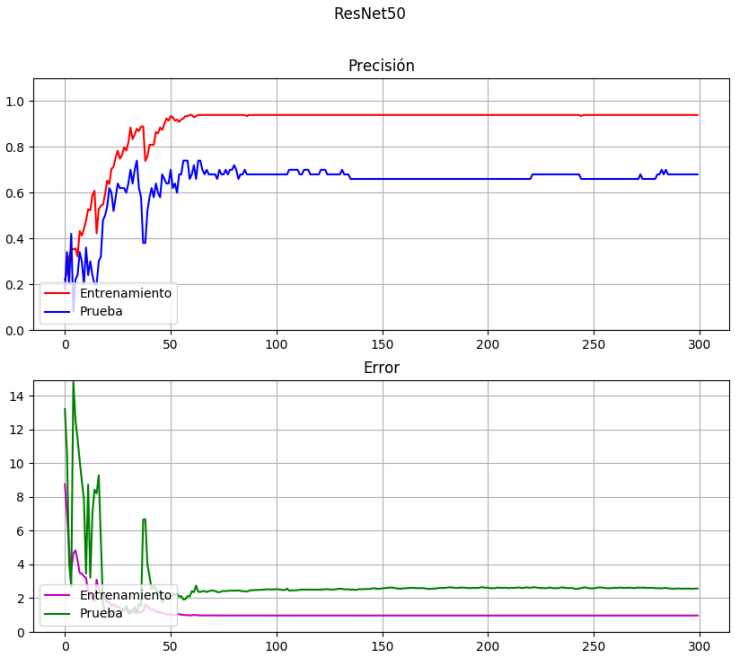

Figura B.78: Desempeño de ResNet50 durante el entrenamiento por órgano, primera ronda, tallos.

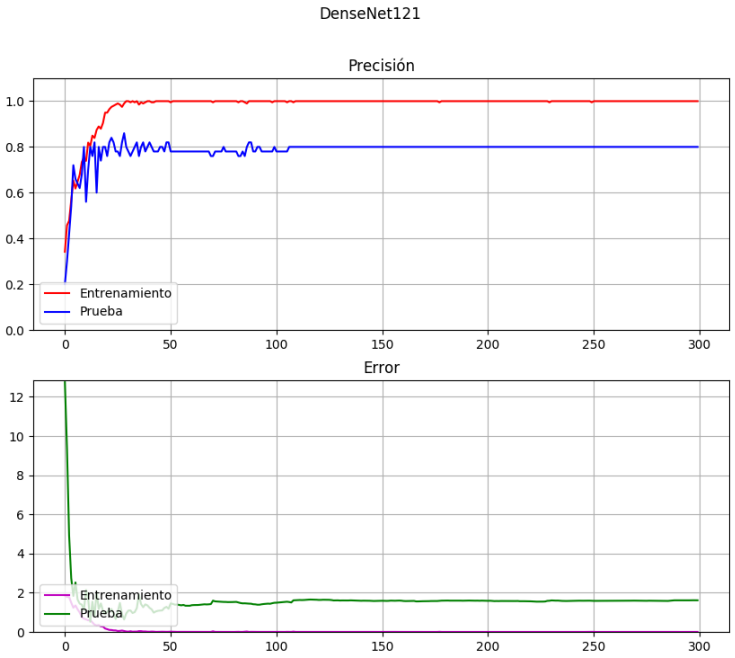

Figura B.79: Desempeño de DenseNet121 durante el entrenamiento por órgano, primera ronda, tallos.

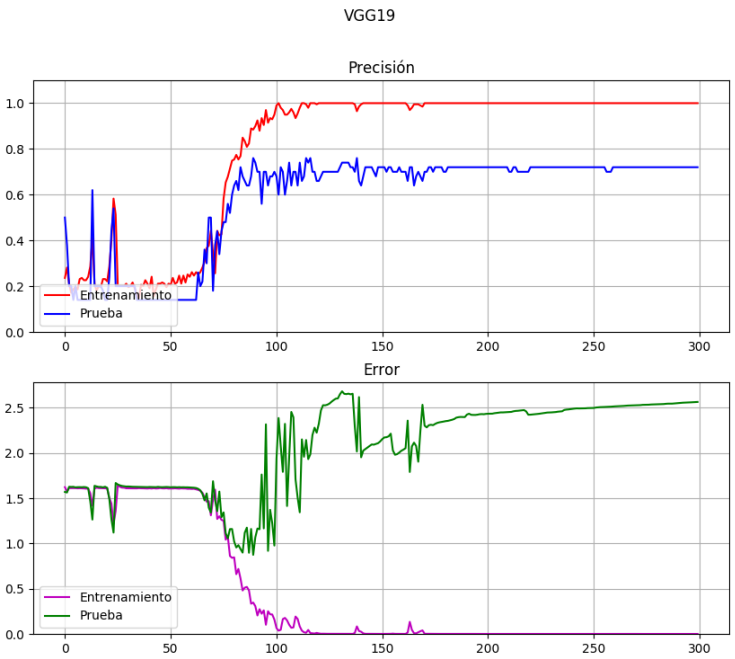

Figura B.80: Desempeño de VGG19 durante el entrenamiento por órgano, primera ronda, tallos.

Anexo C

Desempeños por clase, entrenamientos generales

En la Tabla C.1 se presenta el desempeño Top-1 de los modelos para cada clase individual. Los números en las filas corresponden a una clase particular del conjunto de datos, mientras que las cifras en cada columna de dichas filas corresponden a la precisión alcanzada por cada modelo en esa clase particular. Para facilitar la visualización de estas tablas, las casillas se colorean de rojo a medida que la precisión de los modelos bajan – es decir, a menor precisión el rojo se vuelve más intenso, mientras que este se debilita para precisiones más altas. La P al final denota el promedio del modelo para todas las clases, es decir, la precisión general del modelo.

CLS	D169	XC	D201	D121	MB2	VG19	RN50	IN3	INRN	VG16
001	0.00	0.57	0.00	0.00	0.57	0.43	0.14	0.14	0.29	0.43
002	0.29	0.71	0.43	0.29	0.43	0.14	0.14	0.57	0.57	0.71
003	0.29	0.86	0.29	0.43	0.57	0.86	0.29	0.29	0.86	0.57
004	0.00	0.14	0.00	0.14	0.43	0.71	0.14	0.14	0.14	0.86
005	0.86	0.71	1.00	0.86	0.57	0.43	0.57	0.43	1.00	0.29
006	0.00	0.00	0.00	0.00	0.43	0.29	0.00	0.43	0.14	0.71
007	0.00	0.00	0.00	0.00	0.00	0.86	0.00	0.00	0.00	0.71
008	0.14	0.00	0.14	0.14	0.29	0.14	0.29	0.00	0.29	0.00
009	0.00	0.14	0.14	0.00	0.00	0.14	0.00	0.00	0.14	0.29
010	0.14	0.43	0.29	0.14	0.29	0.29	0.00	0.29	0.29	0.14
011	0.43	0.14	0.29	0.43	0.57	0.00	0.43	0.00	0.57	0.29
012	0.00	0.00	0.00	0.14	0.00	0.29	0.00	0.14	0.14	0.14
013	0.00	0.29	0.14	0.14	0.29	0.71	0.29	0.14	0.29	0.00
014	0.43	0.57	0.43	0.43	0.14	0.43	0.29	0.43	0.57	0.43

015	0.14	0.29	0.14	0.00	0.43	0.29	0.29	0.14	0.43	0.14
016	0.00	0.29	0.00	0.00	0.14	0.00	0.14	0.29	0.14	0.14
017	0.43	1.00	0.29	0.43	1.00	0.86	0.86	0.57	0.71	0.86
018	0.29	0.43	0.14	0.29	0.29	0.29	0.29	0.00	0.43	0.14
019	0.43	0.57	0.29	0.29	0.71	0.29	0.00	0.14	0.57	0.57
020	0.00	0.14	0.00	0.00	0.00	0.00	0.00	0.00	0.14	0.14
021	0.14	0.29	0.00	0.29	0.43	0.14	0.14	0.14	0.14	0.57
022	0.43	0.57	0.57	0.57	0.57	0.71	0.43	0.71	0.71	0.86
023	0.71	0.86	1.00	0.86	0.86	0.71	0.29	0.57	0.86	0.86
024	0.00	0.00	0.00	0.00	0.00	0.38	0.00	0.00	0.00	0.38
025	0.00	0.14	0.00	0.00	0.29	0.00	0.00	0.14	0.00	0.00
026	0.57	1.00	0.57	0.86	0.86	0.43	0.43	0.57	0.86	0.57
027	0.14	0.14	0.00	0.14	0.29	0.29	0.14	0.00	0.14	0.29
028	0.29	0.29	0.29	0.29	0.00	0.14	0.43	0.14	0.43	0.14
029	0.00	0.86	0.29	0.00	0.00	0.86	0.29	0.00	0.86	0.71
030	0.14	0.29	0.29	0.14	0.29	0.14	0.43	0.43	0.57	0.14
031	0.86	1.00	0.86	0.86	0.86	0.86	0.14	0.86	0.86	0.86
032	0.17	0.33	0.33	0.33	0.33	0.00	0.50	0.17	0.50	0.17
033	0.29	0.57	0.43	0.43	0.29	0.57	0.43	0.29	0.43	0.43
034	0.86	0.86	0.71	1.00	1.00	0.43	0.57	0.86	1.00	0.86
035	0.14	0.00	0.00	0.00	0.00	0.14	0.14	0.14	0.00	0.14
036	0.29	0.14	0.29	0.14	0.14	0.00	0.14	0.14	0.14	0.14
037	0.71	0.86	0.43	0.71	0.57	0.57	0.43	0.14	0.86	0.86
038	0.43	0.43	0.29	0.14	0.43	0.14	0.29	0.29	0.57	0.00
039	0.17	0.17	0.00	0.33	0.17	0.17	0.17	0.17	0.17	0.00
040	0.29	0.14	0.00	0.00	0.14	0.00	0.00	0.14	0.14	0.00
041	0.43	0.71	0.71	0.43	0.71	0.71	0.57	0.71	0.71	0.86
042	0.00	0.14	0.00	0.00	0.14	0.86	0.14	0.00	0.00	0.71
043	0.71	0.57	0.86	0.29	0.29	0.57	0.43	0.29	0.57	0.43
044	0.00	0.29	0.14	0.14	0.00	0.14	0.14	0.14	0.29	0.14
045	0.29	0.14	0.00	0.00	0.86	0.14	0.57	0.43	0.14	0.71
046	0.71	0.57	0.57	0.57	0.43	0.29	0.43	0.57	0.71	0.29
047	0.57	0.57	0.71	0.57	0.57	0.57	0.57	0.43	0.86	0.57
048	0.43	1.00	0.14	0.14	0.86	0.71	0.43	0.29	0.71	1.00
049	0.14	0.14	0.00	0.00	0.00	0.14	0.29	0.29	0.14	0.14

050	0.14	0.43	0.14	0.43	0.43	0.71	0.43	0.43	0.71	0.29
051	0.14	0.14	0.29	0.14	0.00	0.14	0.29	0.43	0.43	0.43
052	0.14	0.14	0.14	0.14	0.00	0.00	0.29	0.00	0.29	0.00
053	0.14	0.71	0.43	0.29	0.57	0.43	0.43	0.29	0.43	0.57
054	0.00	0.14	0.00	0.00	0.00	0.00	0.29	0.14	0.00	0.43
055	0.14	0.14	0.14	0.29	0.00	0.00	0.29	0.29	0.29	0.57
056	0.00	0.00	0.14	0.00	0.00	0.00	0.29	0.14	0.14	0.00
057	0.00	0.57	0.00	0.00	0.00	0.43	0.14	0.14	0.14	0.29
058	0.43	0.57	0.14	0.43	0.00	0.00	0.57	0.00	0.43	0.29
059	0.57	0.86	0.43	0.57	0.43	0.14	0.43	0.57	0.71	0.43
060	0.43	0.57	0.29	0.29	0.71	0.71	0.14	0.14	0.57	0.29
061	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
062	0.14	0.14	0.14	0.29	0.43	0.00	0.29	0.14	0.14	0.00
063	0.57	0.43	0.57	0.57	0.71	0.00	0.29	0.43	0.57	0.29
064	0.43	0.43	0.43	0.29	0.43	0.43	0.43	0.29	0.43	0.14
065	0.14	0.14	0.00	0.00	0.14	0.14	0.14	0.14	0.00	0.14
066	0.00	0.14	0.00	0.00	0.00	0.29	0.14	0.29	0.00	0.29
067	0.57	0.57	0.00	0.29	0.43	0.57	0.29	0.29	0.71	0.86
068	0.29	0.29	0.00	0.57	0.29	0.00	0.29	0.29	0.43	0.14
069	0.29	0.71	0.43	0.14	0.86	0.57	0.43	0.29	0.43	0.57
070	0.29	0.14	0.00	0.29	0.00	0.00	0.00	0.00	0.43	0.00
071	0.57	0.71	0.71	0.14	0.86	0.29	0.71	0.43	0.71	0.57
072	0.17	0.33	0.00	0.00	0.17	0.17	0.17	0.17	0.00	0.00
073	0.00	0.00	0.00	0.00	0.43	1.00	0.00	0.00	0.00	0.71
074	0.43	0.86	0.29	0.29	0.43	0.57	0.57	0.43	0.86	1.00
075	0.43	0.43	0.29	0.29	0.43	0.71	0.29	0.43	0.57	0.71
076	0.29	0.86	0.57	0.29	0.86	0.71	0.57	0.71	0.86	0.57
077	0.00	0.14	0.00	0.00	0.14	0.00	0.14	0.00	0.14	0.00
078	0.00	0.14	0.14	0.00	0.14	0.29	0.00	0.14	0.14	0.14
079	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
080	0.29	0.71	0.29	0.29	0.86	0.86	0.29	0.57	0.71	0.71
081	0.00	0.29	0.00	0.00	0.14	0.43	0.00	0.00	0.14	0.00
082	0.29	0.57	0.29	0.14	0.71	0.29	0.71	0.71	0.43	0.57
083	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.29	0.00	0.14
084	0.57	0.57	0.43	0.43	0.14	0.00	0.57	0.43	0.57	0.29

085	0.29	0.43	0.14	0.14	0.29	0.00	0.29	0.29	0.43	0.14
086	0.43	0.57	0.43	0.14	0.29	0.14	0.00	0.29	0.29	0.00
087	0.57	0.86	0.43	0.57	0.86	0.71	0.57	0.14	0.86	0.29
088	0.29	0.14	0.29	0.57	0.14	0.43	0.43	0.29	0.29	0.00
089	0.14	0.57	0.14	0.29	0.57	0.00	0.14	0.29	0.43	0.43
090	0.00	0.00	0.00	0.00	0.00	0.14	0.00	0.00	0.14	0.00
091	0.29	0.14	0.29	0.14	0.14	0.29	0.14	0.14	0.29	0.00
092	0.71	0.57	0.57	0.43	0.14	0.14	0.57	0.14	0.43	0.00
093	0.00	0.14	0.14	0.00	0.14	0.00	0.14	0.00	0.00	0.00
094	0.43	0.43	0.29	0.14	0.71	0.57	0.43	0.14	0.43	0.57
095	0.00	0.14	0.14	0.14	0.29	0.14	0.14	0.00	0.14	0.00
096	0.29	0.29	0.43	0.43	0.29	0.14	0.57	0.57	0.43	0.00
097	0.00	0.14	0.14	0.00	0.00	0.00	0.00	0.00	0.00	0.00
098	0.00	0.29	0.00	0.00	0.14	0.29	0.14	0.00	0.14	0.00
099	0.57	0.43	0.43	0.29	0.71	0.14	0.14	0.29	0.86	0.29
100	0.86	0.86	0.71	0.86	0.71	0.29	0.86	0.71	0.86	0.57
101	0.00	0.00	0.14	0.00	0.00	0.00	0.14	0.14	0.29	0.00
102	0.71	0.71	0.57	0.29	0.57	0.29	0.14	0.43	0.14	0.29
103	0.43	0.43	0.43	0.29	0.43	0.29	0.43	0.43	0.43	0.14
104	0.43	0.57	0.43	0.14	0.86	0.71	0.00	0.71	0.57	0.71
105	1.00	1.00	1.00	0.86	1.00	0.43	0.00	0.71	1.00	0.71
106	0.00	0.00	0.00	0.00	0.14	0.57	0.00	0.00	0.00	0.71
107	0.14	0.14	0.00	0.14	0.00	0.00	0.00	0.00	0.00	0.00
108	0.43	0.71	0.14	0.43	0.00	0.14	0.71	0.43	0.86	0.14
109	0.00	0.29	0.00	0.14	0.14	0.00	0.00	0.29	0.29	0.86
110	0.00	0.00	0.00	0.00	0.00	0.00	0.14	0.00	0.00	0.00
111	0.14	0.29	0.14	0.14	0.14	0.14	0.14	0.29	0.29	0.43
112	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P	0.27	0.39	0.25	0.24	0.34	0.31	0.26	0.26	0.39	0.34

Tabla C.1: Desempeño Top-1 por clase, primera ronda. Los nombres de los modelos han sido abreviados en los encabezados de la tabla de la forma siguiente: VG16 = VGG16, VG19 = VGG19, XC = XCEPTION, MB2 = MOBILENETV2, RN50 = RESNET50, INRN = INCEPTIONRESNET, IN3 = INCEPTIONV3, D121 = DENSENET121, D169 = DENSENET169, D201 = DENSENET201

En la Tabla C.2 se presenta el desempeño Top-1 de los modelos para cada clase individual. Esta tabla se adhiere al formato descrito para la Tabla C.1.

CLS	D169	VG16	XC	INRN	RN50	D121	MB2	D201	IN3	VG19
001	0.14	0.57	0.57	0.43	0.14	0.14	0.57	0.14	0.00	0.57
002	0.29	0.00	0.43	0.57	0.86	0.57	0.00	0.57	0.00	0.00
003	0.14	0.86	0.57	0.29	0.43	0.43	0.43	0.43	0.57	0.43
004	0.29	0.43	0.00	0.00	0.00	0.29	0.71	0.14	0.43	0.71
005	0.14	0.29	0.14	0.86	0.86	0.71	0.14	0.71	0.29	0.57
006	0.00	0.71	0.29	0.00	0.00	0.00	0.14	0.29	0.00	0.86
007	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
008	0.14	0.14	0.00	0.00	0.00	0.00	0.00	0.14	0.14	0.14
009	0.14	0.14	0.43	0.14	0.29	0.00	0.00	0.00	0.00	0.14
010	0.00	0.14	0.29	0.29	0.14	0.14	0.00	0.14	0.14	0.14
011	0.29	0.29	0.00	0.43	0.71	0.14	0.43	0.57	0.14	0.00
012	0.00	0.14	0.14	0.00	0.00	0.00	0.14	0.00	0.00	0.00
013	0.00	0.57	0.29	0.14	0.14	0.00	0.00	0.14	0.14	0.43
014	0.14	0.43	0.43	0.29	0.29	0.43	0.14	0.14	0.14	0.43
015	0.00	0.14	0.00	0.14	0.43	0.00	0.14	0.00	0.14	0.14
016	0.14	0.00	0.00	0.43	0.00	0.00	0.00	0.00	0.14	0.14
017	0.57	0.43	1.00	0.57	0.86	0.14	0.43	0.43	0.86	0.57
018	0.29	0.14	0.29	0.43	0.00	0.00	0.00	0.29	0.43	0.57
019	0.14	0.43	0.29	0.43	0.00	0.14	0.29	0.14	0.00	0.43
020	0.00	0.14	0.00	0.00	0.14	0.00	0.14	0.00	0.14	0.00
021	0.43	0.57	0.29	0.00	0.29	0.14	0.43	0.14	0.14	0.14
022	0.71	0.57	0.86	0.57	0.29	0.29	0.71	0.57	0.43	0.57
023	0.86	0.71	0.71	0.86	0.14	0.86	0.29	0.57	0.57	0.57
024	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
025	0.14	0.00	0.00	0.14	0.00	0.14	0.14	0.00	0.14	0.29
026	0.14	0.57	0.86	0.57	0.57	0.14	0.43	0.14	0.43	0.71
027	0.29	0.14	0.43	0.14	0.29	0.14	0.14	0.14	0.00	0.29
028	0.00	0.00	0.14	0.57	0.14	0.29	0.14	0.14	0.00	0.00
029	0.00	1.00	0.71	1.00	0.00	0.00	0.86	0.71	0.00	0.43
030	0.00	0.14	0.00	0.43	0.57	0.00	0.71	0.14	0.14	0.14
031	0.29	0.71	0.57	1.00	0.57	0.86	0.14	0.86	0.71	0.57
032	0.17	0.00	0.50	0.17	0.00	0.17	0.00	0.17	0.17	0.00

033	0.57	0.43	0.71	0.57	0.14	0.29	0.14	0.14	0.29	0.29
034	0.00	0.71	0.71	0.86	0.57	0.86	0.29	0.57	0.71	0.71
035	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.14
036	0.00	0.00	0.14	0.29	0.29	0.14	0.14	0.00	0.14	0.14
037	0.00	0.29	0.57	1.00	0.57	0.29	0.29	0.14	0.29	0.57
038	0.43	0.14	0.71	0.43	0.29	0.00	0.43	0.57	0.14	0.14
039	0.00	0.17	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.17
040	0.14	0.00	0.14	0.29	0.00	0.14	0.00	0.00	0.00	0.00
041	0.29	0.71	0.86	0.57	0.00	0.29	0.29	0.43	0.43	0.71
042	0.14	0.86	0.14	0.00	0.00	0.14	0.14	0.00	0.00	0.71
043	0.43	0.29	0.57	0.57	0.29	0.29	0.57	0.71	0.00	0.14
044	0.00	0.14	0.57	0.29	0.00	0.29	0.14	0.14	0.00	0.00
045	0.14	0.14	0.43	0.29	0.43	0.29	0.29	0.43	0.57	0.57
046	0.43	0.43	0.57	0.57	0.00	0.14	0.14	0.29	0.57	0.00
047	0.43	0.57	0.71	0.71	0.29	0.43	0.57	0.57	0.29	0.57
048	0.57	0.57	0.71	0.43	0.86	0.14	0.29	0.57	0.71	0.57
049	0.00	0.00	0.00	0.00	0.14	0.29	0.00	0.00	0.00	0.00
050	0.14	0.57	0.29	0.43	0.29	0.43	0.43	0.29	0.71	0.43
051	0.00	0.14	0.43	0.29	0.29	0.43	0.14	0.29	0.57	0.14
052	0.00	0.00	0.14	0.14	0.00	0.00	0.00	0.14	0.14	0.14
053	0.29	0.43	0.29	0.43	0.14	0.00	0.00	0.00	0.14	0.43
054	0.00	0.43	0.29	0.00	0.29	0.00	0.00	0.00	0.14	0.14
055	0.29	0.14	0.29	0.14	0.00	0.14	0.00	0.29	0.57	0.00
056	0.00	0.14	0.14	0.14	0.14	0.00	0.00	0.00	0.00	0.00
057	0.00	0.00	0.00	0.00	0.00	0.00	0.14	0.00	0.00	0.43
058	0.14	0.43	0.57	0.14	0.29	0.57	0.14	0.14	0.14	0.14
059	0.00	0.00	0.43	0.71	0.00	0.00	0.43	0.29	0.29	0.57
060	0.57	0.29	0.14	0.14	0.71	0.00	0.00	0.29	0.00	0.29
061	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
062	0.14	0.00	0.14	0.29	0.00	0.14	0.00	0.00	0.00	0.00
063	0.57	0.29	0.43	0.57	0.57	0.57	0.29	0.43	0.43	0.14
064	0.14	0.00	0.43	0.57	0.29	0.29	0.14	0.00	0.29	0.14
065	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
066	0.14	0.00	0.14	0.29	0.14	0.14	0.14	0.14	0.14	0.00
067	0.29	0.57	0.14	0.43	0.00	0.00	0.14	0.29	0.86	0.29

068	0.14	0.00	0.29	0.43	0.00	0.29	0.57	0.57	0.14	0.00
069	0.29	0.00	0.43	0.57	0.71	0.29	0.00	0.14	0.57	0.57
070	0.29	0.00	0.14	0.00	0.00	0.57	0.14	0.00	0.29	0.00
071	0.57	0.29	0.71	0.86	0.71	0.43	0.43	0.43	0.29	0.43
072	0.17	0.00	0.17	0.00	0.00	0.17	0.17	0.17	0.17	0.00
073	0.14	0.71	0.29	0.00	0.14	0.00	0.00	0.00	0.00	1.00
074	0.57	0.57	0.57	0.86	0.43	0.29	0.43	0.14	0.71	0.71
075	0.14	0.71	0.43	0.43	0.43	0.14	0.29	0.29	0.14	0.71
076	0.14	0.57	0.57	0.86	0.00	0.57	0.29	0.14	0.86	0.57
077	0.14	0.00	0.14	0.00	0.14	0.00	0.00	0.14	0.00	0.00
078	0.00	0.29	0.14	0.29	0.00	0.14	0.14	0.00	0.00	0.43
079	0.14	0.00	0.00	0.14	0.00	0.00	0.00	0.14	0.00	0.00
080	0.00	0.71	0.57	0.43	0.57	0.00	0.43	0.14	0.43	0.71
081	0.00	0.29	0.29	0.00	0.29	0.00	0.00	0.14	0.29	0.14
082	0.14	0.29	0.57	0.43	0.57	0.14	0.29	0.14	0.29	0.14
083	0.00	0.29	0.14	0.14	0.00	0.00	0.00	0.00	0.29	0.29
084	0.29	0.00	0.29	0.29	0.29	0.29	0.14	0.29	0.29	0.00
085	0.00	0.00	0.43	0.14	0.57	0.14	0.29	0.29	0.00	0.00
086	0.00	0.00	0.29	0.14	0.29	0.14	0.14	0.00	0.14	0.00
087	0.29	0.57	0.86	0.86	0.14	0.86	0.71	0.43	0.14	0.14
088	0.29	0.14	0.57	0.43	0.57	0.14	0.14	0.43	0.00	0.14
089	0.00	0.43	0.43	0.43	0.57	0.14	0.14	0.29	0.43	0.29
090	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
091	0.00	0.14	0.14	0.43	0.29	0.29	0.43	0.29	0.00	0.00
092	0.00	0.00	0.14	0.57	0.00	0.29	0.14	0.14	0.14	0.00
093	0.14	0.00	0.00	0.14	0.14	0.00	0.00	0.14	0.00	0.00
094	0.57	0.29	0.57	0.43	0.57	0.43	0.14	0.57	0.29	0.71
095	0.00	0.00	0.14	0.29	0.14	0.00	0.14	0.00	0.00	0.00
096	0.29	0.00	0.43	0.43	0.00	0.29	0.00	0.00	0.14	0.00
097	0.00	0.14	0.00	0.14	0.14	0.00	0.00	0.43	0.00	0.00
098	0.00	0.00	0.00	0.29	0.00	0.14	0.14	0.00	0.14	0.00
099	0.29	0.14	0.29	0.57	0.00	0.14	0.29	0.00	0.29	0.14
100	0.86	0.00	0.86	1.00	0.57	0.71	0.71	1.00	0.71	0.29
101	0.00	0.00	0.14	0.00	0.14	0.00	0.00	0.14	0.00	0.00
102	0.00	0.14	0.43	0.29	0.14	0.29	0.29	0.43	0.29	0.14

103	0.29	0.86	0.14	0.43	0.00	0.29	0.14	0.43	0.43	0.29
104	0.00	0.57	0.00	0.71	0.29	0.00	0.14	0.00	0.57	0.57
105	1.00	0.43	0.86	1.00	0.00	0.86	0.43	1.00	1.00	0.57
106	0.00	0.14	0.00	0.00	0.00	0.00	0.00	0.00	0.14	0.43
107	0.00	0.14	0.00	0.14	0.00	0.00	0.00	0.00	0.00	0.14
108	0.29	0.29	0.43	0.43	0.29	0.14	0.14	0.00	0.00	0.00
109	0.14	0.29	0.14	0.14	0.14	0.00	0.00	0.14	0.00	0.86
110	0.00	0.14	0.00	0.00	0.00	0.14	0.00	0.00	0.00	0.00
111	0.14	0.00	0.29	0.43	0.00	0.14	0.14	0.14	0.00	0.14
112	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P	0.18	0.26	0.32	0.34	0.22	0.19	0.19	0.22	0.22	0.26

Tabla C.2: Desempeño Top-1 por clase, segunda ronda. Los nombres de los modelos han sido abreviados en los encabezados de la tabla de la forma siguiente: VG16 = VGG16, VG19 = VGG19, XC = XCEPTION, MB2 = MOBILENETV2, RN50 = RESNET50, INRN = INCEPTIONRESNET, IN3 = INCEPTIONV3, D121 = DENSENET121, D169 = DENSENET169, D201 = DENSENET201

En la Tabla C.3 se presenta el desempeño Top-1 de los modelos para cada clase individual. Esta tabla se adhiere al formato descrito para la Tabla C.1.

CLS	VG19	MB2	IN3	XC	D201	D121	RN50	INRN	VG16	D169
001	0.14	0.14	0.14	0.29	0.14	0.29	0.29	0.43	0.00	0.14
002	0.71	0.14	0.00	0.14	0.00	0.00	0.00	0.00	0.86	0.00
003	0.14	0.43	0.57	0.43	0.14	0.14	0.43	0.29	0.14	0.14
004	0.57	0.14	0.43	0.71	0.00	0.00	0.00	0.00	0.71	0.00
005	0.14	0.29	0.43	0.14	0.29	0.00	0.14	0.43	0.29	0.29
006	0.29	0.29	0.43	0.71	0.71	0.57	0.29	0.57	0.14	0.57
007	0.00	0.14	0.00	0.00	0.00	0.14	0.14	0.00	0.00	0.29
008	0.43	0.71	1.00	0.86	0.86	0.57	0.57	1.00	0.43	0.57
009	0.12	0.38	0.50	0.50	0.50	0.62	0.62	0.62	0.38	0.75
010	0.14	0.14	0.43	0.43	0.43	0.14	0.14	0.14	0.14	0.29
011	0.14	0.71	0.00	0.86	0.29	0.43	0.14	0.57	0.29	0.43
012	0.43	0.86	0.71	0.86	0.43	0.43	1.00	0.71	0.57	0.43
013	0.14	0.29	0.00	0.57	0.29	0.29	0.14	0.57	0.14	0.00
014	0.14	0.57	0.57	0.71	0.43	0.57	0.29	0.71	0.29	0.43

015	0.57	0.86	0.57	0.71	0.86	0.71	0.86	0.57	0.43	0.57
016	0.00	0.29	0.43	0.71	0.29	0.43	0.57	0.43	0.00	0.29
017	0.29	0.14	0.43	0.71	0.14	0.29	0.71	0.43	0.29	0.14
018	0.57	0.43	0.14	0.71	0.43	0.29	0.14	0.43	0.71	0.14
019	0.29	0.57	0.86	0.71	0.29	0.57	0.43	0.43	0.29	0.43
020	0.14	0.29	0.14	0.57	0.14	0.00	0.14	0.29	0.00	0.14
021	0.00	0.29	0.00	0.29	0.14	0.14	0.14	0.43	0.00	0.14
022	0.29	0.43	0.29	0.43	0.00	0.00	0.14	0.14	0.86	0.14
023	0.14	0.29	0.14	0.14	0.00	0.00	0.00	0.00	0.00	0.00
024	0.43	1.00	1.00	1.00	0.86	0.86	1.00	1.00	0.43	1.00
025	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.00	0.00	0.14
026	0.00	0.43	0.29	0.71	0.14	0.29	0.29	0.57	0.00	0.14
027	0.14	0.00	0.00	0.29	0.00	0.29	0.14	0.00	0.14	0.14
028	0.14	0.43	0.86	0.57	0.14	0.43	0.43	0.43	0.29	0.43
029	0.00	0.86	0.71	1.00	1.00	1.00	1.00	1.00	0.29	0.86
030	0.29	0.71	0.43	0.71	0.29	0.57	0.43	0.71	0.43	0.43
031	0.14	1.00	0.71	1.00	0.57	0.86	0.86	1.00	0.00	0.29
032	0.29	0.57	0.57	0.86	0.29	0.14	0.43	0.43	0.29	0.14
033	0.00	0.17	0.33	0.83	0.00	0.00	0.17	0.50	0.00	0.33
034	0.29	0.29	0.43	0.86	0.29	0.29	0.14	0.43	0.29	0.14
035	0.14	0.29	0.29	0.57	0.57	0.71	0.43	0.57	0.00	0.29
036	0.33	0.33	0.00	0.67	0.17	0.33	0.33	0.67	0.33	0.17
037	0.14	0.14	0.00	0.14	0.14	0.14	0.14	0.14	0.00	0.00
038	0.00	0.29	0.00	0.00	0.00	0.00	0.00	0.00	0.43	0.00
039	0.43	0.00	0.29	0.43	0.00	0.14	0.00	0.29	0.00	0.00
040	0.14	0.43	0.43	0.43	0.43	0.43	0.29	0.71	0.14	0.43
041	0.43	0.71	0.57	1.00	0.43	0.71	0.57	1.00	0.29	0.57
042	0.00	0.00	0.00	0.14	0.14	0.14	0.14	0.14	0.00	0.29
043	0.14	0.71	0.14	0.71	0.57	0.43	0.71	0.86	0.43	0.43
044	0.00	0.14	0.00	0.14	0.14	0.14	0.00	0.14	0.00	0.00
045	0.00	0.00	0.00	0.29	0.14	0.00	0.00	0.14	0.29	0.14
046	0.00	0.29	0.00	0.00	0.00	0.14	0.00	0.00	0.00	0.00
047	0.57	0.71	0.29	1.00	0.29	0.29	0.29	1.00	0.57	0.00
048	0.14	0.00	0.00	0.29	0.14	0.00	0.29	0.29	0.00	0.14
049	0.14	0.00	0.14	0.29	0.00	0.00	0.43	0.29	0.00	0.00

050	0.14	0.86	0.86	1.00	0.71	1.00	0.57	1.00	0.00	0.86
051	0.29	0.71	0.57	0.86	0.57	0.71	0.57	0.86	0.14	1.00
052	0.29	0.43	0.43	0.29	0.00	0.29	0.29	0.29	0.57	0.43
053	0.00	0.29	0.00	0.00	0.00	0.00	0.00	0.00	0.29	0.00
054	0.00	0.00	0.14	0.57	0.29	0.00	0.14	0.29	0.14	0.29
055	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.71	0.00
056	0.14	0.43	0.00	0.71	0.43	0.57	0.29	0.71	0.00	0.29
057	0.43	0.71	0.71	1.00	0.43	0.57	0.29	0.86	0.43	0.57
058	0.14	0.71	0.43	0.86	0.43	0.57	0.43	0.71	0.57	0.29
059	0.00	0.43	0.43	0.57	0.00	0.29	0.29	0.14	0.14	0.14
060	0.29	0.71	1.00	0.43	0.14	0.14	0.00	0.57	0.43	0.14
061	0.14	0.57	0.14	0.14	0.14	0.29	0.00	0.29	0.14	0.00
062	0.29	1.00	0.43	1.00	0.43	0.43	0.43	0.86	0.29	0.43
063	0.00	0.14	0.14	0.00	0.00	0.14	0.14	0.00	0.00	0.14
064	0.00	0.29	0.14	0.43	0.00	0.00	0.14	0.00	0.00	0.14
065	0.00	0.14	0.29	0.29	0.14	0.29	0.14	0.00	0.00	0.14
066	0.00	0.43	0.57	0.29	0.14	0.00	0.00	0.14	0.00	0.00
067	0.14	0.14	0.00	0.71	0.14	0.00	0.29	0.14	0.14	0.00
068	0.14	0.14	0.00	0.43	0.43	0.14	0.14	0.14	0.14	0.00
069	0.00	0.43	0.43	0.71	0.29	0.29	0.29	0.57	0.00	0.43
070	0.14	0.71	1.00	0.71	1.00	0.86	0.71	1.00	0.71	0.86
071	0.00	0.14	0.00	0.14	0.00	0.00	0.00	0.14	0.00	0.00
072	0.00	0.43	0.57	0.43	0.43	0.43	0.29	0.43	0.00	0.43
073	0.14	0.43	0.43	0.57	0.71	0.43	0.57	0.57	0.43	0.57
074	0.43	0.86	1.00	1.00	0.86	1.00	0.86	0.86	0.71	0.86
075	0.57	0.71	0.57	1.00	0.71	0.71	0.57	0.71	0.71	0.57
076	0.00	0.29	0.00	0.43	0.14	0.29	0.00	0.29	0.00	0.14
077	0.14	0.57	0.29	0.71	0.57	0.71	0.00	0.57	0.43	0.71
078	0.57	0.71	0.29	0.86	0.71	0.57	0.29	0.86	0.57	0.57
079	0.00	0.29	0.14	0.57	0.14	0.14	0.29	0.29	0.00	0.00
080	0.00	0.00	0.29	0.14	0.00	0.00	0.00	0.00	0.00	0.00
081	0.14	0.00	0.00	0.00	0.14	0.00	0.00	0.00	0.14	0.00
082	0.14	0.29	0.14	0.43	0.00	0.43	0.29	0.43	0.00	0.29
083	0.14	0.57	0.57	0.71	0.43	0.29	0.14	0.00	0.57	0.14
084	0.29	0.14	0.00	0.57	0.29	0.43	0.14	0.14	0.00	0.57

085	0.14	0.43	0.43	0.43	0.29	0.14	0.14	0.57	0.00	0.14
086	0.71	0.71	0.71	0.86	0.14	0.00	0.14	0.71	0.57	0.14
087	0.00	0.14	0.00	0.43	0.14	0.14	0.14	0.29	0.00	0.43
088	0.43	0.43	0.29	1.00	0.14	0.71	0.14	0.29	0.43	0.43
089	0.71	0.86	0.86	0.86	0.71	0.57	0.57	0.86	0.57	0.71
090	0.00	0.33	0.17	0.83	0.00	0.17	0.17	0.17	0.00	0.17
091	0.43	0.86	0.29	1.00	0.43	0.29	0.00	1.00	0.29	0.43
092	0.14	0.71	0.43	0.71	0.43	0.71	0.43	0.57	0.14	1.00
093	0.29	0.29	0.57	0.57	0.14	0.00	0.14	0.43	0.14	0.00
094	0.29	0.29	0.14	0.43	0.29	0.29	0.00	0.43	0.29	0.29
095	0.29	0.29	0.00	0.29	0.00	0.14	0.14	0.14	0.14	0.00
096	0.00	0.00	0.29	0.14	0.00	0.00	0.00	0.00	0.00	0.00
097	0.29	0.29	0.14	0.57	0.43	0.43	0.29	0.57	0.14	0.29
098	0.00	0.00	0.14	0.29	0.00	0.14	0.14	0.00	0.14	0.29
099	0.00	0.00	0.14	0.29	0.14	0.14	0.14	0.00	0.14	0.29
100	0.00	0.14	0.29	0.43	0.43	0.43	0.29	0.57	0.00	0.57
101	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
102	0.14	0.71	0.29	0.29	0.43	0.29	0.29	0.43	0.00	0.43
103	0.00	0.43	0.29	0.29	0.43	0.14	0.57	0.43	0.00	0.43
104	0.43	0.86	0.43	0.86	0.71	0.86	0.43	0.86	0.43	0.71
105	0.57	0.86	0.71	1.00	0.43	0.29	0.71	1.00	0.86	0.86
106	0.29	0.29	0.00	0.43	0.29	0.29	0.14	0.43	0.29	0.14
107	0.29	0.43	0.71	0.71	0.57	0.43	0.29	1.00	0.14	0.43
108	0.29	1.00	1.00	1.00	0.86	1.00	1.00	1.00	0.00	0.71
109	0.00	0.00	0.14	0.00	0.14	0.00	0.14	0.00	0.14	0.00
110	0.00	0.14	0.29	0.14	0.29	0.29	0.14	0.29	0.00	0.43
111	0.00	0.43	0.29	0.57	0.29	0.43	0.14	0.71	0.29	0.14
112	0.14	0.43	0.29	0.29	0.29	0.29	0.14	0.43	0.14	0.43
P	0.20	0.40	0.33	0.53	0.30	0.32	0.28	0.43	0.23	0.30

Tabla C.3: Desempeño Top-1 por clase, tercera ronda. Los nombres de los modelos han sido abreviados en los encabezados de la tabla de la forma siguiente: VG16 = VGG16, VG19 = VGG19, XC = XCEPTION, MB2 = MOBILENETV2, RN50 = RESNET50, INRN = INCEPTIONRESNET, IN3 = INCEPTIONV3, D121 = DENSENET121, D169 = DENSENET169, D201 = DENSENET201

En la Tabla C.4 se presenta el desempeño Top-1 de los modelos para cada clase individual. Esta tabla se adhiere al formato descrito para la Tabla C.1.

CLS	IN3	D201	VG19	INRN	XC	VG16	D121	RN50	D169	MB2
001	0.29	0.14	0.00	0.43	0.14	0.14	0.29	0.43	0.43	0.29
002	0.29	0.00	1.00	0.00	0.14	0.86	0.00	0.29	0.14	0.14
003	0.29	0.14	0.57	0.29	0.00	0.57	0.29	0.00	0.14	0.14
004	0.43	0.00	0.43	0.00	0.00	0.71	0.00	0.00	0.00	0.43
005	0.43	0.14	0.29	0.43	0.43	0.43	0.71	0.00	0.43	0.29
006	0.29	0.57	0.00	0.57	0.57	0.14	0.43	0.14	0.43	0.14
007	0.00	0.14	0.14	0.00	0.00	0.00	0.00	0.00	0.14	0.29
008	0.43	0.29	0.00	0.43	0.57	0.86	0.43	0.43	0.29	0.29
009	0.25	0.38	0.12	0.50	0.25	0.25	0.25	0.12	0.25	0.25
010	0.43	0.57	0.43	0.43	0.43	0.14	0.43	0.14	0.29	0.29
011	0.43	0.43	0.43	0.86	0.57	0.57	0.71	0.00	0.43	0.14
012	0.43	0.43	0.43	0.71	0.71	0.57	0.43	0.29	0.14	0.57
013	0.14	0.14	0.29	0.29	0.00	0.43	0.14	0.00	0.00	0.00
014	0.29	0.29	0.14	0.43	0.71	0.29	0.43	0.00	0.57	0.14
015	0.29	0.43	0.57	0.71	0.71	0.57	0.86	0.00	0.86	0.43
016	0.29	0.57	0.00	0.43	0.00	0.14	0.14	0.43	0.29	0.29
017	0.14	0.00	0.43	0.14	0.29	0.43	0.00	0.00	0.00	0.00
018	0.43	0.29	0.71	0.43	0.00	0.86	0.43	0.00	0.29	0.57
019	0.14	0.29	0.29	0.57	0.43	0.29	0.43	0.14	0.29	0.86
020	0.00	0.29	0.14	0.00	0.00	0.14	0.29	0.00	0.29	0.00
021	0.14	0.00	0.00	0.14	0.00	0.00	0.00	0.14	0.14	0.29
022	0.43	0.00	0.57	0.29	0.29	0.57	0.29	0.00	0.14	0.00
023	0.00	0.43	0.14	0.00	0.00	0.00	0.14	0.29	0.14	0.00
024	0.71	0.57	0.43	0.71	0.86	0.57	0.71	0.00	0.86	1.00
025	0.00	0.00	0.00	0.14	0.29	0.14	0.14	0.14	0.00	0.00
026	0.14	0.14	0.14	0.71	0.14	0.14	0.43	0.14	0.00	0.14
027	0.29	0.14	0.00	0.00	0.00	0.00	0.14	0.00	0.00	0.00
028	0.29	0.43	0.29	0.43	0.00	0.29	0.43	0.00	0.29	0.43
029	0.57	1.00	0.43	1.00	1.00	0.29	1.00	1.00	0.86	0.57
030	0.14	0.14	0.43	0.29	0.71	0.57	0.00	0.00	0.00	0.29
031	0.29	0.14	0.57	1.00	0.86	0.57	0.71	0.57	0.57	0.71
032	0.14	0.14	0.43	0.57	0.29	0.57	0.14	0.57	0.29	0.43

033	0.00	0.00	0.00	0.50	0.33	0.00	0.17	0.00	0.00	0.00
034	0.14	0.43	0.29	0.29	0.29	0.57	0.43	0.29	0.57	0.57
035	0.00	0.29	0.00	0.29	0.71	0.00	0.43	0.00	0.43	0.14
036	0.00	0.33	0.17	0.17	0.50	0.17	0.33	0.00	0.50	0.17
037	0.43	0.00	0.00	0.29	0.14	0.00	0.00	0.29	0.00	0.00
038	0.14	0.14	0.43	0.14	0.14	0.29	0.00	0.00	0.00	0.00
039	0.00	0.00	0.00	0.43	0.00	0.29	0.00	0.00	0.00	0.00
040	0.29	0.14	0.14	0.57	0.14	0.00	0.29	0.29	0.14	0.00
041	0.14	0.57	0.57	0.57	0.43	0.29	0.43	0.29	0.29	0.14
042	0.00	0.00	0.14	0.00	0.14	0.14	0.00	0.00	0.00	0.29
043	0.43	0.43	0.14	0.71	0.43	0.43	0.29	0.71	0.29	0.43
044	0.29	0.00	0.00	0.14	0.00	0.00	0.14	0.14	0.00	0.00
045	0.14	0.00	0.00	0.14	0.29	0.00	0.00	0.00	0.14	0.00
046	0.00	0.14	0.00	0.29	0.14	0.00	0.14	0.00	0.00	0.00
047	0.43	0.00	0.43	1.00	0.14	0.57	0.14	0.29	0.14	0.29
048	0.00	0.00	0.00	0.00	0.14	0.00	0.00	0.00	0.00	0.14
049	0.00	0.00	0.14	0.14	0.00	0.14	0.14	0.00	0.14	0.29
050	0.29	0.43	0.00	0.57	0.71	0.14	0.86	0.29	0.71	0.29
051	0.57	0.71	0.00	0.86	0.86	0.00	0.57	0.57	0.71	0.14
052	0.00	0.14	0.43	0.29	0.29	0.57	0.14	0.14	0.14	0.00
053	0.00	0.00	0.00	0.14	0.00	0.14	0.00	0.00	0.00	0.00
054	0.14	0.00	0.00	0.29	0.14	0.00	0.43	0.14	0.00	0.14
055	0.43	0.14	0.71	0.00	0.00	0.57	0.00	0.00	0.00	0.00
056	0.29	0.43	0.00	0.57	0.43	0.00	0.57	0.00	0.14	0.14
057	0.14	0.57	0.29	0.71	0.57	0.29	0.57	0.86	0.57	0.14
058	0.14	0.43	0.29	0.43	0.43	0.43	0.29	0.71	0.14	0.14
059	0.00	0.14	0.00	0.00	0.29	0.00	0.29	0.43	0.00	0.14
060	0.00	0.00	0.57	0.29	0.29	0.00	0.14	0.14	0.00	0.71
061	0.00	0.14	0.14	0.14	0.00	0.00	0.00	0.14	0.00	0.00
062	0.14	0.29	0.29	0.57	0.57	0.14	0.29	0.29	0.29	0.29
063	0.00	0.00	0.00	0.00	0.14	0.00	0.00	0.00	0.00	0.00
064	0.00	0.14	0.00	0.14	0.14	0.00	0.00	0.00	0.14	0.14
065	0.00	0.14	0.29	0.14	0.00	0.14	0.14	0.14	0.00	0.00
066	0.00	0.00	0.14	0.00	0.00	0.14	0.00	0.00	0.00	0.00
067	0.57	0.00	0.14	0.00	0.14	0.43	0.00	0.14	0.14	0.57

068	0.00	0.00	0.14	0.29	0.00	0.00	0.00	0.14	0.00	0.00
069	0.29	0.29	0.00	0.14	0.14	0.14	0.43	0.14	0.29	0.29
070	0.29	0.71	0.43	1.00	0.71	0.57	0.14	0.57	0.86	0.57
071	0.00	0.00	0.29	0.00	0.00	0.14	0.00	0.00	0.00	0.00
072	0.00	0.00	0.00	0.43	0.29	0.00	0.14	0.43	0.14	0.14
073	0.14	0.43	0.29	0.43	0.14	0.43	0.57	0.00	0.57	0.14
074	0.71	0.71	0.57	1.00	0.71	0.71	0.71	0.29	0.71	0.00
075	0.29	0.57	0.57	0.57	0.43	1.00	0.71	0.29	0.71	0.29
076	0.00	0.00	0.00	0.00	0.29	0.00	0.29	0.00	0.00	0.00
077	0.14	0.29	0.29	0.71	0.86	0.43	0.29	0.71	0.43	0.29
078	0.43	0.29	0.00	0.71	0.71	0.29	0.29	0.43	0.29	0.29
079	0.00	0.14	0.00	0.29	0.14	0.00	0.29	0.14	0.00	0.00
080	0.00	0.00	0.29	0.00	0.00	0.29	0.00	0.00	0.00	0.14
081	0.00	0.00	0.00	0.00	0.00	0.14	0.00	0.00	0.00	0.00
082	0.29	0.29	0.00	0.57	0.43	0.00	0.14	0.14	0.43	0.00
083	0.57	0.00	0.71	0.43	0.71	0.71	0.29	0.14	0.43	0.14
084	0.43	0.14	0.00	0.14	0.14	0.00	0.29	0.14	0.00	0.00
085	0.00	0.00	0.00	0.29	0.00	0.00	0.14	0.00	0.00	0.29
086	0.57	0.14	0.86	0.43	0.29	0.86	0.14	0.14	0.43	0.29
087	0.00	0.14	0.00	0.14	0.00	0.14	0.14	0.29	0.14	0.14
088	0.29	0.29	0.29	0.71	0.57	0.57	0.00	0.43	0.57	0.43
089	0.43	0.29	0.57	0.71	0.86	0.57	0.29	0.57	0.57	0.29
090	0.00	0.00	0.00	0.00	0.17	0.00	0.00	0.00	0.00	0.00
091	0.29	0.00	0.29	0.57	0.14	0.43	0.57	0.57	0.14	0.14
092	0.00	0.29	0.14	0.71	0.71	0.29	0.57	0.43	0.57	0.14
093	0.14	0.00	0.00	0.14	0.00	0.29	0.00	0.00	0.00	0.00
094	0.14	0.43	0.57	0.29	0.43	0.00	0.00	0.29	0.43	0.14
095	0.14	0.00	0.00	0.14	0.14	0.00	0.00	0.14	0.14	0.00
096	0.14	0.00	0.00	0.14	0.00	0.00	0.14	0.00	0.14	0.14
097	0.14	0.57	0.00	0.29	0.43	0.29	0.43	0.57	0.57	0.14
098	0.14	0.14	0.00	0.43	0.00	0.00	0.00	0.00	0.14	0.00
099	0.00	0.00	0.14	0.00	0.14	0.00	0.00	0.14	0.14	0.14
100	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.00	0.14
101	0.00	0.00	0.14	0.00	0.00	0.00	0.00	0.14	0.00	0.00
102	0.14	0.00	0.00	0.29	0.29	0.00	0.43	0.29	0.29	0.57

103	0.14	0.29	0.14	0.43	0.29	0.00	0.29	0.29	0.14	0.14
104	0.71	0.14	0.57	0.86	0.71	0.57	0.57	0.43	0.29	0.43
105	0.43	0.43	0.71	0.57	0.71	0.71	0.71	0.14	0.57	0.29
106	0.14	0.14	0.29	0.29	0.14	0.29	0.14	0.00	0.14	0.14
107	0.43	0.43	0.57	0.57	0.29	0.29	0.29	0.00	0.29	0.14
108	0.57	0.86	0.29	1.00	0.86	0.71	0.86	0.14	0.86	0.00
109	0.14	0.00	0.00	0.14	0.14	0.00	0.00	0.00	0.00	0.14
110	0.14	0.00	0.00	0.29	0.14	0.00	0.14	0.00	0.00	0.00
111	0.14	0.14	0.00	0.57	0.29	0.14	0.14	0.00	0.29	0.29
112	0.00	0.29	0.00	0.29	0.14	0.14	0.14	0.00	0.14	0.14
P	0.21	0.21	0.22	0.37	0.30	0.26	0.26	0.18	0.24	0.19

Tabla C.4: Desempeño Top-1 por clase, cuarta ronda. Los nombres de los modelos han sido abreviados en los encabezados de la tabla de la forma siguiente: VG16 = VGG16, VG19 = VGG19, XC = XCEPTION, MB2 = MOBILENETV2, RN50 = RESNET50, INRN = INCEPTIONRESNET, IN3 = INCEPTIONV3, D121 = DENSENET121, D169 = DENSENET169, D201 = DENSENET201

Anexo D

Desempeños por clase, entrenamientos por órgano

CLS	D121	D169	D201	INRN	IN3	MB2	RN50	VG16	VG19	XC
001	0.86	0.71	0.71	0.86	0.43	0.71	0.57	0.86	0.14	0.86
002	0.57	0.86	0.71	0.86	0.71	0.57	0.57	0.86	0.86	0.43
003	0.86	0.43	0.57	1.00	0.86	0.86	0.71	1.00	0.14	0.57
004	0.29	0.14	0.29	0.29	0.71	0.43	0.14	0.86	0.00	0.71
005	0.86	1.00	0.57	1.00	1.00	1.00	0.43	0.71	0.57	0.86
006	0.14	0.29	1.00	0.71	0.57	0.57	0.71	0.86	0.29	1.00
007	0.14	0.14	0.00	0.57	0.57	0.14	0.29	1.00	0.71	0.14
008	1.00	0.43	0.43	0.57	0.57	0.71	0.71	1.00	0.29	0.43
009	0.00	0.43	0.14	0.86	0.29	0.29	0.57	0.86	0.00	0.57
010	0.71	0.43	1.00	0.86	0.86	0.71	0.57	1.00	0.57	0.43
011	0.00	0.43	0.00	0.71	0.29	0.29	0.43	0.86	0.29	0.29
012	0.43	0.71	0.86	0.43	0.71	0.71	0.29	0.57	0.29	0.43
013	0.43	0.57	0.57	0.71	0.57	0.57	0.57	0.86	0.29	0.71
014	0.71	0.86	0.57	0.71	0.43	1.00	0.43	0.71	0.43	0.86
015	0.86	0.86	0.86	0.86	0.86	1.00	0.29	1.00	0.57	1.00
016	0.29	0.57	0.57	0.86	0.71	0.71	0.29	1.00	0.14	0.57
017	0.29	0.71	0.57	0.71	0.29	0.43	0.71	1.00	0.43	0.43
018	0.57	0.29	0.29	1.00	0.86	0.86	1.00	0.86	0.86	0.00
019	0.57	0.71	0.71	0.86	0.86	0.57	0.43	0.71	0.43	0.86
020	0.00	0.17	0.33	0.50	0.17	0.33	0.33	0.83	0.57	0.00
021	0.43	0.57	0.43	1.00	0.71	0.71	0.86	0.86	0.71	0.43
022	0.86	0.86	0.86	1.00	1.00	0.71	0.57	0.71	0.29	0.86
023	0.29	0.43	0.00	0.43	0.14	0.29	0.43	0.71	0.14	0.71
024	0.43	0.29	0.43	0.86	0.29	0.14	0.43	1.00	0.29	0.71

025	0.14	0.57	0.43	0.43	0.43	0.43	0.14	1.00	0.57	0.43
026	0.57	0.71	0.43	0.86	1.00	0.71	0.29	0.86	0.14	0.71
027	0.71	0.29	0.29	0.29	0.71	0.71	0.14	0.86	0.43	0.86
028	0.43	0.14	0.14	0.43	0.14	0.57	0.57	0.71	0.29	0.43
029	0.57	0.71	0.57	1.00	0.43	0.71	0.86	1.00	0.57	1.00
030	0.29	0.43	0.57	0.86	0.71	0.57	0.57	0.71	0.71	0.57
031	0.29	0.14	0.57	0.43	1.00	1.00	0.14	0.86	0.29	1.00
032	0.29	0.57	0.43	0.29	0.29	0.71	0.57	0.86	0.43	0.57
033	0.14	0.86	0.14	0.71	0.71	0.71	0.71	0.86	0.14	0.71
034	0.00	0.14	0.00	0.29	0.43	0.14	0.14	0.86	0.29	0.43
035	0.14	0.14	0.29	0.14	0.29	0.43	0.57	1.00	1.00	0.29
036	0.57	0.57	0.29	0.86	0.57	0.71	0.43	0.86	0.43	0.86
037	0.86	0.86	0.86	0.86	0.57	0.57	0.71	0.86	0.00	0.86
038	0.29	0.29	0.43	0.29	0.29	0.14	0.00	1.00	0.29	0.43
039	0.71	0.57	0.43	0.57	0.71	0.86	0.29	0.71	0.71	0.43
040	0.43	0.29	0.71	0.57	0.43	0.43	0.86	0.86	0.14	0.29
041	0.57	0.71	0.71	1.00	0.86	0.57	0.71	0.86	0.29	1.00
042	0.14	0.71	0.14	0.29	0.29	0.43	0.14	1.00	0.43	0.57
043	0.33	0.17	0.17	0.67	0.17	0.33	0.17	0.83	0.57	0.17
044	0.00	0.00	0.00	0.00	0.29	0.00	0.71	1.00	0.57	0.43
045	0.29	0.71	0.43	1.00	0.86	0.57	0.71	1.00	0.00	0.71
046	0.57	0.43	0.43	0.43	0.57	0.43	0.57	0.86	0.71	0.71
047	0.71	0.86	0.43	1.00	0.57	0.86	0.43	0.71	0.57	0.71
048	0.29	0.43	0.14	0.57	0.29	0.14	0.86	0.71	0.14	0.71
049	0.86	0.43	0.86	0.86	0.57	0.43	0.71	0.86	0.29	0.71
050	0.43	0.71	0.43	0.86	0.43	0.86	0.86	1.00	0.86	1.00
051	0.86	0.57	0.71	0.57	0.57	0.71	0.43	0.71	1.00	0.71
052	0.29	0.43	0.29	0.86	0.71	0.57	0.57	1.00	0.57	0.71
053	0.57	0.29	0.86	0.43	0.86	0.86	0.57	0.86	0.43	0.29
054	0.71	0.86	0.86	1.00	0.57	0.86	0.71	1.00	0.71	0.86
055	0.71	0.71	0.86	0.86	0.43	0.43	0.57	1.00	0.43	1.00
056	0.29	0.57	0.29	0.57	0.71	0.29	0.00	0.86	0.14	0.57
057	0.57	0.71	0.71	0.86	0.86	0.71	0.57	0.86	0.43	1.00
058	1.00	1.00	0.86	1.00	1.00	1.00	0.29	0.86	0.00	1.00
059	0.43	0.57	0.57	0.71	0.43	0.43	0.71	1.00	0.29	0.57

060	0.86	0.71	0.71	0.86	1.00	1.00	0.86	1.00	0.00	1.00
061	0.14	0.29	0.14	0.00	0.00	0.29	0.14	0.86	0.43	0.29
062	0.71	0.71	0.86	1.00	1.00	0.43	0.43	1.00	0.14	0.86
P	0.47	0.53	0.49	0.69	0.59	0.58	0.50	0.88	0.40	0.64

Tabla D.1: Desempeño Top-1 por clase, primera ronda por órgano, flores.

CLS	MB2	INRN	VG19	IN3	D201	XC	D121	D169	VG16	RN50
001	0.71	0.86	1.00	1.00	0.57	1.00	0.29	0.86	0.71	0.57
002	0.71	0.86	1.00	0.71	0.86	0.29	0.71	0.86	0.71	0.57
003	0.57	0.86	1.00	0.29	0.71	0.86	0.86	0.71	0.86	0.71
004	0.86	0.86	0.71	0.71	0.71	0.86	0.57	0.86	1.00	0.57
005	1.00	1.00	1.00	1.00	0.86	0.71	0.57	0.71	1.00	0.29
006	1.00	0.86	0.71	0.86	0.86	0.86	0.71	0.57	1.00	0.86
007	0.86	0.86	0.86	0.57	0.86	0.29	0.71	0.71	1.00	0.29
P	0.82	0.88	0.90	0.73	0.78	0.69	0.63	0.76	0.90	0.55

Tabla D.2: Desempeño Top-1 por clase, primera ronda por órgano, frutos.

CLS	D169	IN3	INRN	MB2	D201	VG16	VG19	RN50	D121	XC
001	0.29	0.43	0.57	0.43	0.29	0.71	0.71	0.00	0.14	0.43
002	0.43	0.43	0.29	0.43	0.29	0.86	1.00	0.29	0.43	0.71
003	0.14	0.14	0.43	0.29	0.14	0.71	1.00	0.00	0.29	0.43
004	0.57	0.71	0.71	0.43	0.71	1.00	0.86	0.29	1.00	0.29
005	0.00	0.86	0.29	0.29	0.29	0.71	0.43	0.14	0.00	0.29
006	0.86	0.57	0.57	0.71	0.71	0.71	0.86	0.14	0.43	0.71
007	0.57	0.57	0.57	0.43	0.14	0.86	0.86	0.71	0.29	0.71
008	0.14	0.43	0.86	0.86	0.43	1.00	1.00	0.14	0.43	0.71
009	0.67	0.33	0.67	0.50	0.33	0.83	0.67	0.00	0.50	0.33
010	0.29	0.29	0.43	0.43	0.29	0.86	0.57	0.14	0.43	0.57
011	0.14	0.14	0.71	0.57	0.14	1.00	1.00	0.14	0.14	0.57
012	0.29	0.14	0.43	0.43	0.14	1.00	0.86	0.29	0.71	0.43
013	0.14	0.57	0.29	0.43	0.14	0.86	0.71	0.14	0.14	0.29
014	0.14	0.43	0.86	0.00	0.43	0.86	1.00	0.14	0.57	0.57
015	0.29	0.14	0.43	0.29	0.43	0.71	0.86	0.29	0.43	0.57
016	0.29	0.57	0.86	0.29	0.43	0.43	0.86	0.43	0.57	0.29

017	0.86	0.57	0.57	0.29	0.57	0.86	0.71	0.71	0.29	0.43
018	0.43	0.57	0.57	0.29	0.43	0.86	0.71	0.71	0.43	0.57
019	0.14	0.14	0.29	0.43	0.14	0.57	0.86	0.43	0.29	0.57
020	0.29	0.00	0.43	0.00	0.43	0.71	0.71	0.14	0.43	0.00
021	0.57	0.14	0.86	0.71	0.57	0.57	0.86	0.29	0.43	0.71
022	0.14	0.57	0.43	0.14	0.43	1.00	0.71	0.00	0.14	0.14
023	0.00	0.14	0.57	0.29	0.43	0.71	0.57	0.14	0.14	0.29
024	0.29	0.57	0.86	0.71	0.71	0.86	0.71	1.00	0.00	0.71
025	0.57	0.43	0.71	0.29	0.86	0.71	0.71	0.57	0.86	0.43
026	0.43	0.14	0.71	0.43	0.29	0.86	0.86	0.14	0.71	0.71
027	0.43	0.43	0.71	0.29	0.57	0.86	0.86	0.57	0.14	0.43
028	0.57	0.43	0.71	0.29	0.29	1.00	0.86	0.29	0.71	0.43
029	0.43	0.43	0.86	0.43	0.57	0.57	0.86	0.43	0.43	0.57
030	0.43	0.14	0.29	0.29	0.14	1.00	0.86	0.00	0.00	0.43
031	0.43	0.14	0.71	0.43	0.43	1.00	1.00	0.00	0.57	0.43
032	0.43	0.29	0.86	0.57	0.43	0.57	1.00	0.71	0.29	0.29
033	0.43	0.29	0.86	0.14	0.29	0.86	0.86	0.00	0.43	0.43
034	0.14	0.43	0.86	0.57	0.14	0.86	0.71	0.29	0.43	0.29
035	0.29	0.43	0.43	0.29	0.57	0.86	0.71	0.29	0.29	0.43
036	0.14	0.00	0.43	0.29	0.14	0.86	0.71	0.00	0.14	0.43
037	0.57	0.43	0.86	0.57	0.57	1.00	1.00	0.00	0.14	0.29
038	0.14	0.43	0.57	0.43	0.14	0.86	0.71	0.29	0.14	0.29
P	0.35	0.37	0.61	0.39	0.38	0.82	0.81	0.27	0.37	0.45

Tabla D.3: Desempeño Top-1 por clase, primera ronda por órgano, hojas.

CLS	D201	D121	D169	INRN	VG16	MB2	VG19	IN3	RN50	XC
001	0.71	0.86	0.71	0.86	0.86	1.00	0.86	0.86	0.86	0.71
002	0.71	0.86	0.71	1.00	0.86	1.00	1.00	0.71	1.00	1.00
003	0.86	0.71	0.86	1.00	1.00	0.86	1.00	0.86	0.86	0.86
004	0.86	0.86	1.00	1.00	1.00	0.86	0.86	0.86	1.00	1.00
005	0.75	1.00	0.88	1.00	1.00	1.00	1.00	0.88	0.88	1.00
P	0.78	0.86	0.83	0.97	0.94	0.94	0.94	0.83	0.92	0.92

Tabla D.4: Desempeño Top-1 por clase, primera ronda por órgano, tallos.