UNIVERSIDAD NACIONAL DEL ALTIPLANO

ESCUELA DE POSGRADO

MAESTRÍA EN INGENIERÍA DE SISTEMAS

TESIS

ESTUDIO COMPARATIVO DE REDES NEURONALES CONVOLUCIONALES PARA LA CLASIFICACIÓN DE ESPECIES FORESTALES MADERABLES EN LA AMAZONÍA PERUANA

PRESENTADA POR:

DANITZA YVETTE BERMEJO ESCOBAR

PARA OPTAR EL GRADO ACADÉMICO DE:

MAESTRO EN INGENIERÍA DE SISTEMAS

PUNO, PERÚ

2021

DEDICATORIA

A Dios, a mis padres Hugo y Mary, con todo mi cariño a mis hermanos Brandon y Karen, y a todos quienes me apoyaron durante este tiempo.

Danitza.

AGRADECIMIENTOS

En primer lugar deseo agradecer a Dios por darnos la dicha de recibir un nuevo día y permitirnos cuidar a los que amamos.

Agradezco la Escuela de Posgrado de la Universidad Nacional del Altiplano por acogerme en sus aulas y permitirme este crecimiento personal.

De forma muy especial agradezco a mi asesora Ph. D. Guina Sotomayor Alzamora por ser inspiración para todo estudiante y por su asistencia incondicional durante todo el desarrollo de la tesis.

A mi compañero y mentor M.Sc. Gerson Vizcarra por su paciencia e incomparable soporte en cada nuevo reto.

Agradezco al Instituto de Investigaciones de la Amazonía peruana (IIAP) por darme la oportunidad de formar parte de este proyecto.

Al M.Sc. Erwin Junger Dianderas por ser guía y respaldo del equipo del trabajo.

A los ingenieros Kevin Pinedo y Anthony Perez por brindarnos su conocimiento y hacernos sentir como en casa.

Finalmente, quisiera agradecer al Fondo Nacional de Desarrollo Científico, Tecnológico y de Innovación Tecnológica (FONDECYT) iniciativa del Consejo Nacional de Ciencia, Tecnología e Innovación Tecnologíca (CONCYTEC) que financió el presente trabajo bajo el contrato 022-2018-FONDECYT-BM-IADT-AV.

ÍNDICE GENERAL

Pág.

CAPÍTULO II

PLANTEAMIENTO DEL PROBLEMA

CAPÍTULO III

MATERIALES Y MÉTODOS

CAPÍTULO IV

RESULTADOS Y DISCUSIÓN

Puno, 16 de noviembre de 2021

ÁREA: Ciencias de la ingeniería.
TEMA: Inteligencia artificial.
LÍNEA: Sistemas, computación e informática.

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Pág.

ÍNDICE DE ANEXOS

ÍNDICE DE ACRÓNIMOS

CNN Red Neuronal Convolucional(Convolutional Neural Network)

NN Red Neuronal(Neural Network)

ReLU Unidad Lineal Rectificador(Rectified Linear Units)

PReLU Unidad Lineal Rectificador Paramétrica(Parametric Rectified Linear Unit)

ILSVRC Competencia de reconocimiento visual a gran escala de ImageNet (ImageNet Large Scale Visual Recognition Challenge)

NIN Ret en una Ret (Network In Network)

SUNAT Superintendencia Nacional de Aduanas y de Administración Tributaria

SERFOR Servicio Nacional Forestal y de Fauna Silvestre

IIAP Instituto de Investigaciones de la Amazonía Peruana

UTM Universal Transversal de Mercator(Universal Transverse Mercator)

SGD Descenso Gradiente Estocástico(Stochastic Gradient Descent)

RESUMEN

Actualmente, el uso de técnicas de aprendizaje profundo en la clasificación de especies se ha convertido en un área atractivo de investigación. Asimismo, la clasificación de especies forestales maderables en el Perú fue determinada como un problema crítico-prioritario por las autoridades forestales, debido al impacto negativo que lleva una incorrecta clasificación sobre los intereses nacionales. La presente investigación se centró en la comparación de los modelos de aprendizaje por transferencia: AlexNet, VGG-19, Inception V3 y ResNet-101, redes neuronales convolucionales pre-entrenadas para la clasificación de especies. Para ello, se presentó el conjunto de datos denominado Peruvian Amazon Forestry Dataset, conformado por 59,441 imágenes pertenecientes a 10 especies forestales maderables de la Amazonia peruana en peligro de extinción y con importancia económica. Para los experimentos se usaron dos tipos de datos de entrada (imágenes segmentadas e imágenes no segmentadas). Los modelos fueron evaluados cuantitativa (exactitud, precisión, exhaustividad y valor-F1) y cualitativamente (interpretabilidad visual). Los resultados presentan valores significativos para la red VGG-19 al utilizar entradas no segmentadas, alcanzando un 97.61 % de exactitud en el entrenamiento, 98.87 % en la validación, y una exactitud del 97.02 % en el test. Además, gracias a la interpretabilidad, se observó que VGG-19 clasifica a partir de la forma y la vena de la hoja. Por último, es posible concluir que el modelo VGG-19 es una herramienta útil para que especialistas y *materos* para clasificar especies forestales maderables en tiempo real.

Palabras Clave: Amazonia peruana, clasificación de imágenes, especies forestales maderables, redes neuronales convolucionales

ABSTRACT

Currently, the use of deep learning techniques in species classification has become an attractive area of research. Likewise, the classification of timber forest species in Peru was identified as a critical priority problem by forestry authorities, due to the negative impact of incorrect classification on national interests. The current research focused on the comparison of transfer learning models: AlexNet, VGG-19, Inception V3 and ResNet-101, pre-trained convolutional neural networks for species classification. For this purpose, the Peruvian amazon forestry dataset, consisting of 59,441 images belonging to 10 endangered and economically important timber forest species of the Peruvian Amazon, was presented. Two types of input data (segmented images and non-segmented images) were used for the experiments. The models were evaluated quantitatively (accuracy, precision, recall, and F1-value) and qualitatively (visual interpretability). The results present significant values for the VGG-19 network when using non-segmented inputs, reaching 97.61 % accuracy in training, 98.87 % accuracy in validation, and 97.02 % accuracy in testing. Moreover, thanks to the interpretability, it was observed that VGG-19 classifies from leaf shape and vein. Finally, it is possible to conclude that the VGG-19 model is a useful tool for specialists and *materos* to classify timber forest species in real time.

Keywords: Convolutional neural networks, image classification, Peruvian amazon, timber forest species.

INTRODUCCIÓN

La flora de la Amazonía cubre el 21 % de la superficie terrestre (Keenan *et al*., 2015) y posee una gran biodiversidad con más de 15,000 especies forestales (Ter *et al*., 2020). La diversidad de especies forestales juega un papel muy importante en diversas áreas como la protección del medio ambiente, la industria maderera, y en las actividades cotidianas y productivas de los seres humanos, ya que mantienen el equilibrio entre el dióxido de carbono y oxigeno de la atmósfera, ayudan a satisfacer algunas necesidades humanas y a su vez contribuyen en el flujo económico (ITP/CITEmadera, 2018). No obstante, el incorrecto manejo de los recursos forestales conlleva a consecuencias adversas como el cambio climático (Valqui *et al*., 2016).

En el Perú, desde el 2012, a fin de regular el aprovechamiento sostenible de los recursos forestales maderables se establecieron leyes y normativas que implican procedimientos y métodos para una correcta clasificación de las especies forestales maderables. Sin embargo, en un escenario típico, la clasificación de las especies forestales maderables es llevada a cabo por un *matero*, quien, clasifica las especies una vez talado el árbol a través del fuste. Además, la clasificación suele darse a nivel de nombre común, lo que lleva a una posible sobreexplotación de especies no correspondientes a su taxonomía y/o al encubrimiento de lavado de activos (ITP/CITEmadera, 2018), puesto que una o más especies poseen el mismo nombre común.

Por ejemplo, más de 14 especies como la *Virola pavonis, Iryanthera laevis, Otoba glycycarpa, Virola calophylla y Otoba parvifolia* comparten el nombre común cumala (Zárate *et al*., 2012). Además, aún si existiese un experto in situ para realizar la clasificación y no se llegase a reconocer la especie por los métodos tradicionales se debe recurrir a una muestra por laboratorio, lo que convierte el proceso en una tarea complicada incluso para los expertos (Bonnet *et al*., 2018; Chulif *et al*., 2019).

Estudios recientes muestran que los algoritmos de aprendizaje automático pueden apoyar a personas no especializadas y expertas en la tarea de clasificación y reconocimiento de hojas de plantas (Azlah *et al*., 2019). Específicamente, las CNNs demostraron ser una alternativa muy atractiva, llegando a obtener una exactitud entre 91.78 % y 99.90 % (Elnemr, 2019; Saini *et al*., 2020; Sun *et al*., 2017). No obstante, son pocos los estudios orientados a la clasificación automatizada de especies forestales maderables de la Amazonía, donde

la flora posee propiedades peculiares debido a las características geológicas del terreno.

En el presente trabajo se aborda la tarea de clasificación de especies forestales maderables de la Amazonía peruana, utilizando modelos de aprendizaje profundo. Específicamente, se realiza una comparación de las CNNs AlexNet, VGG-19, Inception y ResNet-101, para lo cual, se construyó un *dataset* con 58,114 imágenes de hojas de 10 especies forestales maderables de la Amazonía peruana, con esta información se buscó una solución algorítmica para desarrollar en futuro una herramienta móvil capaz de apoyar a usuarios expertos o no de manera acertada en la clasificación de especies forestales maderables.

El trabajo de investigación está organizado de la siguiente manera: en el capítulo I, se detallan los conceptos que sustentan el trabajo, y se presentan los antecedentes relacionados al estado del arte; en el capítulo II, se desarrollan el problema, objetivos e hipótesis; en el capítulo III, se describen el diseño de la investigación, materiales y métricas utilizadas, así como detalles sobre el *dataset* compuesto; en el capítulo IV, se presentan los resultados organizados en dos grupos: cuantitativos y cualitativos representados en tablas y figuras. Finalmente, se hace mención a las conclusiones y recomendaciones a considerar para futuras investigaciones.

CAPÍTULO I

REVISIÓN DE LITERATURA

En este capítulo se desarrolla la revisión literaria respecto a la tarea de clasificación de especies forestales como tema de interés conforme se han ido desarrollando los algoritmos de aprendizaje profundo. La automatización de este proceso resulta esencial en la mayoría de las investigaciones realizadas anteriormente, donde los resultados obtenidos se ven afectados directamente por el tipo de algoritmo de aprendizaje profundo utilizado y por la especie forestal.

1.1 Marco teórico

1.1.1 Especies forestales maderables

Se refiere por **especie** al grupo de organismos que comparten una cualidad determinada, así, las especies forestales maderables comprenden la vegetación leñosa que en su mayoría son árboles. A nivel mundial, la Amazonía es conocida como fuente de biodiversidad (Neves, 2019) ya que alberga más de 15,000 especies forestales maderables (Ter *et al*., 2020).

Conocer la diversidad de las especies forestales maderables es de vital importancia ya que los productos obtenidos a lo largo de la cadena forestal productiva varían de una especie a otra. En el Perú, a fin de lograr el manejo sostenible de este recurso, existen normas vigentes que buscan el uso de un nombre único para una especie durante toda la cadena de producción forestal. De esta manera se busca proteger especies en peligro de extinción y la uniformidad de calidad de los productos maderables. Específicamente, una de estas normas trata sobre la trazabilidad de los recursos forestales

maderables cuyo objetivo es localizar a un individuo en cualquier etapa de la cadena forestal productiva.

Trazabilidad de los recursos forestales maderables

La trazabilidad es aquel proceso que busca verificar el origen legal de los productos y subproductos forestales maderables desde su extracción de los bosques naturales hasta su comercialización en los mercados nacionales e internacionales. Según la Resolución de Dirección Ejecutiva Nro. 230-2019-MINAGRI-SEFOR-DE 2019, el fin de la trazabilidad es promover el sector forestal maderable y elevar su competitividad de manera sostenible. Por lo tanto, la trazabilidad de los productos maderables permite la regulación de los recursos forestales durante la cadena productiva forestal.

En el Perú, el Servicio Nacional Forestal y de Fauna Silvestre (SERFOR) está encargado de verificar el origen legal de los especímenes y realizar el rastreo histórico de los productos forestales maderables a través del uso de diversos instrumentos y materiales basados en el registro de información por parte de los usuarios y autoridades forestales (MINAGRI, 2014). Algunos de estos instrumentos son: plan de manejo forestal, libro de operaciones, guía de transporte forestal, y aplicativos informáticos de registro de información. Su uso es acorde a la cadena forestal productiva, la cual, comienza por su planificación y culmina con la comercialización de los productos maderables.

Cadena Productiva

La cadena forestal productiva consta de siete etapas como se puede apreciar en la Figura 1 donde la **planificación** agrupa los procesos de evaluación (selección y ubicación) presentados dentro de un plan de manejo forestal el cual debe ser previamente aprobado para dar inicio al proceso. Dicho esto, el plan de manejo consiste en la información censo forestal del árbol como: ubicación, código y la **identificación/clasificación de la especie**.

La siguiente etapa consiste en el **aprovechamiento forestal** consiste en las actividades de tala, trozado de la madera y su salida. El **transporte primario** es la

movilización de los productos del centro de extracción hacia su industrialización. La **transformación primaria** implica el procesamiento de los productos al estado natural. La siguiente etapa consiste en el **transporte secundario**, de los centros de transformación hacia los depósitos o centros de comercialización. La **transformación secundaria** puede ser realizados en centros de transformación primaria o secundaria. Finalmente, la **comercialización** se realiza durante toda duración de toda la cadena productiva, orientado tanto al mercado interno como al externo.

Figura 1. Cadena forestal productiva

Adaptado de Resolución de Dirección Ejecutiva Nro. 230-2019-MINAGRI-SEFOR-DE, 2019

Clasificación tradicional especies forestales

Según la Figura 1, la primera etapa de la cadena productiva forestal es la planificación que consiste en la elaboración, presentación y aprobación del plan de manejo donde intervienen diversos actores como las autoridades forestales, profesionales y la comunidad indígena. El fin principal del plan de manejo es unificar la infor-

mación censo forestal del árbol durante toda la cadena de producción forestal, la cual consiste en datos relacionados a:

- **Ubicación del árbol**, especificado en coordenadas Universal Transversal de Mercator (Universal Transverse Mercator, UTM), este dato permite controlar la extracción responsable del árbol sin afectar en la deforestación extrema y reconocer el tipo de hábitat común en la especie.
- **Asignación de un código**, etiqueta al árbol con una identificación única
- **Identificación de especie**, el cual permanece durante toda la cadena productiva forestal, una correcta identificación o clasificación de la especie forestal permite la protección de especies en peligro de extinción y la uniformidad de calidad en los productos maderables.

Los datos recolectados para la elaboración del plan de manejo, según las normas peruanas, deben ser recopilados antes de realizar la tala de los árboles y es el profesional a cargo de la inspección el responsable de realizar la identificación y clasificación de la especie. Para este proceso el profesional puede apoyarse de un *matero* (personal) y proceder con el reconocimiento de la especie, el cual se da de dos formas:

- Verificación de las características con el apoyo de un manual dendrológico. Un manual dendrológico, por lo general, contiene información sobre la entidad de la especie, frutos, muestra seca, mapa de distribución, fuste, raíz, hojas, flor y semilla (OSINFOR, 2018).
- De sospechar de una incorrecta verificación, se recolecta una muestra (tres ejemplares de hojas, fruto y/o flor) y se prosigue con el procedimiento tradicional. La Figura 2 muestra el método tradicional que inicia con la ubicación del espécimen y finalizando con el etiquetado del espécimen.

Especies forestales en el Perú

La Amazonia posee una de las mayores diversidades en especies forestales a nivel mundial y su flora cubre el 21 % de la superficie terrestre (Keenan *et al.*, 2015;

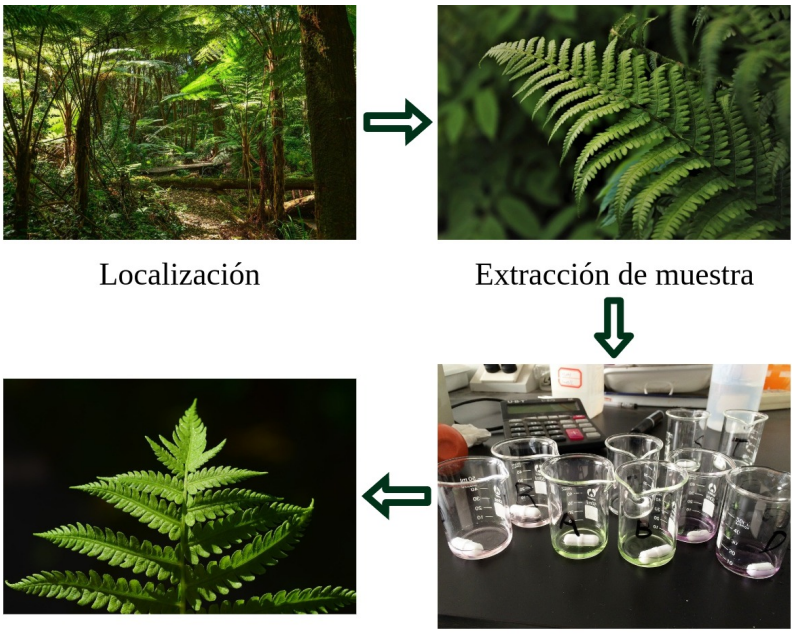


Figura 2. Proceso de la clasificación de hojas por método tradicional

Wittmann *et al*., 2006), pero debido a las propiedades del terreno, a la fecha, se desconoce el total exacto de especies forestales maderables que la habitan. Sin embargo, se estima que podrían existir más de 15,000 especies de árboles en toda la Amazonia (Ter *et al*., 2020), de los cuales según Farfan-Rios *et al*. 6,350 podrían estar en territorio peruano. Además, en el Perú, se tiene una lista conformada por 275 especies forestales maderables reconocidas y estandarizadas.

De acuerdo con la lista existente de las 275 especies forestales maderables a continuación se describe brevemente sobre las 10 especies forestales maderables utilizadas en el presente trabajo de investigación (Bendezú, 2016; OSINFOR, s.f.; «Resolución de Dirección Ejecutiva Nro.118-2019-MINAGRI-SEFOR-DE», 2019). Para lo cual, se presenta cada especie por su nombre científico, seguidamente se señala: reino, clase, familia, nombre común, nombre comercial, uso, hábitat, y finalmente una breve descripción sobre las hojas (*L* y *W* presentan el largo y ancho promedio en centímetros respectivamente).

1. ***Aniba rosaeodora ducke***

   Reino: Plantae

   Clase: Magnoliopsida

Familia: Lauraceae

Nombre común: Moena amarilla,palo de rosa, palo rosa.

Nombre comercial: Moena amarilla

Uso: Construcción en general, semilla comestible.

Hábitat: Bosques amazónicos primarios suelo arcilloso o arcilloso-arenoso.

Hojas: Compuestas alternas, pariráceas y subcoriáceas, oblongas o elípticas, haz glabra, envés piloso, $L = [9;20] \wedge W = [3;6]$.

2. ***Cedrela odorata***

   Reino: Plantae

   Clase: Magnoliopsida

   Familia: Meliaceae

   Nombre común: Cedro, cedro blanco, cedro colorado, cedro de altura, cedro del bajo, cedro de restinga, cedro rojo, kanu, seetrú

   Nombre comercial: Cedro

   Uso: Carpintería, ebanistería fina.

   Hábitat: Bosques primarios de tierra firme. Amazonía peruana, departamentos de: Loreto, Amazonas, Junín, Cerro de Pasco, Madre de Dios y Puno.

   Hojas: Compuestas alternas, simétricas, $L = [9;12] \wedge W = [3{,}5;5]$.

3. ***Cedrelinga cateniformis ducke***

   Reino: Plantae

   Clase: Magnoliopsida

   Familia: Fabaceae

   Nombre común: Achapo blanco, achuapo, aguano, aguano masha, cedro macho, cedro masha, cedro rana, huaira caspi, huayra caspi, tsaik.

   Nombre comercial: Tornillo

   Uso: Carpintería, carrocería, muebles, juguetes, estructura de viviendas.

   Hábitat: Bosques de terraza alta, bosques montañas, bosques de colina baja. Amazonía peruana, departamentos de: Loreto, Amazonas, Madre de Dios.

   Hojas: Compuestas alternas, dispuestas en espiral, longitudinalmente estriado, con una glándula en su ápice, haz y envés pálido, $L = [4;15] \wedge W = [2;9]$.

4. ***Dipteryx micrantha***

   Reino: Plantae

Clase: Magnoliopsida

Familia: Fabaceae

Nombre común: Cumarú, kumarut, shihuahuaco hoja pequeña

Nombre comercial: Shihuahuaco

Uso: Carpintería.

Hábitat: Países Brasil, Bolivia, Ecuador y Perú (Loreto).

Hojas: Compuestas alternas, pinnadas, alternas, lámina oblonga, glabra en ambas caras, coriáceo, borde entero, base redonda, $L = [6;8,4] \wedge W = [2,6;3,4]$.

5. ***Otoba glycycarpa*** **Reino:** Plantae

   Clase: Magnoliopsida

   Familia: Myristicaceae

   Nombre común: Aguanillo, banderilla, caobilla, cumala, cumala colorada.

   Nombre comercial: Aguanillo

   Uso: Construcción de canoas, madera, leña y colorante de piel.

   Hábitat: Bosque de colina, bosque aluvial.

   Hojas: Simple y alternas, haz glabro, $L = [12;20] \wedge W = [4;7]$.

6. ***Otoba parvifolia***

   Reino: Plantae

   Clase: Magnoliopsida

   Familia: Myristicaceae

   Nombre común: Aguanillo, banderilla, caobilla, cumala, cumala colorada, favorito, kumala, mamilla, pintana negra.

   Nombre comercial: Aguanillo

   Uso: Construcción de canoas, madera, leña, colorante de piel, resina para tratar hongos en la piel.

   Hábitat: Bosque aluvial.

   Hojas: Simple y alternas, elípticas, $L = [12,20] \wedge W = [4,7]$.

7. ***Simarouba amara***

   Reino: Plantae

   Clase: Magnoliopsida

   Familia: Simaroubaceae

   Nombre común: Hualaja, marupa

Nombre comercial: Marupa

Uso: Aceite comestible.

Hábitat: Bosques caducifolios en lugares abiertos.

Hojas: Alternas, imparipinnadas, oblongos u obovados, ápice emarginado, base aguda, $L = [10;30]$.

8. ***Swietenia macrophylla***

Reino: Plantae

Clase: Magnoliopsida

Familia: Meliaceae

Nombre común: Aguano, caoba, ébano, mara, mahogani, pasich

Nombre comercial: Caoba

Uso: Carpintería, muebles.

Hábitat: Sitios al nivel del mar hasta zonas de montaña.

Hojas: Alternas, paripinnadas o a veces imparipinnadas, láminas ovadas y asimétricas, ápice agudo y falcado, base aguda, $L = [9;13] \wedge W = [3;4]$.

9. ***Virola flexuosa***

Reino: Plantae

Clase: Magnoliopsida

Familia: Myristiceae

Nombre común: Caupiri de altura, cumala, cumala blanca, cumala negra.

Nombre comercial: Cumala

Uso: Medicina Tradicional.

Hábitat: Amazonía peruana, departamentos de: Amazonas, Huánuco, Loreto, Madre de Dios, Pasco, San Martín, Tumbes y Ucayali.

Hoja: Simples alternas, lámina elíptica, haz verde brillante, envés pubescente de color verde claro, $L = [20;25] \wedge W = [6;8]$.

10. ***Virola pavonis***

Reino: Plantae

Clase: Magnoliopsida

Familia: Myristiceae

Nombre común: Aguano cumala, caupiri del bajo, cumala, cumala amarilla, cumala blanca, cumala caupiri, caupiri del bajo, caupiri del varillal, cumala

zancuda.

Nombre comercial: Cumala

Uso: Medicina Tradicional.

Hábitat: Amazonía peruana, departamentos de: Amazonas, Huánuco, Loreto, Madre de Dios, Pasco, San Martín, Tumbes y Ucayali.

Hoja: Simples alternas, lámina elíptica, borde entero, haz verde oscuro y envés blanco con pelos dispersos, $L = [7;19] \wedge W = [2;4]$.

1.1.2 Conceptos complementarios

La siguiente subsección presenta conceptos previos necesarios respecto al aprendizaje profundo como la noción de tensor y el preprocesamiento de imágenes. Dichos conceptos, ayudarán proseguir con mayor fluidez la siguiente subsección donde se ampliará sobre el aprendizaje profundo y sus principales características.

Tensor

Con base en la teoría de vectores y matrices. Donde, un vector cualquiera $x \in R^D$ con D elementos y la matriz $\mathbf{X} \in R^{H \times W}$ con H filas y W columnas. El vector x puede ser, a su vez, una matriz de 1 columna y D filas y así en matrices de orden mayor. Un **tensor** de orden 3 denotado por $x \in R^{H \times W \times D}$, donde los elementos son H, W, D. Cada elemento puede ser indexado por (i, j, d) respectivamente, donde $0 \leq i < H, 0 \leq j < W \wedge 0 \leq d < D$. Asimismo, un tensor de orden 3 es pensar que este tiene D canales y cada canal tiene un tamaño $H \times W$. Que para que pueda ser operado un canal este contiene todos los números indexados (i, j, d) del tensor, así, cuando se toma el primer elemento $D = 1$ el tensor es reducido a una matriz.

Un tensor es conocido como un objeto matemático o entidad algebraica de varios componentes. La cantidad de componentes describe el orden del tensor. Así, un número escalar es un tensor de orden cero; un vector es un tensor de orden uno; y una matriz es un tensor de orden dos. Entonces un tensor de orden tres es una imagen a color. Una **imagen** con H filas y W columnas, es un **tensor** de tamaño $H \times W \times 3$. Si el color de una imagen está en formado por 3 canales RGB (rojo, verde y azul), a su vez, cada canal es una matriz $H \times W$ (tensor de orden 2)

contiene el valor de todos los píxeles que puede ser R, G o B.

El manejo de las imágenes como tensores y no como matrices permite conservar mucha más información. Los tensores de orden 3 o superior son usados para resolver tareas con las CNNs.

Preprocesamiento

El preprocesamiento de imágenes se aplica para normalizar los valores (características) de las imágenes antes de entrenar a una red. Lo que incluye dimensionar el tamaño, reducir las diferencias de iluminación, la reducción de ruido y otras conversiones que puedan afectar la clasificación de imágenes (Rizk, 2019). Recordando que una imagen está compuesta por píxeles y un píxel, normalmente, tiene 3 canales (RGB) con valor P entre 0 y 255. La normalización de las entradas en una red es importante para que cada parámetro de entrada tenga una distribución similar de datos. Por lo que, el proceso de entrenamiento es automatizado en tiempo. Por lo que, cada píxel debe ser operado para obtener un valor positivo entre 0 y 1. En otras palabras, $P \in [0,255], \frac{P}{255} = [0,1]$.

La imagen de la hoja

Para la botánica, la hoja es descrita como el órgano principal encargado de la fotosíntesis, respiración o la transpiración de la planta. Por lo general, la hoja es una estructura laminada compuesta por una parte plana llamada **lámina** o **limbo**, y por el **peciolo** encargado de unir la hoja con el tallo. El limbo presenta dos lados haz y envés, y ensanchamientos llamados **nervios** o **venas**. También, la hoja tiene un **margen** o **borde** que puede ser dentado, ondulado o liso. Por ejemplo, la Figura 3 presenta una hoja de la especie *Simarouba amara*.

La imagen de una hoja captada por herramientas tecnológicas proporciona información acerca del individuo al que pertenece. El método tradicional utiliza la imagen de la hoja como muestra para clasificar la especie del individuo ya que las características morfológicas poseen datos que ayudan a distinguir una especie de otra. Basándose en este conocimiento, la tecnología aborda la tarea de clasificación de especies utilizando la hoja quien resalta por su influencia en el desempeño

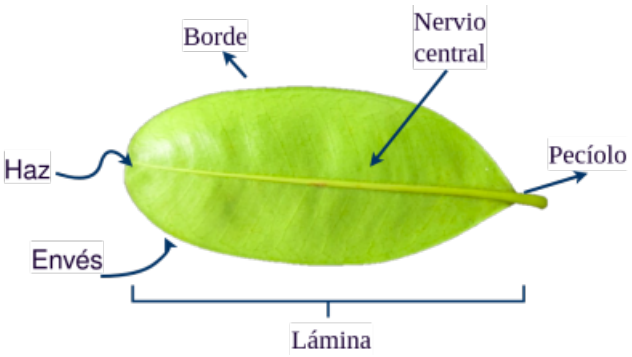


Figura 3. Imagen digital de hoja de la especie *Simarouba amara*

de los algoritmos (S. Zhang *et al*., 2020). Algunos trabajos se enfocan en características morfológicas específicas de la hoja como la textura (Ibrahim *et al*., 2018), la forma y contorno (Carranza-Rojas & Mata-Montero, 2016), y otros utilizan las venas (Tan *et al*., 2018).

Segmentación de la imagen de la hoja

La segmentación de la hoja consiste en extraer la hoja en sí de la imagen (Azlah *et al*., 2019). La Figura 4 muestra un ejemplo de segmentación de la imagen de una hoja de la especie *Simarouba amara* donde a la izquierda se muestra la imagen previa a la segmentación y a la derecha se tiene la imagen después de eliminar el fondo.

Figura 4. Izquierda imagen original, Derecha imagen segmentada

1.1.3 Aprendizaje profundo

El aprendizaje profundo (en inglés *Machine Learning*) es parte del aprendizaje automático que a su vez es una rama de la inteligencia artificial y permite que los

algoritmos *'aprendan'* de los datos de la entrada por medio de un entrenamiento (Bengio *et al*., 2017). Cuando un algoritmo aprende, es capaz de resolver una tarea en específica, en muchos casos, las tareas que debe realizar resultan complicadas para un ser humano (Bengio *et al*., 2017).

El aprendizaje puede ser supervisado, no supervisado y semi supervisado. Si el entrenamiento se lleva a cabo con datos etiquetados el algoritmo y está entrenado para una tarea en específica es **supervisado**. En caso de que los datos sean desconocidos se trata del aprendizaje **no supervisado**. Finalmente, el aprendizaje **semi supervisado** se encuentra entre estas dos categorías (Bengio *et al*., 2017). Generalmente, la tarea de clasificación de imágenes se realiza mediante el aprendizaje supervisado (O'Shea & Nash, 2015).

En los últimos años, el aprendizaje profundo ha resaltado por su capacidad de aprender utilizando redes neuronales artificiales logrando distintivos en diversos campos de aplicación de la inteligencia artificial, como el procesamiento de lenguaje natural (L. Deng & Liu, 2018; Otter *et al*., 2020) o en la visión artificial (Voulodimos *et al*., 2018).

1.1.4 Red neuronal

Una Red Neuronal (Neural Network, NN) está inspirada en el proceso biológico de las neuronas en el cerebro humano donde cada neurona es un elemento de procesamiento e interconexión. En 1943, McCulloch y Pitts crearon un modelo computacional basado en la lógica de umbral mediante una interpretación matemática el algoritmo que surgió dio pase a la aplicación de las NNs en la inteligencia artificial originando las **redes neuronales artificiales**. Una red neuronal artificial, al igual que en las NNs, las neuronas o nodos están interconectadas y procesan señales (Mehrotra *et al*., 1997). A su vez, al grupo de neuronas que reciben el mismo grupo de entradas se les denomina **capa**.

Las diferentes capas en una red neuronal transforman sus entradas enviando información desde la primera a la última capa. En otros casos, las señales también viajan en dirección inversa lo que se conoce como **Redes recurrentes**. Además, al conjunto de capas se le llama perceptrón multicapa y está conformado por tres tipos de capas:

La **capa de entrada** es aquella que recibe la entrada. La **capa de salida** da los valores de salida de la red. Las capas entre la capa de entrada y capa de salida se llaman **capas ocultas**. Para mayor complementar la Figura 5 ilustra el ejemplo del esquema de una **red neuronal artificial** y sus capas.

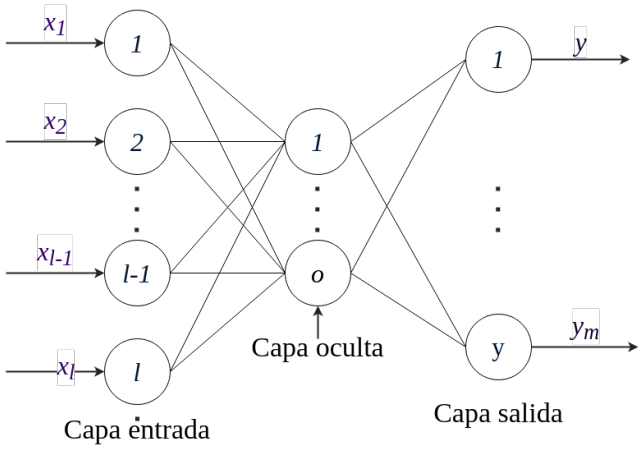


Figura 5. Esquema de una red neuronal artificial

Fuente: Adaptado de *Elements of artificial neural networks*, por Mehrotra *et al*., 1997.

Cuando el número de capas ocultas son más de dos se convierte en una **Red neuronal profunda**. Dichas redes tienen la capacidad de aprender a resolver problemas, lo que originó al **aprendizaje profundo** más conocido por su nombre en inglés *Deep learning*. Los algoritmos de aprendizaje y las redes artificiales profundas llevaron a un nuevo desafío ya que llevaba mucho tiempo entrenar las redes neuronales. Lo que llevo la oportunidad de experimentar con la **Red Neuronal Convolucional (Convolutional Neural Network, CNN)** las cuales destacan en este tipo de tarea.

1.1.5 Red Neuronal Convolucional

Una CNN es un tipo de red neuronal profunda que ha demostrado aportar beneficios positivos a la visión artificial (Rawat & Wang, 2017), algunas de las tareas que se pueden resolver haciendo uso de las CNNs son: reconocimiento de imágenes, clasificación de imágenes, detección de objetos, y entre otros. Su uso se popularizó con el uso de la red Alexnet propuesta por Krizhevsky *et al*. (2012). La Figura 6 muestra un ejemplo de una típica CNN. Donde, en contraste a técnicas tradicionales del aprendizaje profundo la idea es que cada capa debe extraer de forma autónoma las características de una imagen para poder discriminar otras clases.

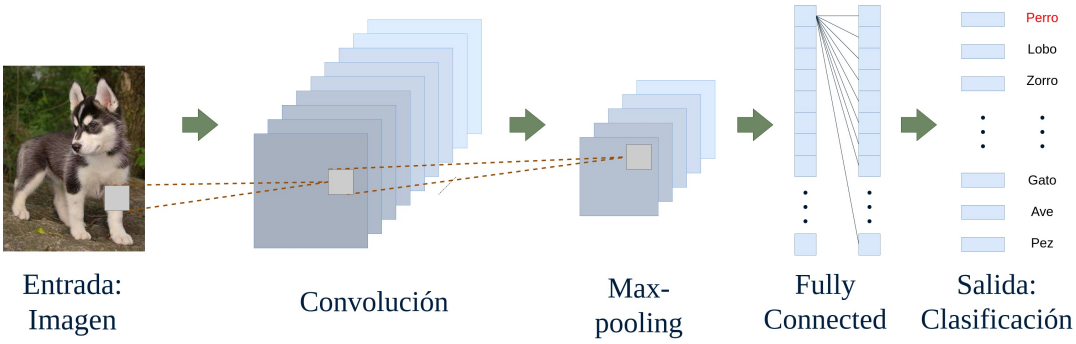


Figura 6. Ejemplo de una arquitectura típica de una CNN

El nombre de Convolucional viene de la operación matemática **convolución** aplicada en la entrada (Bengio *et al.*, 2017). Cuando se trata del procesamiento de imágenes se aplica como operador un ***kernel*** que se trata de una matriz pequeña cuyo fin es aplicar un filtro para extraer algunas características de la imagen que es un **tensor** de **orden 3** con filas H, columnas W y 3 canales (RGB). Kernel se desplaza sobre la matriz de entrada y aplica una multiplicación por pares de dos matrices y suma el resultado de la multiplicación finalmente como salida muestra la matriz resultante. La imagen 7 ilustra un ejemplo sobre como es la operación de una convolución, donde la imagen de entrada es 3×4 y la convolución *Kernel* es de 2×2.

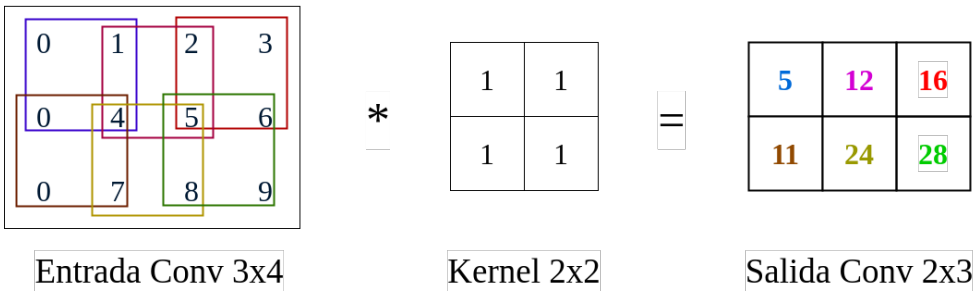


Figura 7. Ejemplo operación convolución (*Conv*)

En la imagen 7 el cálculo de la convolución en la región superior izquierda es: $1 \times 0 + 1 \times 0 + 1 \times 1 + 1 \times 4 = 5$, si se desplaza un píxel a la derecha la convolución es: $1 \times 2 + 1 \times 4 + 1 \times 2 + 1 \times 5 = 12$ y así sucesivamente. El número de desplazamientos de píxeles en una matriz de entrada se conoce como ***Stride***. Por ejemplo, si el *stride* es 1 (*uno*), el movimiento debe ser de un píxel a la vez, como se muestra en la Figura 7. El ***Padding*** es usado cuando el filtro no fue aplicado correctamente a la imagen de entrada, por lo que, se puede rellenar la imagen con 0 (ceros) o soltar parte de la imagen en la zona afectada, a fin de mantener solo una parte válida de la imagen.

Tipos de capas

La entrada forma una serie conocida como **capa**, su nombre varía según su función, por ejemplo, al tener una convolución se llama capa convolucional. A la fecha existen diversas variantes en la arquitectura de las CNNs pero, por lo general, una CNN consiste en varias capas convolucionales, capas de activación, capas de *pooling*, una serie de capas completamente conectadas, capa de pérdida, y entre otras. Para representar una arquitectura de CNN (J. Wu, 2017) se sugiere la siguiente secuencia:

$$x_1 \rightarrow w_1 \rightarrow x_2 \rightarrow \ldots \rightarrow x_i \rightarrow w_i \rightarrow \ldots \rightarrow x_j \rightarrow w_j \rightarrow y \tag{1}$$

- **Capa convolucional**

  La primera **entrada** x_1 generalmente es un tensor de orden 3 dado por $H \times W \times D$ (ancho $\times$ altura $\times$ profundidad) y x_i es la entrada de la capa i de tamaño $H_i \times W_i \times D_i$. En la primera capa, los parámetros involucrados son w_1 y la salida de la primera capa e ingreso de la segunda capa es x_2, el proceso finaliza con la capa x_j siendo y la salida. La capa **convolucional** extrae las características de cada x_i. Los parámetros de la capa consisten en el número de filtros que se pueden aprender (*kernel*). A su vez, cada filtro es influenciado por la H_j, W_j y D_j de x_j que calcula el producto de las entradas del filtro y la entrada produciendo un mapa de activación bidimensional de ese filtro. En consecuencia, la red aprende de filtros que se activan cuando se detecta algún objeto en el espacio de la entrada. Por ejemplo, si en la Figura 6 aparecen varios perros en una imagen de entrada el patrón de cabeza de perro se activarán en distintos espacios y/o ubicaciones.

  La **función de activación** $f(.)$ suele ser una función no lineal que afecta directamente en el entrenamiento de una red. Los modelos utilizados en el presente trabajo utilizan la función de Unidad Lineal Rectificador (Rectified Linear Units, ReLU) y Unidad Lineal Rectificador Paramétrica (Parametric Rectified Linear Unit, PReLU) los cuales demostraron presentar un mejor rendimiento (Simonyan & Zisserman, 2014). Por ejemplo, la Imagen 8 muestra una comparación en el tiempo de entrenamiento de ReLU (línea

roja) y a la función tradicional tangente hiperbólica (línea azul).

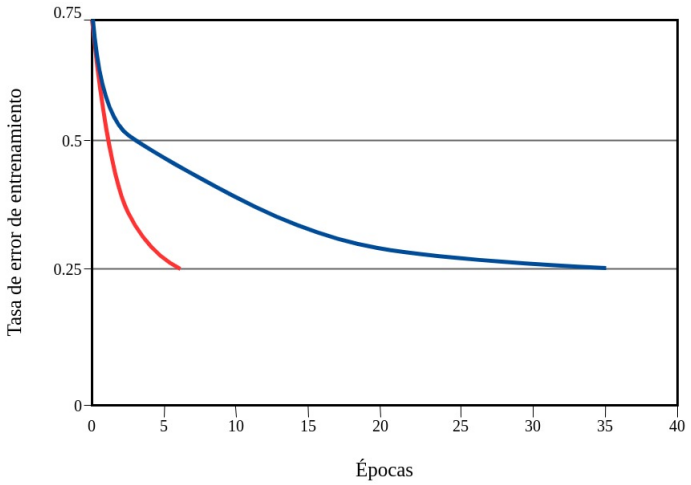


Figura 8. Tiempo de entrenamiento de las funciones ReLU y Tangente hiperbólica.

Adaptado *Imagenet classification with deep convolutional neural networks*, por Krizhevsky *et al*., 2012

- **ReLU**

  La función Unidad Lineal Rectificador (Rectified Linear Units, ReLU) introducida por Krizhevsky *et al*. (2012), está definida matemáticamente como:

$$f(x_i) = \text{máx}(0, x_i) \tag{2}$$

  ReLU toma a x_i como entrada a la capa si $x_i < 0, R(x_i) = 0 \wedge x_i > 0, R(x_i) = x_i$.

- **PReLU**

  Unidad Lineal Rectificador Paramétrica (Parametric Rectified Linear Unit, PReLU), fue propuesto por K. He *et al*. (2015) a fin de aprender los parámetros de los rectificadores durante un *backpropagation* resultado mejora la precisión. Matemáticamente PReLU está dado por:

$$f(x_i) = \text{máx}(0, x_i) + a_i \,\text{mín}(0, x_i) \tag{3}$$

  Donde x_i es la entrada de la función no lineal en un canal i y a_i un

parámetro en el rango de $[0,1]$ que controla la pendiente en la parte negativa. Si el valor de $a_i = 0$ entonces la función PReLU se convierte en una función ReLU. La Figura 9 muestra las representaciones gráficas de ReLU y PReLU cuando el coeficiente no es negativo.

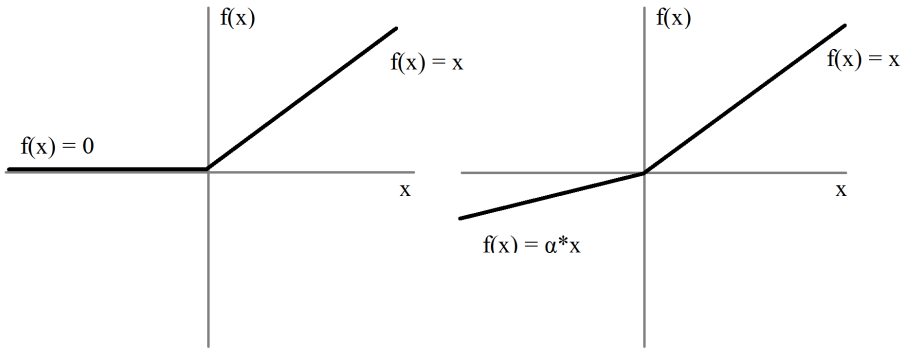


Figura 9. Funciones ReLU y PReLU

Fuente: Tomado de *Delving deep into rectifiers: Surpassing human-level performance on imagenet classification*, por K. He *et al.*, 2015

- **Capa pooling**

  La capa ***pooling*** es utilizada para reducir los parámetros en entradas grandes. Una entrada, contiene elementos en una matriz de $H \times W$ y puede ser subdividida en subregiones no superpuestas $H_i \times W_i$. Entre los diversos tipos de *pooling* tenemos a *average pooling* que se trata del agrupamiento en promedio de los elementos, *sum pooling* es la agrupación en suma de todos los elementos y ***max pooling*** toma el elemento mayor de una subregión. El fin de usar ReLU y la operación pooling es minimizar el coste computacional.

- **Capa completamente conectada**

  La **capa completamente conectada** en inglés *fully connected layer* es un perceptrón multicapa. Donde, en cada neurona está interconectada a las otras. La capa completamente conectada se puede utilizar al final de una CNN pues se refiere al cálculo de cualquier elemento en la salida y o x_{j+1} cuando requiere todos los elementos de la entrada x_j para obtener características con mayor peso. Debido a que la salida contiene representaciones distribuidas de la entrada original. Las reglas de aprendizaje de una capa completamente conectada son similares al de una CNN.

En una CNN se suele agregar una capa adicional para la propagación de errores hacia atrás, este método permite a la CNN aprender buenos valores de parámetros. Una estrategia común es aplicar una transformación **Softmax** en la capa x_{j-1}. Así, la última capa x_j es una capa de pérdida, en tareas de clasificación se suele utilizar la **entropía cruzada** cuyo fin es cuantificar la diferencia entre dos probabilidades.

Entrenamiento

El aprendizaje profundo permite a las CNNs aprender información abstracta. Sin embargo, un gran número de parámetros conduce al problema de sobreajuste en inglés *overfitting*, el cual es la consecuencia de sobreentrenar un algoritmo donde la función se ajusta a un conjunto de datos limitados. Este error puede ser medido por el rendimiento del modelo ante un nuevo *dataset* o al analizar las curvas de aprendizaje, la Figura 10 presenta un ejemplo hipotético de un modelo con *overfitting*.

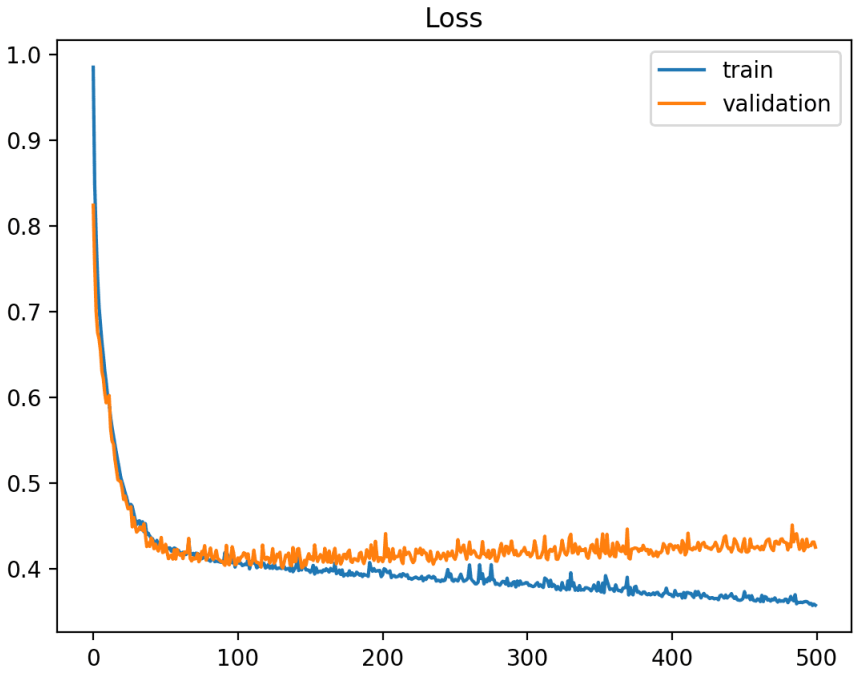


Figura 10. Ejemplo de *overfitting* en las curvas de entrenamiento y validación.
Adaptado de *Example of Train and Validation Learning Curves Showing and Overfit Model* por Brownlee, 2018, Machine Learning Mastery.

Del gráfico, se dice que presenta sobreajuste ya que la curva de pérdida del entrenamiento continúa disminuyendo, mientras que la curva de validación disminuye hasta cierto punto y luego comienza a aumentar, esta dinámica es muestra de *overfitting*. Ante este problema emergieron diversos métodos de regularización para controlar la aparición del *overfitting*. Algunas técnicas de regularización, son:

- **Dropout**

  EL concepto de 'abandono' en inglés *dropout* fue propuesto por Hinton *et al*. (2012) y ampliado por Srivastava *et al*. (2014). Donde en cada etapa de entrenamiento el algoritmo aleatoriamente abandona a la mitad de las características detectadas a fin de mejorar la capacidad de generalización y reducir la adaptación del aprendizaje. El abandono se da cuando cierto porcentanje de unidades de salida son valorados como cero. Por ejemplo, en la red VGG el porcentaje del abandono es de 0.5 (Simonyan & Zisserman, 2014).

- **Pre-entrenamiento**

  Cuando se utiliza el pre-entrenamiento las redes comienzan con parámetros previamente entrenados. El método permite acelerar el aprendizaje y mejorar la capacidad de generalización. La arquitectura AlexNet popularizó el uso del pre-entrenamiento en efecto al mejor rendimiento que generaba en comparación el entrenamiento de redes tradicional (Krizhevsky *et al*., 2012).

- **Fine-tunning**

  Fine-tunning es un concepto propio del aprendizaje por transferencia. Los modelos se refinan para adaptarse a *datasets* específicos y realizar otras tareas. Algunos modelos reemplazan la última capa de salida para adjuntar una nueva capa para que tome características de nivel inferior (K. He *et al*., 2016; Simonyan & Zisserman, 2014; Szegedy *et al*., 2016).

- **Retropropagación**

  La retropropagación o del inglés *backpropagation* calcula el gradiente de una función objetivo (también denominada costo/pérdida) para delimitar los parámetros de una red para minimizar los errores (Rawat & Wang, 2017).

- **Descenso gradiente**

  Como se vio anteriormente la salida de una capa es la entrada de la siguiente

capa. En la etapa de entrenamiento de una red la distribución de los datos varía entre épocas. Los parámetros de las capas cambian entorpeciendo el entrenamiento que requiere tasas de aprendizaje más pequeños.

El descenso gradiente es un algoritmo cuyo objetivo es encontrar el mínimo de una función dada, en las CNNs es la función de pérdida. Para lo cual realiza dos operaciones: Primero, calcula la gradiente. Segundo, mueve en la dirección opuesta de la gradiente desde el punto actual.

La idea es pasar el conjunto entrenado a través de las capas ocultas de una NN actualizando los parámetros de las capas calculando los gradientes. Así, si la pendiente tiene valor negativo, los pesos son ajustados después de cada época. Algunos de los métodos utilizados son el descenso gradiente estocástico, descenso gradiente por Batch, descenso gradiente por MiniBatch.

Descenso gradiente estocástico (SGD)

El Descenso Gradiente Estocástico (Stochastic Gradient Descent, SGD) es usado en *datasets* de gran tamaño. Las CNN requieren de una gran cantidad de datos para entrenar por consiguiente los *datasets* son grandes. Pero, surge un problema, cuando tenemos muchos datos para entrenar en un único paso el cálculo de los gradientes de cada entrada no resulta ser la mejor alternativa. Por eso el SGD propone considerar una única entrada por paso. Así, a largo plazo la actualización de parámetros es frecuente.

Descenso gradiente Batch

En el descenso gradiente Batch, también conocido como Normalización Batch, todos los datos de entrenamiento son tomados para dar un único paso. Donde, la entrada de la capa se normaliza ajustando el promedio los gradientes de todos los entrenamientos y con ese promedio se actualiza los parámetros.

Descenso gradiente Mini-Batch

Este método es una combinación de la normalización Batch con SGD donde ni todos los datos son utilizados a la vez ni un dato es utilizado una única vez. En su lugar, se usa un lote fijo datos extraídos del *dataset* por entrenamiento. Así, se utiliza a una actualización de parámetros frecuente como también se tiene cálculos rápidos para la inicialización de altas tasas de aprendizaje.

- **Skip/Shortcut**

  Las conexiones por *Skip* o *Shortcut* son conexiones por salto adicionales entre nodos de diferentes capas de una NN que omiten una o más capas del procesamiento lineal. El uso de Skip ha mejorado el entrenamiento de NN profundas (K. He *et al*., 2016).

Clasificación de imágenes con CNN

Rawat y Wang (2017) define la clasificación de imágenes como la tarea de categorizar una serie de imágenes en una de varias clases ya predefinidas, por ejemplo, en la Figura 6 la salida es 'PERRO' o 'es un 90 % probable que sea un PERRO'. Tradicionalmente, algoritmos como: agrupamiento k-means, clasificador Bayes, clasificador de máxima probabilidad, máquinas de soporte vectorial (SVM), clasificador de mínima distancia euclídea, y entre otros (Hemanth & Estrela, 2017), han sido utilizados para la clasificación de imágenes. Sin embargo, su lógica no les permite aprender; en contraste al uso de las CNN cuyos algoritmos han sido desarrollados para que el aprendizaje automático sea posible durante el entrenamiento (Bengio *et al*., 2017).

Cuando un modelo de CNN está entrenado para clasificar imágenes suele buscar características en su nivel base. Por ejemplo, mientras que el ojo humano para identificar un perro busca orejas o un hocico, un algoritmo busca curvaturas u otro parámetro que limite características.

Los retos en la clasificación de imágenes con CNN son diversos entre ellos tenemos la deformación, iluminación, la variación de la escala, la percepción visual, color, borde, orientación, la variación dentro de una misma clase, oclusión de la imagen, y el fondo de la imagen (Hemanth & Estrela, 2017). Algunos de ellos están relacionados directamente con la estructura compleja de una imagen natural.

Algunas aplicaciones de la clasificación de imágenes con CNN se integran a otros sistemas como un sistema de reconocimiento facial, o un sistema de clasificación de residuos, del mismo modo, Facebook usa la clasificación de imágenes para el etiquetado automático, Amazon para la recomendación de productos y Google para buscar a sus usuarios entre fotos. En el caso de la clasificación de hojas como

se detalla en la Sección 1.2 las CNNs obtuvieron resultados superiores a 74 % de exactitud. La Figura 11 presenta una arquitectura genérica de una CNN para la clasificación de hojas.

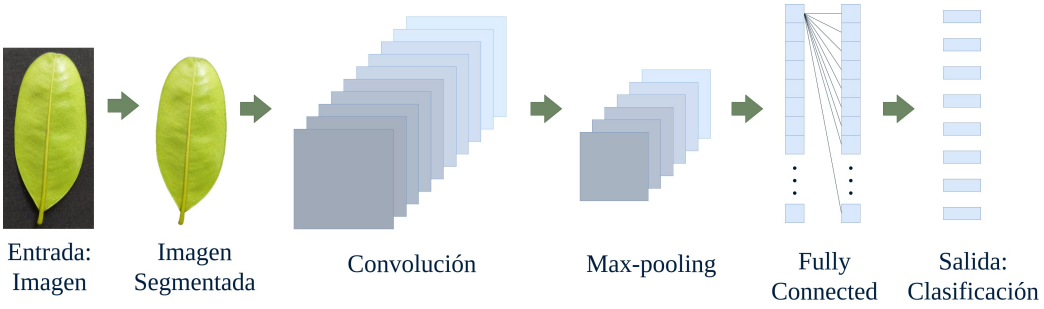


Figura 11. Ejemplo clasificación de hojas

1.1.6 Arquitecturas de Redes neuronales convolucionales

En la presente subsección se expone sobre las arquitecturas AlexNet, VGG-19, ResNet-101 e Inception V3, redes con gran popularidad y diversas reseñas ya que se ubicaron en el top de la Competencia de reconocimiento visual a gran escala de ImageNet (ImageNet Large Scale Visual Recognition Challenge, ILSVRC). La ILSVRC es una competencia anual organizada por el proyecto **ImageNet**, proyecto cuyo fin se basa en proporciona una gran base de datos para su uso en visión artificial. Los resultados obtenidos de estas cuatro arquitecturas en la ILSVRC fueron:

- ILSVRC 2012, AlexNet logró un margen de error de top 5 del 15.3 % (Krizhevsky *et al.*, 2012).
- ILSVRC 2014, VGG obtuvo el segundo lugar con un error de 7.3 % (Simonyan & Zisserman, 2014).
- ILSVRC 2014, Inception logró un margen de error de top 5 del 6.67 % (Szegedy *et al.*, 2015).
- ILSVRC 2015, ResNet obtuvo un margen de error de top 5 del 3.57 % (K. He *et al.*, 2016).

AlexNet

AlexNet, fue publicado por Krizhevsky *et al.* (2012) presentando la primera CNN en implementar una ReLU como función de activación. La arquitectura de Alex-

Net presentó originalmente 5 capas convolucionales seguidas de 3 capas completamente conectadas formando un total de 8 capas.

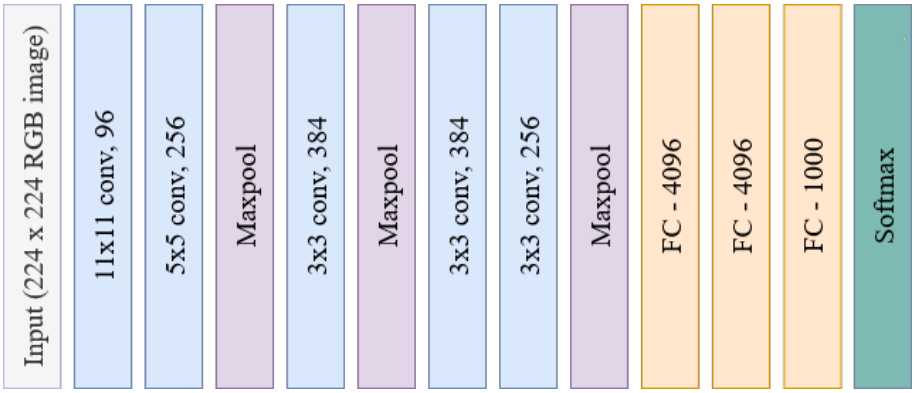


Figura 12. Arquitectura original de AlexNet

Fuente: Adaptado de *Imagenet classification with deep convolutional neural networks*, por Krizhevsky *et al*., 2012

La Figura 12 presenta la arquitectura de Alexnet, donde el Input presenta a la imagen de entrada en canal RGB de 224×224 pixels, conv es la capa convolucional, Maxpool es una capa de *Max pooling*, FC presenta una capa completamente conectada y la última presenta a la capa de la operación de Softmax.

AlexNet resuelve dos problemas principalmente. Primero, tiene un entrenamiento mucho más rápido gracias al uso de ReLU en la parte no lineal, en contraste a una función Sigmoidea. La ventaja de ReLU sobre la función Sigmoidea se debe a sus derivados, mientras que la región de saturación de la función Sigmoidea es de 0 y 1 provocando un estancamiento en el proceso de entrenamiento, ReLU evita el conflicto al ser lineal. Además, AlexNet usa una capa de ReLU después de cada capa de convolución y después de cada capa completamente conectada (Krizhevsky *et al*., 2012). El otro problema resuelto es la reducción de un sobreajuste y/o sobre entrenamiento. Esto debido a que después cada convolución y capa completamente conectada se hace el uso de una capa de Dropout.

VGG-19

VGG-19 es una de las versiones propuestas por la red VGG publicada por Simonyan y Zisserman (2014) el cual contribuye con el diseño de CNNs más profundas

que AlexNet. VGG nace de la necesidad de mejorar el tiempo de entrenamiento y reducir el número de parámetros en cada capa de convolución. El concepto es usar dos capas pequeñas en vez de una. La idea propone reemplazar dos capas convolucionales 3×3 en vez de una capa convolucional 5×5 ya que ambas soluciones cubren la misma área. En consecuencia, no se necesitan capas de gran tamaño. Existen diferentes versiones de la red VGG y su diferencia es la cantidad de capas en cada red neuronal, las versiones con más reseñas son VGG-12 y VGG-19.

La Figura 13 ilustra la configuración inicial de VGG-19. La red VGG-19 consiste en *stacks* lineales de bloques conformada de capas convolucionales, ReLU como función de activación no lineal, una capa *Max Pooling* y 3 capas completamente conectadas y como último una capa de *Softmax*. El 19 representa las 19 capas profundas usadas para el entrenamiento (16 capas convolucionales y 3 capas completamente conectadas). Además, usa estrictamente filtros de 3×3, *stride* de 1x1 y *padding* de 1 píxel. El *Max Pooling* usa *Stride* de 2×2 y *kernel* 2×2.

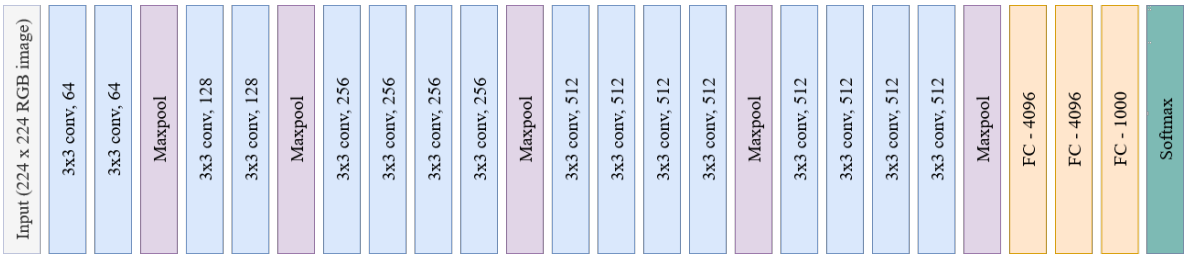


Figura 13. Arquitectura original de VGG-19

Fuente: Adaptado de *Very deep convolutional networks for large-scale image recognition*, por Simonyan y Zisserman, 2014

Inception V3

Inception fue presentado por GoogLeNet Szegedy *et al*. (2015) como un módulo, el cual consiste en un grupo de módulos de inicio creados con la idea de funcionar incluso con las restricciones de bajo presupuesto computacional o de memoria y aun así llegar a redes mucho más profundas. Inception funciona gracias al uso de la convolución de 1×1 a la mitad de la red y su módulo de inicio.

La Convolución de 1×1, presentado por primera vez en Ret en una Ret (Network In Network, NIN) por Lin *et al*. (2013) es usado junto a la función ReLU como

módulo de reducción de las dimensiones a fin de simplificar el cálculo. Este diseño fue conocido como **Cuello de botella**, el cual, permite aumentar la profundidad y ancho de la red. Por ejemplo, se desea convolucionar $28 \times 28 \times 192$ mapas de características:

- Primera solución convencional:
  Input: $[28 \times 28 \times 192] => [5 \times 5 conv, 32] => [24 \times 24 \times 32]$
  $(28 \times 28 \times 32) \times (5 \times 5 \times 192) = 120{,}422$ millones de operaciones
  Total: 120.422 millones de operaciones
- Segunda solución usando convolución 1×1 antes de la capa convolucional 5×5.
  Input: $[28 \times 28 \times 192] => [1 \times 1 conv, 16] => [28 \times 28 \times 16]$
  $[28 \times 28 \times 16] => [5 \times 5 conv, 32] => [24 \times 24 \times 32]$
  $(28 \times 28 \times 16) \times (1 \times 1 \times 192) = 2{,}4$ millones de operaciones
  $(28 \times 28 \times 32) \times (5 \times 5 \times 16) = 10$ millones de operaciones
  Total: 12.4 millones de operaciones

Módulo de Inicio, es usado para dar aún más profundidad a la red. Al usar ReLU después de cada convolución 1×1 se busca tener menos procesos lineales. Por ejemplo, la Figura 14 presenta el funcionamiento del módulo de inicio, a la izquierda 14.a cada convolución de 1×1, 3×3, 5×5 y max pooling 3×3 después de ser operados son concatenados para posteriormente ser usado como entrada para el siguiente módulo. En la derecha 14.b se usa módulo de inicio con la convolución 1×1 obteniendo el mismo fin, pero con menos operaciones.

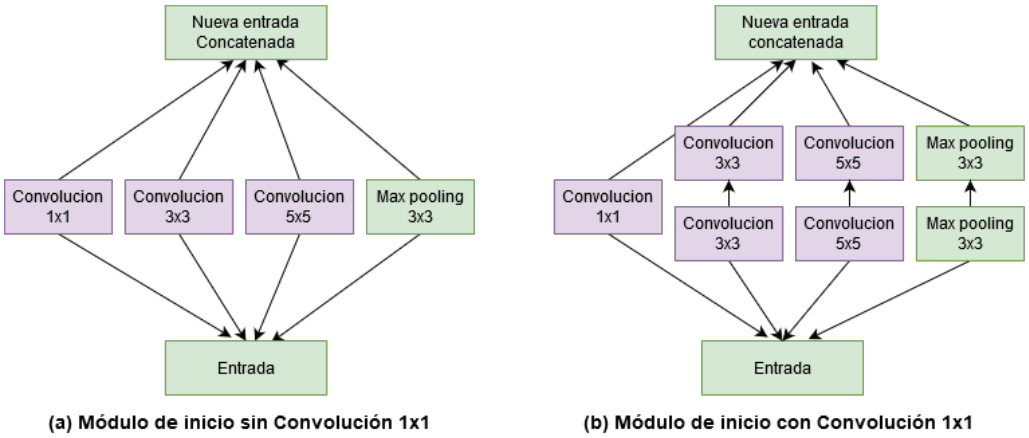


Figura 14. Funcionamiento del módulo de inicio en Inception

Fuente: Adaptado de *Going deeper with convolutions*, por Szegedy *et al.*, 2015

Son cuatro las versiones de Inception, pero la que presenta mejores y mayor número de reseñas es la versión V3. Inception V3 (Szegedy *et al.*, 2016) presenta tres tipos de inicios que están influenciados por la forma de factorizar sus convoluciones:

- **Factorización en pequeñas convoluciones**, consiste en cambiar una convolución grande en otras dos más pequeñas. Por ejemplo, una convolución de 5×5 es reemplazada por dos convoluciones de 3×3. Mientras que la convolución de 5×5 genera 25 parámetros, el número de parámetros generados por las dos convoluciones de 3×3 es 18.
- **Factorización asimétrica dentro de convoluciones**, una convolución grande es reemplazada por dos convoluciones asimétricas de $n \times 1$ y $1 \times n$. Por ejemplo, la convolución de 3×3 puede ser reemplazada por una convolución de 3×1 seguida por una convolución de 1×3. La convolución 3×3 genera 9 parámetros, por otro lado, las convoluciones 3×1 y 1×3 generan 6 parámetros.

Los módulos de inicio propuestos en Inception V3 se representan en la Figura 15, donde a la izquierda superior 15.a se presenta al módulo de inicio original. A la derecha superior 15.b a un nuevo inicio con convoluciones pequeñas. A la izquierda inferior 15.c se muestra la convolución usando la factorización asimétrica. Y hacia la derecha inferior 15.d se presenta un nuevo inicio para aquellos con

dimensiones grandes.

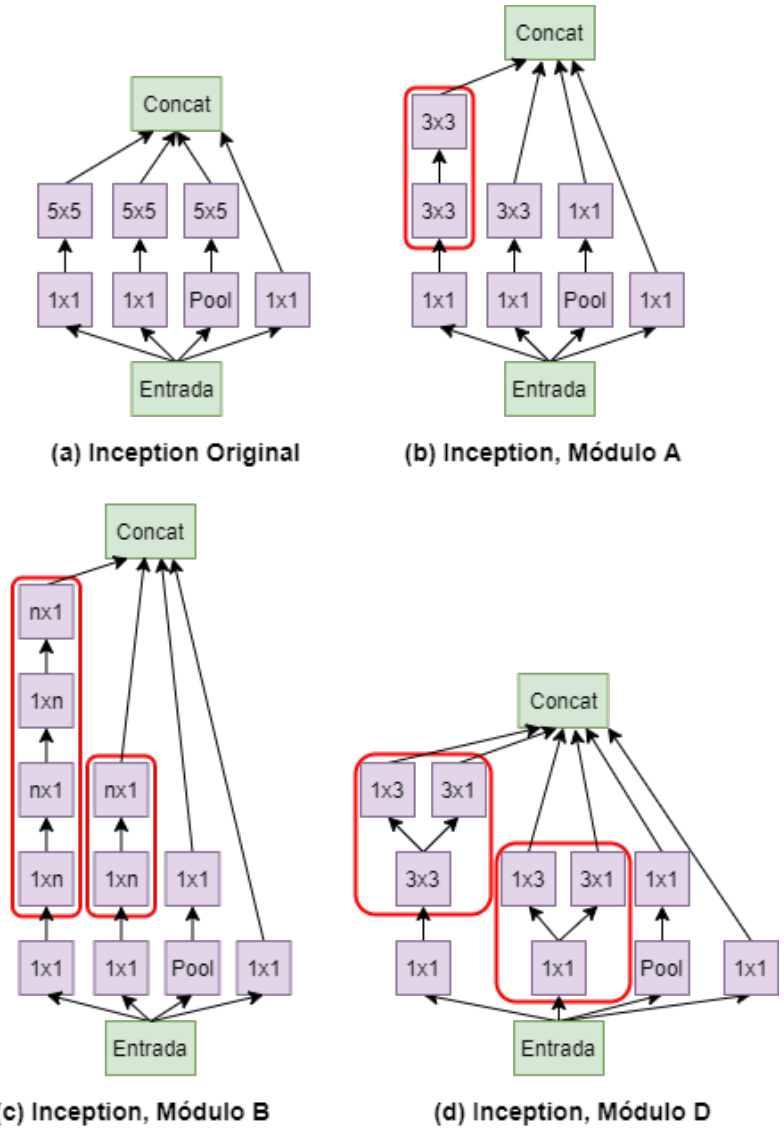


Figura 15. Módulos de inicio según su factorización en Inception

Fuente: Adaptado de *Going deeper with convolutions*, por Szegedy *et al.*, 2016

Teniendo en cuenta lo anteriormente expuesto la arquitectura de Inception V3 es presentada en la Tabla 1 con un total de 48 capas profundas. Hace el uso de redes pre-entrenadas, un clasificador auxiliar como regularizador y la reducción eficiente de las dimensiones/cuadrados. Además, se usa un *Batch* y ReLU después de cada convolución.

Tabla 1
Esquema original del modelo Inception V3

Tipo	Tamaño del Patch/recuadros	Tamaño de entrada
Convolución	$3 \times 3/2$	$299 \times 299 \times 3$
Convolución	$3 \times 3/1$	$149 \times 149 \times 32$
Padding-Convolución	$3 \times 3/1$	$147 \times 147 \times 32$
Pooling	$3 \times 3/2$	$147 \times 147 \times 64$
Convolución	$3 \times 3/1$	$73 \times 73 \times 64$
Convolución	$3 \times 3/2$	$71 \times 71 \times 80$
Convolución	$3 \times 3/1$	$35 \times 35 \times 192$
3xInception	Módulo **A**	$35 \times 35 \times 288$
5xInception	Módulo **B**	$17 \times 17 \times 768$
2xInception	Módulo **C**	$8 \times 8 \times 1280$
Pooling	8×8	$8 \times 8 \times 2048$
Linear	Lógico	$1 \times 1 \times 2048$
Softmax	Clasificador	$1 \times 1 \times 1000$

Fuente: Adaptado de *Rethinking the Inception Architecture for Computer Vision*, por Szegedy *et al*., 2016

ResNet-101

ResNet-101 es una variante de ResNet publicado por K. He *et al*. (2016) que diseñaron una CNN más profunda cuyo fin es omitir las conexiones comunes haciendo uso de funciones residuales. Razón por la que RestNet puede tener una red muy profunda con varias capas. Sin embargo, cuando se utiliza este método surge el problema del desvanecimiento de gradiente. Por lo que, ResNet utiliza la conexión por *skip/shortcut* pues ajusta la entrada en la capa anterior a la siguiente capa sin modificar la entrada. Además, se agrega una conexión de acceso directo para agregar la entrada x a la salida de algunas capas. La Figura 16 muestra este ajuste residual, donde la formulación $F(x)+x$ pude realizarse como conexiones de acceso directo que saltan una o más capas. Por lo tanto, las capas aprenden por el mapeo residual $F(x) = H(x) - x$, y la entidad x es usada para transferir de nuevo

a las capas anteriores en caso de desvanecimiento de la gradiente, finalmente la salida es $H(x) = F(x) + x$.

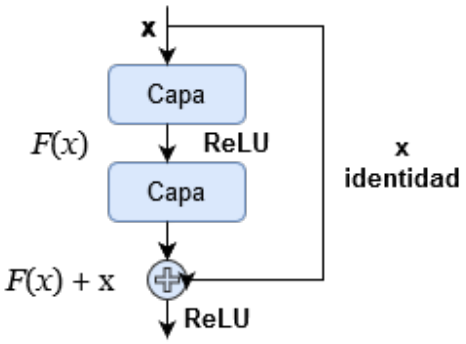


Figura 16. Un bloque parte de una red residual

Fuente: Adaptado de *Deep residual learning for image recognition*, por K. He *et al*., 2016

ResNet usa, además, el diseño del cuello de botella como función residual al inicio y al final de la red una capa convolucional de 1×1, así como se muestra en la Figura 17 donde se ejemplifica el uso del cuello de botella en ResNet el cual es similar al planteado por GoogleNet cuando presentó NIN (Lin *et al*., 2013).

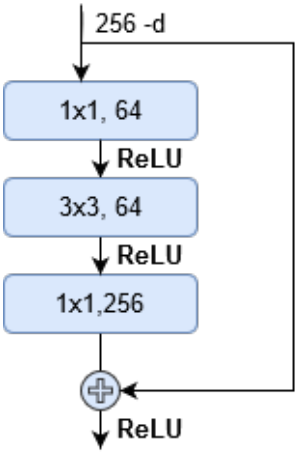


Figura 17. Uso del cuello de botella en ResNet

Fuente: Adaptado de *Deep residual learning for image recognition*, por K. He *et al*., 2016

La Tabla 2 muestra la arquitectura RestNet-101, compuesta por un total de 347 capas que corresponden a un total de 101 capas de una red residual.

Tabla 2
Arquitectura original del modelo Resnet-101

Capa	tamaño de salida	Capa 101
Conv1	112×112	7×7, 64, stride 2
Conv2_x	56×56	3×3 pool, stride 2
		$\begin{pmatrix} 1\times1,\ 64 \\ 3\times3,\ 64 \\ 1\times1,\ 256 \end{pmatrix} \times 3$
Conv3_x	28×28	$\begin{pmatrix} 1\times1,\ 128 \\ 3\times3,\ 128 \\ 1\times1,\ 512 \end{pmatrix} \times 4$
Conv4_x	14×14	$\begin{pmatrix} 1\times1,\ 256 \\ 3\times3,\ 256 \\ 1\times1,\ 1024 \end{pmatrix} \times 23$
Conv5_x	7×7	$\begin{pmatrix} 1\times1,\ 512 \\ 3\times3,\ 512 \\ 1\times1,\ 2048 \end{pmatrix} \times 3$
	1×1	Average maxpool, $1000 - d$ fc, $softmax$
FLOPs		$7{,}6 \times 10^9$

Fuente: Adaptado de *Deep residual learning for image recognition*, por K. He *et al*., 2016

1.1.7 Métricas

Para evaluar la efectividad de un modelo pre-entrenados como AlexNet se usan diversas medidas de desempeño cuantitativas (Goodfellow *et al*., 2016). Una herramienta para evaluar el desempeño de un clasificador automático es haciendo uso de la **matriz de confusión**. Por lo general, es usada dentro del aprendizaje supervisado y permite visualizar si el clasificador está discriminando correctamente las clases. La Figura 18 muestra una matriz de confusión donde las columnas representan el número de predicciones y las filas son ejemplos reales. El resultado muestra cuatro opciones: **TP** verdaderos positivos, **FP** falsos positivos, **FN** los falsos negativos, y los **TN** verdaderos negativos.

Matriz de confusión en la clasificación multiclase para 3 o más clases el número de predicciones correctas e incorrectas es e_{ij}, donde cada columna de la matriz representa el número de predicciones por clase j , mientras que cada fila representa la clase real i. Las predicciones en la diagonal n_{ii} son predicciones correctas; mientras las predicciones fuera de la diagonal son incorrectas. Por ejemplo, del *dataset* con las clases perro, gato y paloma la Figura 19 presenta una hipotética matriz de confusión:

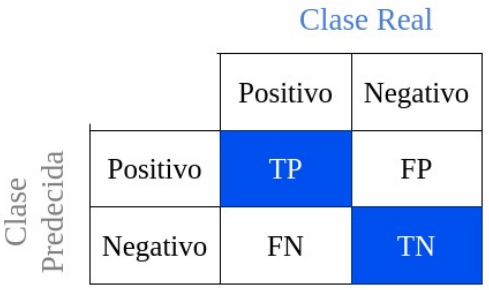


Figura 18. Matriz de confusión

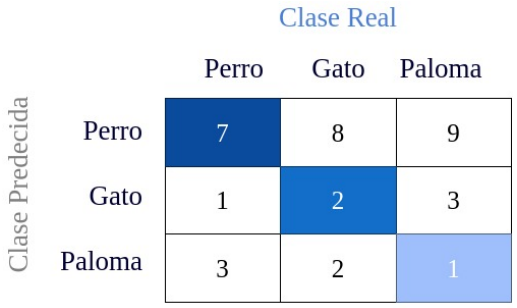


Figura 19. Ejemplo matriz de confusión en la clasificación multiclase

Una vez constituida la matriz de confusión de un modelo, se realiza la comparación de desempeño con algoritmos de clasificación de datos utilizando parámetros de medición como la exactitud de clasificación, exhaustividad, precisión. Estos parámetros se determinan utilizando los términos verdadero positivo (TP), verdadero negativo (TN),falso negativo (FN) y falso positivo (FP), de la siguiente manera:

La **exhaustividad** del inglés *recall* expresa el total de verdaderos positivos que fueron identificados correctamente. Si el resultado es 1 se obtiene la exhaustividad máxima posible teniendo una clasificación perfecta. Por lo contrario, si el valor es 0 los resultados obtenidos no poseen relevancia alguna.

$$Exhaustividad = \frac{TP}{TP+FN} \tag{4}$$

Tenemos la **especificidad** es el total de negativos verdaderos que fueron identificados correctamente.

$$Especificidad = \frac{TN}{TN+FP} \tag{5}$$

La **exactitud** corresponde a la proporción de identificaciones correctas. Es la medida más utilizada, pero no funciona bien si se tiene clases desbalanceadas.

$$Exactitud = \frac{TN+TP}{TP+FP+TN+FN} \tag{6}$$

Por otro lado, las medidas anteriormente mencionadas funcionan perfectamente en *dataset* ideales. Otras medidas usadas en tareas de clasificación de imágenes son: precisión, Tasa de falsos positivos, error, y el valor-F.

La **precisión** es usada a fin de medir la calidad de un modelo. Cuando el resultado se acerca a 0 los resultados son menos relevantes. Mientras que si el valor se aproxima a 1 será un resultado relevante.

$$Precisión = \frac{TP}{TP+FP} \tag{7}$$

La medida del **error** representa la cantidad de identificaciones incorrectas sumando los falsos positivos y falsos negativos en comparación al total de identificaciones.

$$Error = \frac{FP+FN}{TP+TN+FN+FP} \tag{8}$$

El **valor-F** combina las medidas de precisión y exhaustividad a fin de poder medir el rendimiento del modelo. El valor F varia y depende del valor entre las medidas. Un valor de 1 se interpreta como si la precisión y la exhaustividad tienen igual importancia. De manera genérica β representa el valor de F.

$$Valor-F = \frac{1+\beta^2(Precisin * exhaustividad)}{\beta^2 \times precisin + exhaustividad} \tag{9}$$

1.1.8 Interpretabilidad

La interpretabilidad de un modelo consiste en la habilidad (del modelo) de dar una explicación en términos humanos sobre como se lleva a cabo un proceso en particular (Li *et al.*, 2021). Si bien se tienen resultados cuantificativos establecidos por métri-

cas al tener un modelo complejo surge la necesidad humana de comprender como funciona, este proceso no busca conocer cada detalle sobre el modelo, sino sobre las tareas con mayor influencia, las reglas y/o patrones por las cuales el modelo aprende y toma decisiones.

Por lo tanto, la interpretabilidad resalta en modelos de *deep learning* al ser representados hacia el entendimiento humano. Desafortunadamente, no existe una medición formal para evaluar la interpretabilidad de un modelo, sin embargo, actualmente existen enfoques que ayudan a medir la interpretabilidad de un modelo, Li *et al*. (2021) señala tres:

- **Disección de la red**: consiste en tener un conjunto de datos etiquetados donde cada imagen está dividida por características convenientes como la textura o el color. Para posteriormente recuperar el mapa de características utilizados por el modelo para la tarea de clasificación.
- **Por consenso**: primero se calcula las interpretaciones a través de un algoritmo, para luego ser evaluado por un consenso, finalmente es sometido a votación.
- **Evaluación del usuario**: este es evaluado mediante experimentos realizados por parte de los usuarios.

1.1.9 Dataset

Para entrenar una NN se requiere de un conjunto de datos o también conocido por el anglicismo *dataset*. Un *dataset* es una colección de datos habitualmente tabulados. Dentro la comunidad científica y otros círculos de estudios relacionados a la clasificación de imágenes de hojas de especies forestales se manejan conjuntos de datos o *datasets* ya existentes, tales como: Flavia (S. Wu *et al*., 2007), Leafsnap (Kumar *et al*., 2012), BJFU101 (Sun *et al*., 2017), los cuales son de libre acceso y se pueden encontrar en la nube. Los siguientes *datasets* a exponer son aquellos que son mencionados en el presente trabajo.

LeafSnap consta de 185 especies de árboles del noroeste de Estados Unidos. Con un total de 23 916 imágenes, 23 147 son imágenes de alta calidad de hojas prensadas y 7 719 son imágenes tomadas con dispositivos móviles (Kumar *et al*., 2012).

Flavia consiste en 15 especies con 72 imágenes por cada especie. Las imágenes consisten en fotografías hojas frescas tomadas por cámaras digitales o escáneres. El formato de las imágenes es en JPEG con una resolución de 800*x*600 Las hojas fueron recolectadas del jardín botánico Sun Yat-Sen en el campus de la Universidad de Nanjing, Nanking, China (S. Wu *et al*., 2007).

PlantVillage contiene un total de 61 486 imágenes 39 clases correspondientes a los tipos de enfermedades en hojas. Las imágenes fueron tomadas en asociación con las concesiones de las Universidades de Estados Unidos *(Land Grant Universities)*. Usaron una cámara digital de 20.2 megapixeles (Geetharamani & Pandian, 2019).

LifeCLEF2017 la edición del 2017 del LifeCLEF presenta el reto de PlantCLEF con más de 10 000 clases. Las más de 1,1 millones de imágenes corresponden a plantas de Europa y Norte América. La variación que presenta esta edición es que tiene imágenes con ruido que buscan '*confundir*' el entrenamiento (Goeau *et al*., 2017).

LifeCLEF 2019 en esta edición se buscó evaluar la identificación de la flora con datos deficientes. Por lo que las imágenes fueron recolectadas de Guyana y del norte de la selva Amazónica. Esta edición presenta 10000 especies, pero las imágenes por especie varían desde una imagen por especie. De las 434 251 imágenes en total, 375 632 fueron recuperadas automáticamente de los buscadores Google y Bing (Goëau *et al*., 2019).

BJFU100 *dataset* contiene 10 000 imágenes de plantas ornamentales. Las imágenes corresponden a 100 especies del campus de la Universidad Forestal de Beijing, China. Las muestras están tomadas con dispositivos móviles en ambientes naturales. El *dataset* no solamente contiene imágenes de hojas sino también de plantas completas (Sun *et al*., 2017).

Foliage consiste en 6 900 imágenes pertenecientes a 60 tipos de hojas. Es decir, 115 aproximadamente muestras por tipo de hoja (Kadir *et al*., 2012).

En **CIFAR-10** as imágenes son en baja calidad de 32x32 píxeles este *dataset* popular que contiene 60 000 imágenes de 10 clases. Lo que permite realizar pruebas rápidas para ver el desempeño de un algoritmo (Krizhevsky, 2009).

ImageNet es un *dataset* usado para realizar estudios con imágenes que están orde-

nadas según el concepto de WordNet y organizadas según una categoría (sustantivo). ImageNet en sí como *dataset* no posee con un repositorio unificado, en su lugar ofrece la dirección única de cada imagen. Además, desde el 2010 pasó a ser el proyecto ImageNet y lanzó un concurso anual ILSVRC donde diversos algoritmos son evaluados en tareas de clasificación de imágenes y detección de objetos (J. Deng *et al*., 2009).

A continuación, la Tabla 3 presenta las características de interés sobre aquellos *datasets* de imágenes de hojas descritos anteriormente.

Tabla 3
Datasets de imágenes de hojas

Dataset	**Clases**	**Imágenes**	**Lugar**	**Imagen**
Leafsnap	185	23 916	Noroeste EE.UU	300×400
Flavia	15	1 800	Nanking, China	227×277
PlantVillage	39	61 486	EE.UU	Estándar
LifeCLEF 2017	10 000	1.1 Millones	Europa y Norteamérica	Variante
LifeCLEF 2019	10 000	434 251	Guyana y Amazonía	Variante
BJFU100	100	10 000	Beijing, China	3120 x 4208

1.2 Antecedentes

La primera Subsección 1.2.1 presenta los antecedentes orientados al reconocimiento de una determinada flora haciendo uso de técnicas de aprendizaje profundo. La segunda Subsección 1.2.2, expone el estado del arte respecto al reconocimiento de especies forestales de la Amazonía. La tercera Subsección 1.2.3, ilustra sobre aquellos trabajos comparativos en la clasificación de imágenes.

1.2.1 Trabajos sobre clasificación de plantas

H. Zhang *et al*. (2021) propusieron un modelo de aprendizaje profundo para clasificar especies de plantas a gran escala, para el entrenamiento integraron una CNN muy profunda y un árbol de decisión para la extracción de características morfológicas fijas de la planta. Los datos utilizados fueron obtenidos de Orchid2608 Dataset, el

cual consiste en 1,980,000 imágenes de 2,608 especies pertenecientes a la familia Orchid. El modelo propuesto obtuvo un 72.28 % de exactitud.

Saini *et al*. (2020) demostraron que la clasificación de plantas haciendo uso de CNN obtiene mejores resultados que los métodos tradicionales. Usaron 5000 imágenes de dos especies extraídas del Dataset PlantVillage. Obtuviendo una exactitud de 99.90 %. Resultado superior a métodos como el vecino más cercano, máquinas de vectores de soporte o un algoritmo genético de clasificación.

Ashqar *et al*. (2019) implementaron un modelo de clasificación para plántulas o plantas de semillero. El modelo estuvo basado en la arquitectura VGG-16 y modificado en la capa completamente conectada (fully connected) a 12 clases. Para ello, utilizaron 5608 imágenes de 12 especies en sus distintas etapas de desarrollo proporcionado por la Universidad de Aarhus en colaboración con la Universidad de Dinamarca. La exactitud obtenida del modelo entrenado fue de 98.57 %.

Elnemr (2019) aplicó una arquitectura de CNN simple para la clasificación de plántulas. Las métricas utilizadas para su evaluación fueron: exactitud, precisión, exhaustividad y F1-Score. La cantidad de imágenes utilizadas fueron 5539 de 12 especies aportados por el grupo especializado de la Universidad de Aarchus. La arquitectura obtuvo una exactitud de 94,38 %. Sin embargo, al ser más simple en comparación a otras arquitecturas, el autor sugirió agregar la etapa de segmentación.

Tan *et al*. (2018) utilizaron la morfometría de la vena de la hoja para clasificar especies de plantas. Utilizaron un total de 1290 imágenes de 43 especies recolectados en el campus de la Universidad de Malaya, Kuala Lumpur, Malasia. Después de procesar las imágenes, usaron los modelos AlexNet previamente entrenado, AlexNet y D-Leaf (modelo propuesto basado en Alexnet) para la extracción de características. El modelo propuesto D-Leaf obtuvo 94.88 % de exactitud.

Esmaeili y Phoka (2018) clasificaron hojas haciendo uso de las arquitecturas Inception, ResNet y MobileNet. Además, utilizaron el aprendizaje por transferencia al no tener una gran cantidad de datos. Como Dataset usaron Flavia que contiene menos de 900 imágenes de 10 especies del norte de Tailandia. El trabajo se enfocó en demostrar que el tiempo de entrenamiento reduce cuando se aplica el aprendizaje por transferencia en modelos previamente entrenados. Sin embargo, la precisión se

reduce en un 25 % y 30 %.

Rizk (2019) propuso un modelo de clasificación de hojas basado en dos criterios de extracción de características: de su forma y de la venación. Los datos que utilizados fueron obtenidos de Flavia Dataset usando un total de 1907 imágenes de 32 especies. Concluyendo que la identificación de la forma de hoja es esencial; en consecuencia, la venación es secundaria. El modelo propuesto obtiene 96.80 % de exactitud.

Sun *et al*. (2017) realizaron la identificación de plantas en sus ambientes naturales mediante la implementación del modelo llamado ResNet26 basado en el framework Keras. Además, presentaron el *Dataset* BJFU100 que consiste en 10000 imágenes de 100 especies ornamentales recolectados del campus de la Universidad Forestal de Beijing. BJFU100 está abierto a la comunidad y sugieren el uso de las muestras en sus distintas fases del ciclo de vida. Respecto al modelo ResNet26 tiene un 91.78 % de exactitud.

Zhu *et al*. (2018) propusieron un método de alto rendimiento basado en técnicas de aprendizaje profundo para la clasificación de vegetales. Usaron el modelo de AlexNet desarrollado en Caffe para su entrenamiento. Y el conjunto de datos utilizados fueron extraídos de ImageNet Dataset siendo un total de 24000 imágenes. La exactitud obtenida por este método fue de 92.10 %.

G. He *et al*. (2018) desarrollaron un modelo basado en CNN con dos canales que fueron entrenados por separado y posteriormente fusionados en la última capa aprendiendo conjuntamente. Sobre estas dos sub-capas una está basada en VGG-16 y la otra basada en SquuezeNet. Los modelos fueron preentrenados con ImageNet1000, para el entrenamiento utilizaron los datos de Orchidaceae Dataset que consta de 18211 imágenes. Finalmente obtuvieron un 96.81 % de exactitud.

Wick y Puppe (2017) implementaron una CNN para la identificación de hojas. La CNN consta de cuatro capas convolucionales seguidas por una capa de max pooling con la función de activación ReLU, y una capa completamente conectada con dropout. Además, usaron como datos 60 imágenes por especie del dataset Flavia y 120 por especie del dataset Foliage. La exactitud de la red propuesta fue de 99.67 % con Flavia, 99.40 % con Foliage.

Toma *et al*. (2017) participaron en el PlantCLEF 2017 en la clasificación de plantas. Su propuesta se basó en el modelo AlexNet. Los datos del PlantFLEF 2017 consistió en 260 000 imágenes correspondientes a 10 000 especies. Para su entrenamiento usaron el aprendizaje por transferencia y usaron ruido en sus imágenes, debido a que no usaron las imágenes en blanco y negro ni de baja calidad. Durante su entrenamiento obtuvieron un 97.85 % de exactitud.

Barré *et al*. (2017) propusieron LeafNet un sistema para la clasificación de plantas basado en una arquitectura CNN, Usaron tres *Datasets* LeafSnap (23 147 imágenes de 184 especies), Flavia (imágenes entre 50 y 60 por 32 especies) y Foliage (7 200 imágenes de 60 especies). Los resultados en exactitud según el *Dataset* fueron LeafSnap 86.30 %, Foliage 95.80 % y Flavia 97.90 %. Resultados superiores a sistemas tradicionales como LeafSnap.

Carpentier *et al*. (2018) usaron la red ResNet para clasificar árboles por medio de su corteza. A su vez proponen su *Dataset* BarnetNet que consiste en 23 000 imágenes de 23 especies de un parque cerca de la ciudad de Quebec en Canadá. La exactitud obtenida fue del 93.88 %. También, demostraron que al maximizar el número de individuos de árboles en el *dataset* durante el entrenamiento aumenta la exactitud.

1.2.2 Trabajos con especies forestales de la Amazonia

Montenegro (2018) propuso la clasificación de especies a través de sus características físicas y mecánicas de la madera. Bajo la idea de conglomerados usaron el algoritmo CLARA. Que fue diseñado de forma similar al aprendizaje no supervisado ya que de inicio se selecciona una muestra aleatoria buscando que aprenda por sí mismo, Sin embargo, para que funcione necesita que los datos tengan algún patrón similar.

Reátegui y Velasco (2018) propusieron un sistema de reconocimiento para el Camu Camu, arbusto nativo, de la Amazonia peruana. El sistema consistió en dos componentes una aplicación web y el algoritmo para el reconocimiento. Para el reconocimiento se usaron un total de 2 800 imágenes de hojas y una CNN como identificador. El resultado del reconocimiento fue de 93.00 % de precisión.

Apolinario *et al*. (2019) usaron una CNN a fin de reconocer especies forestales de Perú por medio de su madera. Los datos que usaron fueron proporcionados de la

Universidad Nacional Agraria de La Molina en Lima, Perú. El número de imágenes usadas fueron 355 correspondientes a 16 especies. La exactitud obtenida del modelo propuesto fue del 91.62 %.

Chulif *et al*. (2019) participaron en el LifeCLEF 2019 en la identificación de plantas de la Amazonia de Guyana. Usaron 258 909 imágenes del *data set* LifeCLEF correspondientes a 10 000 especies. Como CNN usaron Inception-v4 y Inception-ResNet-V2. Los resultados obtenidos durante su entrenamiento fue una precisión Top-1 de 0.246.

Goëau *et al*. (2019) presentaron el problema del LifeCLEF 2019 sobre la identificación de plantas en una ubicación tropical. Los lugares tropicales de la Amazonia son considerados con datos deficientes. EL conjunto de datos consiste en 10 000 especies obtenidas de Guyana. El equipo con mejor resultado fue de Chulif *et al*. (2019), de los que se pudo observar que la flora tropical era mucho más difícil de identificar que la flora de Europa y Norte América.

Según las Subsecciones 1.2.1 y 1.2.2 la Tabla 4 presenta un resumen de los antecedentes donde se extrae información de cada trabajo como la característica del individuo que se usó para la clasificación, la cantidad de especies e imágenes utilizadas en la investigación, gracias a los datos extraídos se comprueba la teoría sostenida según varios trabajos analizados que la cantidad de imágenes influyen en la etapa de entrenamiento y validación, y en el método de clasificación empleado.

Tabla 4
Trabajos sobre clasificación de plantas

Nro.	*Dataset*	Característica	Nro. Especie	Nro. imágenes	Clasificador	Exactitud	Referencia
			Redes Neuronales Convolucionales				
1	Orchid2608	planta:flor	2608	1980000	Personalizada + árbol de decisión	72.28 %	H. Zhang *et al.* (2021)
2	PlantVillage	hoja: ciclo vital	2	5000	Personalizada	99.90 %	Saini *et al.* (2020)
3	Personalizada	hoja: plántula	12	5608	VGG-16	98.57 %	Ashqar *et al.* (2019)
4	Personalizada	hoja: plántula	12	5539	Personalizada	94.38 %	Elnemr (2019)
5	Personalizada	hoja: vena	43	1290	AlexNet	94.84 %	Tan *et al.* (2018)
6	Flavia	hoja	10	900	Inception ResNet MobileNet	-[25,30] %	Esmaeili y Phoka (2018)
7	Flavia	hoja: forma,vena	32	1907	Personalizada	96.80 %	Rizk (2019)
8	BJFU100	ambiente natural	100	10000	ResNet26	91.78 %	Sun *et al.* (2017)
9	ImageNet	ambiente natural	-	24000	AlexNet	92.10 %	Zhu *et al.* (2018)
10	Orchidaceae	hoja	-	18211	VGG-16	96.81 %	G. He *et al.* (2018)
11	Flavia Foliage	hoja	*N* *M*	60*xN* 120*xM*	Personalizada	99.67 % 99.40 %	Wick y Puppe (2017)
12	PlantCLEF 2017	planta:órganos	10000	260000	AlexNet	97.85 %	Toma *et al.* (2017)
13	Flavia Foliage LeafSnap	hoja	32 60 184	[50,60]x32 7 200 23 147	Personalizada	97.90 % 86.30 % 95.80 %	Barré *et al.* (2017)
14	BarnetNet	corteza	23	23000	ResNet	93.88 %	Carpentier *et al.* (2018)
15	Personalizada	hoja	1	2800	Personalizada	93.00 %	Reátegui y Velasco (2018)
16	Personalizada	madera	16	355	Personalizada	91.62 %	Apolinario *et al.* (2019)

Continuación de la Tabla							
Nro.	***Dataset***	**Característica**	**Nro. Especie**	**Nro. imágenes**	**Clasificador**	**Exactitud**	**Referencia**
Redes Neuronales Convolucionales							
17	LifeCLEF 2019	planta:órganos	10000	258909	Inception Inception-Resnet	96.81 %	Chulif *et al*. (2019)
Otros							
18	personalizada	madera	-	-	CLARA	-	Montenegro (2018)
Fin de la Tabla							

1.2.3 Trabajos comparativos en clasificación de imágenes

Wang *et al*. (2019) analizaron las arquitecturas AlexNet, VGG, GoogLeNet, Inception V2, Inception V3, PReLU, ResNet, ResNeXt, SenseNet, SENet. Las CNNs estudiadas utilizaron los *dataset* ImageNet y CIFAR-10. Los resultados señalan el impacto, evaluación del algoritmo. Como indicador de evaluación usaron Top -5.

Qian *et al*. (2020) con el fin de controlar a plantas invasoras. Utilizaron varios tipos de CNNs LeNet, AlexNet, VGG-11 y su propia arquitectura que denominaron IAPsNET. Para lo que usando 8 clases y 7293 imágenes de $112x122$. Para medir el desempeño de las redes usaron la exactitud que obtuvo un 93.39 %, exhaustidad, precisión y el valor-F con 93.30 %, 93.74 % y 93.52 % respectivamente.

Su *et al*. (2018) analizaron la relación entre la robustez de las Redes neuronales profundas y su exactitud en la tarea de clasificación. A una mayor exactitud reduce la robustez. Estudiaron las redes AlexNet, VGG, Inception, ResNet, DenseNet, MobileNet, NASNet y sus variantes haciendo un total de 18 redes. Usaron como métricas la tasa de éxito, distorsión, la puntuación CLEVER y la transferibilidad.

Pham (2020) realizó la clasificación de dos tipos de diagnósticos positivos o negativos al Covid-19. Para ello utilizó un total de 746 imágenes de tomografías. Las dieciséis CNNs estudiadas fueron AlexNet, GooglNet, SqueezeNet, ShuffleNet, ResNet, Xception, Inception, Inception-ResNet, VGG ,DenseNet,MobileNet,NasNet y sus derivados. Los resultados obtenidos según su exactitud son de 92.62 %. Al aplicar un aumento de datos la exactitud aumentó a un 96.20 %.

Maeda *et al*. (2020) estudiaron las CNNs usaron el *dataset* PlantVillage para clasificar tomates enfermos. Usaron un total de 18160 imágenes $(224 \times 244$ y $299 \times 299)$ divididas en nueve clases enfermas y una sana. El desempeño de las redes fueron medidas en exactitud, precisión, exhaustividad, *(sensitivity)* y el valor-F. La red con mejor exactitud fue GoogleNet con 99.39 % seguido por ResNet-50 con 99.15 % de exactitud.

Adit *et al*. (2020) analizaron el desempeño de las CNNs para la clasificación de enfermedades en plantas. Usaron un *dataset* obtenido de Kaggle que contiene 38 cases y 2000 imágenes de $256x256$. Sus experimentos muestran una ligera ventaja

de Inception V3 sobre AlexNet y LeNet.

Syarief y Setiawan (2020) presentaron una clasificación de enfermedades en las hojas de maíz. Usaron 7 arquitecturas AlexNet, VGG-16, VGG-19, GoogLeNet, Inception V3, ResNet50, ResNet101. Su *dataset* consiste en 200 imágenes y cuatro clases. Las métricas para su evaluación fueron exactitud, exhaustividad y precisión obteniendo 93.50 %, 95.08 %, y 93.00 %, respectivamente.

Too *et al*. (2019) realizaron la identificación de las enfermedades en las plantas. Compararon las CNNs Inception V4, VGG 16, ResNet 50, ResNet101, ResNet152 y DenseNet. Usaron 38 clases de 54306 imágenes del *dataset* PlanVillage. Compararon exactitud, parámetros, función de pérdida y tiempo. Su mejor resultado se da en la época 30 con 99.75 % de exactitud.

Saikia *et al*. (2019) evaluaron las redes VGG16, VGG19, GoogLeNet V4 (Inception V3) y ResNet 50 para ayudar a la clasificación de FNAC en la detección del cáncer de mama. Usando dos clases benigna y maligna usaron 212 imágenes que fueron aumentadas y limpiadas haciendo un total de 2120 imágenes. Su mejor resultado fue por la red GoogLeNet-V3 *(fine-tuned)* con una exactitud del 96.25 %.

Kaur y Gandhi (2020) evaluaron las redes AlexNet, VGG16, VGG19, GoogLeNet, Inception V3, ResNet 50, ResNet 101, InceptionResNet V2 en la tarea de clasificar imágenes cerebrales. Con tal fin usaron tres versiones de *datasets* V1 V2 V3 con un total de 284 imágenes. Las métricas empleadas fueron la exactitud, exhaustividad, precisión, negativos positivos, error, falsos positivos, valor-F, MCC, Cohen's kappa y exactitud random. Los resultados muestran los diferentes desempeños de las redes en distintos ambientes llegando a obtener resultados superiores a una exactitud del 96.95 %.

Tabla 5
Trabajos comparativos en clasificación de imágenes

Nro.	Clases	Imágenes	Arquitectura CNN	Medida	Exactitud	Referencia
1	8	6400	Personalizado LeNet AlexNet VGG-11	Exactitud Exhaustividad Precisión Valor-F	93.39 %	Qian *et al*. (2020)
2	-	1000	Alexnet Redes VGG Redes Inception ResNets DenseNets MobileNets NasNets	Tasa de Éxito Distorsión Puntuación CLEVER Transferibilidad	-	Su *et al*. (2018)
3	2	746	Alexnet GooglNet SqueezeNet ShuffleNet ResNets Xception Inception Inception-ResNet VGG DenseNet MobileNet NasNets	Exactitud Precisión Exhaustividad Valor-F	92.62 % 96.20 %	Pham (2020)

Continuación Tabla

Nro.	Clases	Imágenes	Arquitectura CNN	Medida	Exactitud	Referencia
4	10	18 160	Alexnet GooglNet Inception V3 ResNet 18 ResNet 50	Exactitud Precisión Exhaustividad Negativos positivos Valor-F	99.39 %	Maeda *et al*. (2020)
5	38	2000	LeNet AlexNet Inception V3	Exactitud	-	Adit *et al*. (2020)
6	4	200	AlexNet VGG GoogLeNet Inception V3 ResNets	Exactitud Exhaustividad Precisión	- 93.50 %	Syarief y Setiawan (2020)
7	38	54306	Inception V4 VGG 16 ResNets DenseNet	Exactitud	99.75 %	Too *et al*. (2019)
8	2	212	VGG16 VGG19 GoogLeNet-V3 ResNet-50	Exactitud Pérdida	96-25 % *fine-tuned*	Saikia *et al*. (2019)

Continuación Tabla

Nro.	Clases	Imágenes	Arquitectura CNN	Medida	Exactitud	Referencia
10	-	V1 50	AlexNet	Exactitud	96.95 %	Kaur y Gandhi (2020)
	-	V2 74	VGG16	Exhaustividad		
	-	V3 160	VGG19	Precisión		
			GoogLeNet	Negativos positivos		
			Inception V3	Falsos positivos		
			ResNet 50	Valor-F		
			ResNet 101	MCC		
			InceptionResNet V2	Cohen's kappa		
				Exactitud random		
11	12	5539	Personalizado	Exactitud Exhaustividad Precisión Valor-F	94.38 %	Elnemr (2019)

CAPÍTULO II

PLANTEAMIENTO DEL PROBLEMA

El capítulo desarrolla el problema que dio origen al presente trabajo de investigación. Para ello, se describe sobre la identificación y enunciado del problema, justificación, los objetivos de la investigación y la hipótesis de estudio.

2.1 Identificación del problema

El aprovechamiento de recursos maderables es una de las actividades más antiguas del mundo, su importancia, no solamente se limita al uso doméstico o aporte en el desarrollo de un país, sino que tiene un impacto ambiental mundial. Por consiguiente, una incorrecta gestión de este recurso lleva a consecuencias irreversibles como el cambio climático o la extinción de especies (Valqui *et al*., 2016). Así pues, el aprovechamiento de los recursos forestales de manera sostenible se volvió en foco de discusión en los últimos veinte años (Chu *et al*., 1994).

En este sentido, las normativas sobre el manejo y aprovechamiento forestal fueron modificados en el Perú, donde a fin de promover y regular el manejo de los recursos forestales se introdujo el uso de un plan de manejo (Cossío *et al*., 2014). Del mismo modo, desde el 2012 se establecieron procedimientos y criterios técnicos para la evaluación de los individuos forestales (OSINFOR, 2017), sin embargo, según el informe de la Superintendencia Nacional de Aduanas y de Administración Tributaria (SUNAT), el ingreso de la silvicultura y extracción de madera en 2015 fue de 28.3 millones de dólares americanos; a pesar de existir un total de 3 556 empresas registradas dedicadas a dicha actividad (ITP/CITEmadera, 2018). Además, estudios refieren que se perdieron entre 8.9 y 10.5

millones de hectáreas de bosques peruanos (Valqui *et al*., 2016).

En el 2017, distintos representantes de las autoridades forestales peruanas y otras instituciones relacionadas se reunieron para evaluar el avance y la metodología aplicada en el protocolo estándar de evaluación de individuos maderables (OSINFOR, 2017). En dicha reunión, un tema de discusión central fue el problema respecto a la identificación de especies forestales maderables a nivel científico, ahondando en el problema y basándose en el protocolo para la evaluación de los especímenes maderables (OSINFOR, 2012), se observó lo siguiente: **a)** el encargado de la identificación de especies forestales no es necesariamente un especialista; **b)** el protocolo establece que se puede contar con el apoyo de un *matero*, que, a su vez, puede hacer uso de un manual dendrológico. La clasificación realizada por el matero, por lo general, se realiza a partir del fuste y se da a nivel de nombre común; y **c)** si aún existe dificultad para la identificación de la especie, se procede a obtener una muestra botánica y se sigue con el procedimiento respectivo en un laboratorio, lo cual convierte el proceso en tedioso y lento puesto que los especialistas tienen una disponibilidad limitada. Por otro lado, según el especialista Ricardo Zárate, Investigador del Instituto de Investigaciones de la Amazonía Peruana (IIAP), se tiene una exactitud del 70 % - 90 % en la clasificación de las especies forestales maderables.

Desde el punto de vista económico, una incorrecta clasificación afecta negativamente los ingresos de un país debido a que la diferencia de precios entre especies forestales maderables es significativa. Por ejemplo, en el mercado peruano, las especies *Virola ducke* y *Virola pavoris* comparten el nombre común “cumala”, ambas especies son vendidas al precio de S./ 2.20–3.60 (CNF *et al*., 2016), por lo que se da una posible sobre explotación y afectación de algunas poblaciones con denominación no correspondiente a su taxonomía. Es más, una incorrecta clasificación de las especies forestales maderables, o una combinación de distintas anomalías encubre el lavado de activos (ITP/CITEmadera, 2018).

Es en este punto en el que la visión artificial y las técnicas de aprendizaje profundo intervienen como una propuesta tecnológicamente atractiva. La Subsección 1.2.1 demostró que el uso de algoritmos de aprendizaje profundo en tareas de identificación y clasificación automatizada aplicada a distintas floras no tropicales obtienen resultados con una exactitud entre 91.78 % - 99.90 %. Por otro lado, la Subsección 1.2.2 refiere que los es-

tudios orientados a la clasificación automatizada de especies forestales de la Amazonia presentan distintas oportunidades de estudio focalizada como la creación de *datasets*, y estudios o experimentos con nuevos algoritmos.

Finalmente, se resaltan dos necesidades: **primero**, la creación de nuevos *datasets* pues las especies tropicales poseen características peliculares gracias a su ubicación lo que las hace distintas a otras especies alrededor del mundo; **segundo**, se necesitan realizar mayor número de experimentos y/o estudios con algoritmos para los *datasets* existentes ya que, a la fecha, son los humanos quienes obtienen resultados más exactos en comparación a las máquinas (Bonnet *et al*., 2018; Goëau *et al*., 2019).

2.2 Enunciados del problema

Por lo mencionado, se realizó la siguiente formulación del problema: ¿Qué tipo de red neuronal convolucional obtiene mejores resultados en la clasificación de especies forestales maderables en la Amazonia peruana?

2.3 Justificación

La clasificación correcta de las especies forestales influye positivamente, no solo en la economía, sino también a otros dominios como la biodiversidad, agricultura, salud o a los estudios académicos. Asimismo, la SUNAT señaló qué mediante una identificación correcta del tipo de madera se podría reducir significativamente la exportación de madera protegida bajo una denominación comercial (ITP/CITEmadera, 2018). Por ende, según autoridades forestales y otras instituciones relacionadas es prioritario la identificación de especies en estado crítico por ser de interés nacional proteger y aprovechar de manera sostenible los recursos naturales peruanos (OSINFOR, 2017).

El trabajo se enfocó en la clasificación de especies forestales maderables mediante el uso de la visión artificial en conjunto a técnicas de aprendizaje profundo. Los precedentes en trabajos similares demuestran que existe una mejora respecto a la exactitud en la tarea de clasificación automática (Azlah *et al*., 2019). Del mismo modo, la clasificación automatizada de especies forestales apoya a las personas, en especial a aquellas que carecen de experiencia en la identificación de especies. Al respecto, el representante de Imaza,

Bagua, Amazonas-Perú, indicó que la identificación de especies debe ser simplificada en cuanto a los procedimientos y requisitos que posee (OSINFOR, 2017). Por lo tanto, se requiere de una herramienta simple y confiable para la clasificación de las especies forestales maderables.

De esta manera con la presente investigación se pretende dar una posible solución al problema de una manera eficiente, automática y no destructiva considerando trabajos de vanguardia (Goëau *et al*., 2019; Tan *et al*., 2018; Thanh *et al*., 2018) y manuales dendrológicos existentes (Castillo, 2010; OSINFOR, 2018). Como utilidad metodológica, permite conocer el proceso y técnicas utilizadas en la clasificación de especies forestales maderables utilizando arquitecturas de CNNs, como el preprocesamiento, modelamiento y evaluación en modelos con aprendizaje por transferencia. Por último, resaltar el aporte principal en la comunidad científica creando un *dataset* de imágenes hojas correspondientes a 10 especies forestales maderables de la Amazonia peruana en peligro de extinción y con importancia económica.

2.4 Objetivos

2.4.1 Objetivo general

Determinar el tipo de red neuronal convolucional que obtenga mejores resultados en la clasificación de especies forestales maderables en la Amazonia peruana.

2.4.2 Objetivos específicos

- Conformar un conjunto de datos significativo de imágenes de hojas de especies forestales maderables de la Amazonia peruana.
- Implementar las redes neuronales convolucionales AlexNet, VGG-19, Inception V3 y ResNet-101 para la clasificación de especies forestales maderables utilizando el conjunto de datos conformado, bajo criterios de entrenamiento apropiados.
- Establecer métricas de evaluación apropiadas para redes neuronales convolucionales dedicadas a la clasificación de especies forestales maderables en la Amazonia peruana.

2.5 Hipótesis

La red neuronal convolucional pre-entrenada con aprendizaje por transferencia ResNet-101 obtiene mejores resultados de asertividad en la clasificación de especies forestales maderables de la Amazonia peruana.

CAPÍTULO III

MATERIALES Y MÉTODOS

3.1 Lugar de estudio

La investigación se llevó a cabo en la Reserva Nacional Allpahuayo Mishana ubicada en la provincia de Maynas en el departamento de Loreto a 23 km al sur de Iquitos. Creada mediante Decreto Supremo Nro. $002-2004-AG$, el 16 de Enero del 2004, sobre un área de 58,069.9 hectáreas cuyo objetivo es conservar la biodiversidad y el hábitat de los bosques de arena blanca pertenecientes a la Ecorregión Napo, que a su vez forman parte de los bosques inundables por aguas negras de la cuenca del río Nanay, donde el clima es ecuatorial, lluvioso y tropical con una temperatura media anual de 26.7^{o}C aproximadamente, y una precipitación de 2,616.12 milímetros en promedio.

La presencia de arena blanca hace que el ecosistema de la Reserva Nacional Allpahuayo Mishana adquiera características particulares (como la heterogeneidad de hábitats) garantizando la existencia de diversas especies forestales, formando, así, la concentración más representativa del Amazonas peruano debido a que posee diversidad en sus especies (Álvarez, 2007). Además, la superficie de la Reserva Nacional Allpahuayo Mishana está conformada por predios públicos y privados. El presente trabajo construyó un *dataset* de 10 especies forestales maderables que germinan en los predios correspondientes a la concesionaria **Green Gold Forestry S.A.C.**, concesión con fines maderables con sede en Iquitos, y **la Estación Biológica José Álvarez Alonso** (Centro de Investigaciones Allpahuayo), predio del IIAP donde se pueden realizar actividades de investigación, cursos de campo, y actividades de conservación y turismo.

3.2 Población

La Amazonia es conocida por su gran biodiversidad, y diversas especies que la habitan aún no son han sido estudiadas por la ciencia al ser un área escasamente explorada por el hombre (Góes, 2019), a la fecha se sabe que la Amazonia tiene más de 15,000 especies de árboles, de los cuales se estima que 6,350 pertenecen a la Amazonia peruana (Farfan-Rios *et al*., 2015; Ter *et al*., 2020). La presente investigación se enfocó en especies ya conocidas y estudiadas en territorio peruano. La población consistió en el total de 275 especies forestales conocidas en la Amazonia peruana, los cuales fueron publicados por SERFOR mediante la «Resolución de Dirección Ejecutiva Nro. 230-2019-MINAGRI-SEFOR-DE», 2019.

3.3 Muestra

Para entrenar una red neuronal se requiere de un gran número de datos por especie (Davies, 2017); y porque el proceso de construcción de un conjunto de datos o *dataset* de esta naturaleza requiere de una coordinación previa con expertos y autoridades ambientales; por esta razón se estableció que el tipo de muestreo sea probabilístico y por conveniencia. Asimismo, en reuniones realizadas entre el equipo de trabajo, especialistas del IIAP e independientes, se estableció una lista enfocada en las principales especies en peligro de extinción (tala ilegal) con importancia económica y ecológica en el contexto peruano. Por lo cual, se resaltó la importancia de trabajar con 10 especies que requieren pronta atención.

El tamaño de la muestra consiste en las 10 especies forestales maderables seleccionadas que poseen importancia económica y están en peligro de extinción pertenecientes a la Amazonia peruana, cuyo detalle respecto a su nombre científico, nombre común y nombre comercial se pueden observar en la Tabla 6 donde se ordena alfabéticamente por su nombre científico.

Tabla 6
Muestra: Lista de las especies forestales maderables

N^{o}	Nombre científico	Nombre común	Nombre comercial
1	*Aniba rosaeodora ducke*	Moena amarilla, palo de rosa, palo rosa	Moena amarilla
2	*Cedrela odorata*	Cedro, cedro blanco, cedro colorado, cedro de altura, cedro del bajo, cedro de restinga, cedro rojo, kanu, seetrú	Cedro
3	*Cedrelinga cateniformis ducke*	Achapo blanco, achuapo, aguano, aguano masha, cedro macho,cedro masha, cedro rana, huaira caspi, huayra caspi, pino peruano, tapacho blanco, tornillo, tornillo rosado, tsaik	Tornillo
4	*Dipteryx micrantha*	Charapilla, cumarú, kumarut, shihuahuaco, shihuahuaco hoja pequeña	Shihuahuaco
5	*Otoba glycycarpa*	Aguanillo, banderilla, caobilla, cumala, cumala colorada, favorito	Aguanillo
6	*Otoba parvifolia*	Aguanillo, banderilla, caobilla, cumala, cumala colorada, favorito, kumala, mamilla, pintana negra.	Aguanillo
7	*Simarouba amara*	Hualaja, marupa	Marupa
8	*Swietenia macrophylla*	Aguano, caoba, ébano, mara, mahogani, pasich	Caoba
9	*Virola flexuosa*	Caupiri de altura, cumala, cumala blanca, cumala negra, virola amarilla	Cumala
10	*Virola pavonis*	Aguano cumala, caupiri del bajo, cumala, cumala amarilla, cumala blanca, cumala caupiri, caupiri del bajo, caupiri del varillal, cumala zancuda.	Cumala

3.4 Método de Investigación

La presente investigación es comparativa porque se buscó confrontar el comportamiento de los grupos observados en la clasificación de especies forestales maderables de la Amazonia peruana. De acuerdo con las características del problema, objetivos e hipótesis es experimental, ya que las variables son manipuladas para medir y comparar sus efectos entre sí, este método se aplicó con el propósito de generar los resultados y alcanzar las conclusiones de la investigación.

3.5 Descripción detallada de métodos por objetivos específicos

3.5.1 *Dataset*

La construcción del *dataset* denominado Peruvian Amazon Forestry Dataset para la clasificación de especies forestales maderables se basó dos criterios sucesivos. Primero, se buscó el método más accesible y amigable con el medio ambiente; descartando la clasificación a través del fuste o tronco ya que recurre a la tala del árbol, por lo que se sugirió el uso de otras características naturales como el fruto, la flor y la hoja. Segundo, a juzgar por los resultados de trabajos similares de clasificación de especies, pero con distinta flora, se prefirió el uso de la hoja del árbol sobre los demás órganos ya que la hoja está presente durante todas las etapas de desarrollo y estaciones (Ellis, 2009).

Peruvian Amazon Forestry Dataset[1] es un *open Dataset*, es decir, es de libre acceso y consiste en $59,441$ imágenes de hojas correspondiente a 10 especies forestales recolectadas de la Reserva Nacional Allpahuayo Mishana, Perú. A continuación, se detallan aspectos sobre el protocolo de adquisición de muestras y la captura de imágenes (fotografías).

Protocolo de adquisición

Las imágenes fueron recolectas, etiquetadas, organizadas y capturadas por investigadores del IIAP y botánicos especializados. La Figura 20 muestra a un especia-

[1] teledeteccion.iiap.gob.pe/

lista del equipo de trabajo recolectando la muestra de un espécimen.

Figura 20. Especialista recolectando la muestra de un espécimen

Para entrenar modelos de aprendizaje profundo se requieren de una gran cantidad de datos, por lo que las muestras de imágenes fueron recolectadas de diversos especímenes que germinan en los predios del IIAP y de la concesionaria Green Golden Forestry S.A.C.. El intervalo de tiempo para este proceso se dio en 5 diferentes excursiones con una duración aproximada de 30 días cada una. Debido

a las condiciones climatológicas y proceso natural de oxidación de las hojas, el proceso de adquisición de muestras e imágenes se fijó por jornada.

Las jornadas de excursión fueron distintas entre sí variando por factores como el clima y la iluminación, los cuales afectaron directamente con la captura de las fotografías. Sin embargo, todas las imágenes fueron tomadas con un fondo oscuro como se ve en la Figura 21 y siguiendo el mismo protocolo que consistió en:

- Identificación de espécimen en laboratorio por un especialista
- Recolección aleatoria de hojas en sus distintos ciclos de vida
- Digitalización de la fecha de captura de la imagen
- Digitalización masiva de imágenes de hoja bajo un fondo oscuro (color negro o morado)

Figura 21. Escenario preparado para la captura de imágenes de hojas

Las capturas de las imágenes de las hojas fueron realizadas por 6 distintos tipos de cámaras comerciales cuyas características se muestran en la Tabla 7. La Figura 22 presenta un ejemplo de una imagen digital por cada especie forestal maderable capturada en diferentes excursiones y/o tipo de cámara.

Tabla 7
Especificaciones de las cámaras

Código	Modelo de cámara	MPx	Apertura	Resolución	Formato
DC	Nikon D3500	24.2	f/1.5	6000×4000	JPEG
CP1	SM-A705MN	32	f/1.7	4032×3024	JPEG
CP2	SM-A105M	13	f/1.9	4128×3096	JPEG
CP3	SM-A305G	16	f/2.0	4608×2128	JPEG
CP4	FIG-LX3	13	f/2.2	4160×3120	JPEG
CP5	iPhone 6	8	f/2.2	3264×2448	JPEG

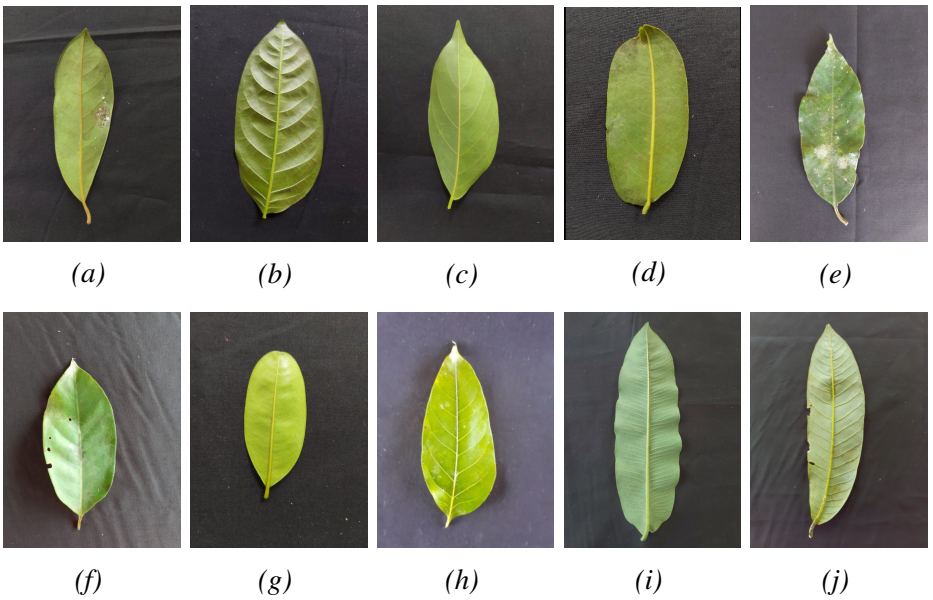


Figura 22. Especies que conforman el *Peruvian Amazon Forestry Dataset*:

(a) *Aniba rosaeodora*. (b) *Cedrela odorata*. (c) *Cedrelinga cateniformis*. (d) *Dipteryx micrantha*. (e)*Otoba glycycarpa*. (f) *Otoba parvifolia*. (g) *Simaruba amara*. (h) *Swietenia macrophylla*. (i) *Virola flexuosa*. (j) *Virola pavonis*.

El protocolo de adquisición tuvo como fin el garantizar la variabilidad de los datos y reducir el riesgo del *overfitting*. Por ejemplo, la Figura 23 muestra la captura visual de las 6 distintas cámaras en las especies *Otoba glycycarpa* y *Cedrela odorata*. Del mismo modo, el tratamiento previo al procesamiento de las imágenes tiene como fin hacer el procedimiento más rápido y evitar posibles problemas de sobreajuste.

(a) *(b)* *(c)* *(d)* *(e)* *(f)*

Figura 23. Ejemplos de capturas de las diferentes cámaras:

(a) SM-A105M. (b) SM-A305G. (c) SM-A705MN. (d)FIG-LX3. (e) Nikon D3500. (f) iPhone 6. De las especies *Otoba glycycarpa* (Arriba) *Cedrela odorata* (Abajo)

Normalización de las imágenes

Al normalizar las imágenes se aplicó un preprocesamiento general para los cuatro modelos pre-entrenados. Como se puede observar en la Figura 23 las imágenes que conforman el *dataset* original son de distintos tamaños. Por lo que se redimensionó el tamaño de las imágenes a 512×512 píxeles en su totalidad, la Figura 24 presenta un ejemplo de una imagen redimensionada a 512×512 píxeles. Luego se experimentó con dos tipos de ejemplos: con fondo removido y manteniendo el fondo original. En ambos casos, se verificó que cada píxel de las imágenes estuvieran entre el valor positivo de $[0, 1]$. Esto con el objetivo de optimizar el tiempo computacional del algoritmo.

Figura 24. Ejemplo imagen redimensionada a 512×512 píxeles.

Asimismo, es importante resaltar que características originales de las imágenes como la iluminación o distancia al eje de enfoque fueron preservados ya que aportan variabilidad en los datos de entrenamiento. La diversidad de datos en el *dataset* previene casos de *overfitting*. Así como, el ruido se consideró como una característica natural necesaria por lo que no se aplicó ningún tipo de método de corrección. Por lo expuesto, se esperó que cada red aprendiera a clasificar especies.

Preprocesamiento de las imágenes

Primero, habiendo dimensionando y seleccionado la imagen de entrada i se aplicó la función de Desenfoque Gaussiano (*Gaussian Blur operation*, *GB*) para que la imagen se vea más suave. El desenfoque Gaussiano mezcla ligeramente los colores de píxeles vecinos afectando directamente a los bordes.

Como siguiente paso, a fin de evaluar los colores de la imagen cambiamos el color RGB *(Red, Green y Blue)* de la imagen al espacio de color Lab, donde L es la luminosidad, el componente a (va de rojo a verde) y el componente b (va de azul a amarillo). El espacio de color Lab busca producir en las imágenes una perspectiva visual similar a la percepción humana. Para resaltar las partes nítidas de la imagen se aplicó la ecuación 1 que binarizó la imagen de salida i^s aplicando la ecualización del histograma (Pizer *et al.*, 1987). Todo esto maximizó el contraste de la imagen en el espacio de color $L \times a \times b$.

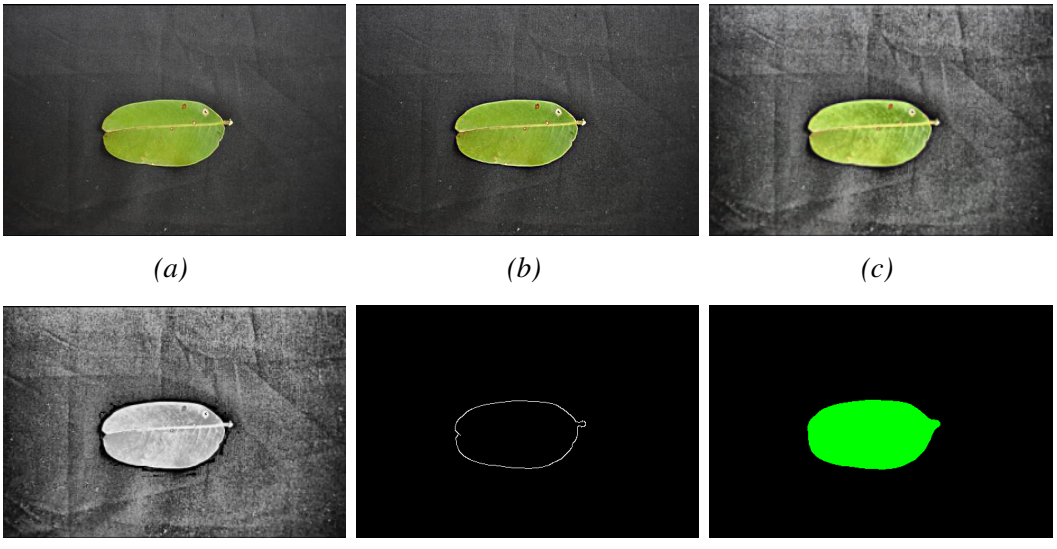
Respecto a lo obtenido hasta este punto del proceso, el color verde de la hoja es predominante al color oscuro (azul oscuro y negro) del fondo. A continuación, se prosiguió con la eliminación parcial del fondo usando la máscara descrita en la ecuación 2.

$$i^s = L \times i - (L-1) \times GB(i) \tag{1}$$

$$Mascara(x,y) = \begin{cases} 1, & \text{if } G(x,y) > B(x,y) \\ 0, & \text{other} \end{cases} \tag{2}$$

En vista que el canal *G* marcó el borde de la hoja, se aplicó un filtro bilateral para difuminar el canal G a fin de preservar el ruido y los bordes más relevantes utilizando la diferencia de los píxeles para determinar el coeficiente del filtro Ω. El algoritmo de Otsu (Fang *et al*., 2009) tiene como objetivo calcular el valor óptimo de Omega (Ω), que se emplea para calcular el valor de dos umbrales (θ_{Bajo} y θ_{Alto}). Mientras que cada valor dentro de cada segmento se hace lo más pequeña posible, la dispersión fuera del segmento se hace lo más alta posible. La ecuación 3 presenta el cálculo en ambos casos según su relación con los bordes vecinos. Finalmente, el algoritmo eliminó al objeto con mayor masa para obtener la imagen segmentada. La Figura 25 muestra gráficamente cada uno de los pasos descritos sobre la eliminación del fondo de la hoja.

$$\theta_{Bajo} = \frac{\Omega - 29{,}75}{1{,}41}$$
$$\theta_{Alto} = \theta_{Bajo} \times 6 \tag{3}$$



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

(d) *(e)* *(f)*

Figura 25. Proceso de la eliminación del fondo:

(a) Ingreso de imagen. (b) Suavizado (nitidez de la imagen). (c) Operación Luminance. (d) Aumento de contraste. (e) Detección de bordes. (f) Imagen segmentada.

3.5.2 Arquitecturas CNN

De acuerdo con lo expuesto en la Subsección 1.1.5, para poder trabajar con una CNN se requiere una gran cantidad de datos para su entrenamiento. Las técnicas para superar este percance consisten en recurrir a métodos como el aumento de datos o la transferencia de aprendizaje.

Tabla 8
Arquitecturas AlexNet, VGG-19, Inception V3 y ResNet-101

Red	**Capas**	**Parámetros**	**Error Top-5**	**Imagen**
AlexNet	8	60 M	15.3 %	227×277
VGG-19	19	144 M	7.5 %	224×244
Inception V3	48	23.8 M	5.6 %	299×299
ResNet-101	101	44.5 M	4.6 %	224×224

3.5.3 Métricas

Las medidas elegidas nos permiten ponderar el desempeño de los modelos pre-entrenados propuestos en términos de exactitud, precisión, exhaustividad y Valor-F1. La matriz de confusión y curva de aprendizaje permiten visualizar y analizar el proceso de aprendizaje. Del mismo modo, la interpretabilidad intrínseca de los modelos permite comprender sobre como aprende el modelo.

Métricas de desempeño

La precisión de clasificación de la exactitud y exhaustividad utiliza los términos: verdadero positivo (TP), verdadero negativo (TN), falso negativo (FN) y falso positivo (FP). TP es el número de imágenes etiquetadas correctamente según su especie, TN, FP y FN es el número de imágenes etiquetadas con otra especie no correspondientes al suyo. Matemáticamente se dan como:

$$Exactitud = \frac{TN + TP}{TP + FP + TN + FN} \times 100\,\%$$

$$Exhaustividad = \frac{TP}{TP+FN} \times 100\,\%$$

Las métricas mencionadas anteriormente son efectivas únicamente en caso el *dataset* sea equilibrado. Los *dataset* desequilibrados requieren de una validación adicional utilizando métricas de rendimiento como la precisión y el Valor-F.

$$Precision = \frac{TP}{TP+FP} \times 100\,\%$$

$$Valor - F1 = 2 \times \frac{(Precision \times exhaustividad)}{precision + exhaustividad}$$

El presente trabajo utilizó, adicionalmente, el cálculo del Macro-F1 y Micro-F1. El **Macro-F1** consiste en el cálculo de cada una de las clases individualmente sumadas y dividida por el total de clases, como se da en la exactitud. El **Micro-F1**, también llamado promedio Micro-F1, es calculado a partir del total de FP, FP y FN resultando un cálculo general, es decir, no se considera cada clase de manera individual.

$$Micro - F1 = \frac{(Precision \times exhaustividad)}{precision + exhaustividad}$$

En el Micro-F1 todas las métricas utilizadas llegan a tener el mismo resultado:

$$Precision = exhaustividad = Micro - F1 = exactitud$$

Matriz de confusión

Las métricas de desempeño son posibles gracias a los índices indicados en la matriz de confusión generada. Como herramienta de *deep learning* ilustra la probabilidad de coincidencia por especie en problemas de clasificación multiclase.

En Python, la matriz de confusión se puede obtener usando la función *confusion_matrix()* que es parte de la librería sklearn (Pedregosa *et al*., 2011). Esta función se importa a Python usando *from sklearn.metrics import confusion_matrix*. También, se debe proporcionar los valores reales y de predicción según el nombre científico de las 10 especies forestales maderables correspondientes al *dataset* utilizado.

Curva de ROC

En el caso de datos desequilibrados, la Curva de ROC (Receiver Operating Characteristic Curve) se utiliza para evaluar el rendimiento del modelo. La representación gráfica del clasificador permite valorar el aprendizaje del modelo y detectar la probabilidad de *overfitting*. En el eje x se traza la tasa de los falsos positivos (FP), en el eje y verdaderos positivos (TP). Los umbrales están definidos en un rango de 0 a 1 esperando que la curva de ROC siempre esté en el cuadrante superior porque representa a un buen clasificador.

En Python, el seguimiento de la curva de aprendizaje a tiempo real es posible gracias a la librería Weights & Biases (Biewald, 2020). La función puede ser utilizada tras registrarse en su sitio web y agregar al *notebook* la función *wandb.login()*. WandB se importa usando *import wandb* e *wandb.init(project='name_test')*. Para extraer el seguimiento de las métricas se utiliza *wandb.log('accuracy': train_acc, 'loss': train_loss)*, los resultados a obtener son visualizados en su sitio web (Ver Anexo **6.**), los reportes obtenidos por Weights & Biases tienen como fin ayudar a optimizar las predicciones.

Interpretabilidad del modelo

Si bien el uso de la interpretabilidad de un modelo como medición formal aún está en proceso. El presente estudio se basó en uno de los enfoques señalados por Li *et al*. (2021). El enfoque utilizado es por **disección de la red**, ya que se tomaron un conjunto de datos etiquetados extraídos del *dataset* con imágenes segmentadas y las imágenes del *dataset* original para ser procesados por un algoritmo que permite ver el mapa de características utilizados por cada modelo.

En este estudio con la interpretabilidad de los modelos, se podrá visualizar las características morfológicas que son utilizadas por cada modelo para poder realizar la clasificación de especies forestales maderables.

3.5.4 Hardware y Software

Hardware

Los experimentos se desarrollaron utilizando datos (imágenes) del *dataset* Peruvian Amazon Forestry Dataset, los cuáles fueron recolectados por 6 distintos tipos de cámaras comerciales (Nikon D3500, SM-A705MN, SM-A305G, FIG-LX3, iPhone 6 y SM-A105M). El *dataset* es gestionado por el supercomputador llamado **Manatí** cuyas especificaciones son: **Nodo principal** (Intel Xeon, 56 núcleos, 64GB RAM), **6 Nodos de procesamiento gráfico** (Intel Xeon, 28 núcleos, NVIDIA TESLA K80, 64GB RAM), y **3 Nodos para el procesamiento gráfico** (Intel Xeon, 28 núcleos, 64GB RAM, 114TB disco duro). Además, para los experimentos de los modelos CNNs pre-entrenados se usó una computadora de escritorio con un procesador Intel Core i9 4.0 GHz, memoria de 32 GB 3000 MHz DDR4 y tarjeta de vídeo NVIDIA Titan RTX.

Software

El software utilizado durante el desarrollo del proyecto se dividió según su necesidad: **herramientas de análisis** como Trello y Overleaf, **herramientas de diseño** draw.io, para la **codificación** Atom y Python, y las **librerías** PyTorch, Scikit-learn y WandB.

- **Trello** utilizado para la organización del proyecto (administración de tareas), se usó en su versión web y en android.
- **Overleaf** usado como gestor de texto el cual se desarrolla en un ambiente LaTeX. Su ventaja frente a otros editores de LaTeX es su disponibilidad en la web lo cual no requiere una instalación previa ni configuración.
- **Atom** como editor de código se utilizó Atom, que fuese desarrollado por GitHub. Además, es de código abierto y disponible en distintos sistemas

operativos, su ventaja está en Github donde su manejo de versiones resulta sencillo.

- **Python** es un lenguaje de programación de código abierto. Gracias a su gran popularidad cuenta con diversos recursos de soporte y desarrollo. Python es conocido por ser un lenguaje multiparadigma ya que soporta programación orientada a objetos.

  En el aprendizaje automático Python fue uno de los primeros lenguajes en ser utilizado para la inteligencia artificial. A la fecha, existen diversos entornos donde se puede codificar con Python. Quizás, los dos espacios más conocidos sean Google Colaboratory y con Microsoft Azure Notebooks donde se nos permite trabajar proyectos de aprendizaje automático con Pytorch en línea.Y gracias, a su creciente comunidad se cuenta con diversas librerías con las cuales trabajar el aprendizaje automático. Algunas de las librerías utilizadas en proyectos de aprendizaje automático son: Scikit, Open CV, Matplotlib, TensorFlow, Pytorch.

- **Pytorch** es una librería de código abierto presentada por Paszke *et al*. (2019) cuyo primer propósito fue que un algoritmo de aprendizaje automático (ML) se desenvuelva tanto en usabilidad como en velocidad. También su fácil depuración y compatibilidad con otras librerías hace que Pytorch sea exitosamente compatible con las configuraciones de una GPU. Como dato curioso Pytorch es utilizada por Facebook como su librería para tareas de aprendizaje automático.

- **Scikit-learn** conocido como SKLearn es una librería popular de Python de código abierto. Presentado por Pedregosa *et al*. (2011) con el objeto de ser usado para construir modelos de aprendizaje profundo. Actualmente, es utilizado como herramienta para el análisis estadístico de predicción de datos, clasificación, *clustering* y regresión.

- **Weights & Biases**, también conocido como WandB o W&B es una plataforma web que permite seguir y versionar los experimentos de aprendizaje profundo rápidamente (Biewald, 2020).

CAPÍTULO IV

RESULTADOS Y DISCUSIÓN

Con base en los objetivos específicos planteados en la investigación, en este capítulo se describe la división del *dataset* conformado, las configuraciones utilizadas en el entrenamiento de los modelos y los resultados obtenidos conforme a las métricas de evaluación utilizadas. Finalmente, se discuten los resultados del análisis comparativo de acuerdo con el objetivo general.

4.1 Distribución del dataset

Se evaluaron y entrenaron modelos de aprendizaje profundo sobre imágenes de hojas de especies forestales maderables de la Amazonia peruana. Para este trabajo de investigación se conformó el conjunto de datos Peruvian Amazon Forestry Dataset con 59,441 imágenes de hojas de 10 especies forestales maderables. Las imágenes fueron digitalizadas por 6 tipos de cámaras, redimensionadas a 512×512 píxeles, normalizadas y segmentadas de acuerdo con lo descrito en la Sección 3.5.1. La Tabla 9 presenta el total de imágenes (ejemplos) digitalizadas y distribuidas según el tipo de cámara por el que fueron capturadas.

La distribución de los datos de Peruvian Amazon Forestry Dataset ha sido establecida bajo el estándar Plinian Core el cual busca la gestión de información sobre especies aplicada a su difusión. La Tabla 10 presenta la distribución del *dataset* según el protocolo Plian Core, donde, los ejemplos para el entrenamiento y validación estuvo conformado por las imágenes digitalizadas por las cámaras DC, CP1, CP2 y CP5 (NIKON D3500, SM-A705MN, SM-A105M y Iphone 6), y el test reúne las capturas de las cámaras CP3 y CP4

(SM-A305G y FIG-LX3). Sobre la división porcentual del *dataset*, la distribución de los datos fue: el 70,89 % para el entrenamiento, un 1,56 % para la validación, y el 27,55 % restante para el test.

Tabla 9
Distribución de las especies forestales maderables por cámara

Especies	Ejemplos						
	DC	**CP1**	**CP2**	**CP3**	**CP4**	**CP5**	**Total**
Aniba rosaeodora	1529	1547	1547	1537	402	—	6562
Cedrela odorata	1302	1302	1304	1303	188	127	5526
Cedrelinga cateniformis	1232	1232	1232	1230	177	176	5279
Dipteryx micrantha	1248	1248	1248	1248	480	340	5812
Otoba glycycarpa	1271	1281	1260	1268	136	322	5538
Otoba parvifolia	1745	1713	1712	1716	385	—	7271
Simarouba amara	980	1216	1216	1210	172	388	5182
Swietenia macrophylla	1564	1586	1568	1572	146	—	6436
Virola flexuosa	1030	1042	1040	1042	190	—	4344
Virola pavonis	1841	1842	1832	1840	136	—	7491
Total	13742	14009	13959	13966	2412	1353	**59441**

Tabla 10
Dataset distribuido según estándar Plian Core

Especies	Ejemplos			
	Entrenamiento	**Validación**	**Test**	**Total**
Aniba rosaeodora	4526	97	1939	6562
Cedrela odorata	3949	86	1491	5526
Cedrelinga cateniformis	3787	85	1407	5279
Dipteryx micrantha	3995	89	1728	5812
Otoba glycycarpa	4038	96	1404	5538
Otoba parvifolia	5065	105	2101	7271
Simarouba amara	3718	82	1382	5182
Swietenia macrophylla	4618	100	1718	6436
Virola flexuosa	3033	79	1232	4344
Virola pavonis	5405	110	1976	7491
Total	42134	929	16378	**59441**

De igual forma, basándose en el protocolo *Plian Core* el *dataset* Peruvian Amazon Forestry Dataset está disponible en la red con acceso libre. Cada una de las imágenes al ser

archivos digitales poseen metadatos que describen sobre la adquisición de la imagen (fecha, tipo de sensor, resolución de la imagen) y los datos taxonómicos de la hoja (nombre científico, nombre común, clasificación taxonómica, reino, clase, sinónimos, tipo de vida, ciclo de vida, y reproducción). Por ejemplo, el Anexo **1.** presenta los metadatos a detalle según el estándar de *Plian Core* para la imagen $virolaFlex_A30_0001.jpg$ que es parte del *dataset* conformado.

4.2 Configuración experimental

Para evaluar el desempeño de los modelos pre-entrenados propuestos, se fijó la misma configuración primaria para todos. El *dataset* fue dividido en tres partes, destinando el $70,89\,\%$ del *dataset* para el entrenamiento, el $1,56\,\%$ para la validación, y el $27,55\,\%$ restante para el test. Los experimentos de los modelos pre-entrenados fueron ejecutados en una PC con las siguientes características: procesador Intel Core i9 de 4.0 GHz, memoria DDR4 de 32 GB a 3000 MHz y NVIDIA Titan RTX y el framework Pytorch 1.3.

4.2.1 Ajustes de las CNNs

La presente investigación utilizó para sus experimentos cuatro arquitecturas pre-entrenadas: AlexNet, VGG-19, Inception V3 y Resnet-101. Las redes con aprendizaje por transferencia se importaron desde Pytorch y luego personalizaron según la naturaleza del *dataset* (Ver Anexos **2.** y **3.**). Para el entrenamiento de cada modelo pre-entrenado se estableció el uso de 16 *mini-batches*, el optimizador ADAM (Kingma & Ba, 2014) y la función de activación ReLU. También, se fijó una tasa de aprendizaje de 1^{e-3} y un máximo de 22 épocas como se ilustra en el Anexo **4.**.

Respecto a los datos de entrada, se manejaron dos grupos a fin de evaluar los modelos pre-entrenados en dos escenarios distintos. Ambos grupos de imágenes fueron duplicados de una en común, donde llamaremos al primer grupo Imágenes no segmentadas o imágenes no preprocesadas (NP) que mantienen su fondo original y el segundo grupo de imágenes segmentadas o imágenes preprocesadas (P) donde se eliminó el fondo.

4.3 Resultados

Para evaluar la efectividad de modelos CNNs pre-entrenados con aprendizaje por transferencia, se consideraron el proceso de entrenamiento de la red, validación y test para evaluar el desempeño cualitativa y cuantitativamente. Se eligieron varias métricas de rendimiento, como son: la exactitud, precisión, exhaustividad y Valor-F1. Para visualizar el desempeño de los algoritmos se emplearon matrices de confusión. Las curvas de aprendizaje ayudan a medir y controlar posibles casos de *overfitting*. De la misma manera, la interpretabilidad intrínseca de los modelos es importante para comprender y mejorar el modelo desde un punto de vista cualitativo (Ver Anexo **5.**).

4.3.1 Resultados cuantitativos

El desempeño de los modelos pre-entrenados en la tarea de clasificación de especies forestales maderables fue evaluado teniendo en cuenta los dos grupos de entradas (imágenes no segmentadas e imágenes segmentadas), los resultados se resumen en diversas tablas y figuras. La Tabla 11 ilustra la comparación de desempeño de todos los modelos pre-entrenados durante el entrenamiento, validación y test utilizando la exactitud como unidad de medida resaltando las mejores puntuaciones.

Tabla 11
Resultados respecto al entrenamiento, validación y test

Modelo	**Entrada**	**Entrenamiento**				**Validación**		**Test**
		Tiempo	**Época**	**Exactitud**	**Error**	**Exactitud**	**Error**	**Exactitud**
AlexNet	No preprocesada	**35m 26s**	19/21	97.30 %	7.78 %	98.11 %	5.92 %	96.61 %
	Preprocesada	53m 12s	20/21	97.21 %	7.95 %	98.49 %	5.94 %	**97.10 %**
VGG-19	No preprocesada	197m 27s	**18/21**	**97.61 %**	6.73 %	**98.87 %**	4.27 %	97.02 %
	Preprocesada	205m 14s	19/21	97.56 %	6.91 %	98.68 %	4.16 %	96.62 %
Inception V3	No preprocesada	105m 28s	18/21	63.69 %	106.55 %	80.15 %	75.27 %	74.11 %
	Preprocesada	120m 45s	19/21	63.73 %	106.28 %	80.53 %	74.70 %	63.45 %
ResNet-101	No preprocesada	198m 56s	19/21	69.33 %	88.91 %	82.42 %	62.59 %	78.22 %
	Preprocesada	175m 15s	21/21	69.86 %	87.20 %	82.04 %	62.91 %	78.23 %

Los datos tabulados indican que el modelo pre-entrenado VGG-19 logró los mejores resultados en el entrenamiento y validación, seguido por AlexNet que obtuvo un me-

jor resultado en el test; desafortunadamente los modelos Inception V3 y ResNet-101 no obtuvieron resultados destacables. El mejor resultado en el entrenamiento fue del 97,61 %, en la validación se obtuvo un 98,87 % como el mejor resultado y en el test el 97,10 % fue el mejor desempeño. También, el valor tiempo óptimo de entrenamiento fue de 35 minutos con 2 segundos, los mejores resultados fueron alcanzados en la 18^{va} época de las 21 establecidas. Además, tomando en cuenta cualquier ejemplo del *dataset* el error o pérdida tiene un efecto directo con la exactitud obtenida, mientras mayor sea la exactitud la pérdida será menor.

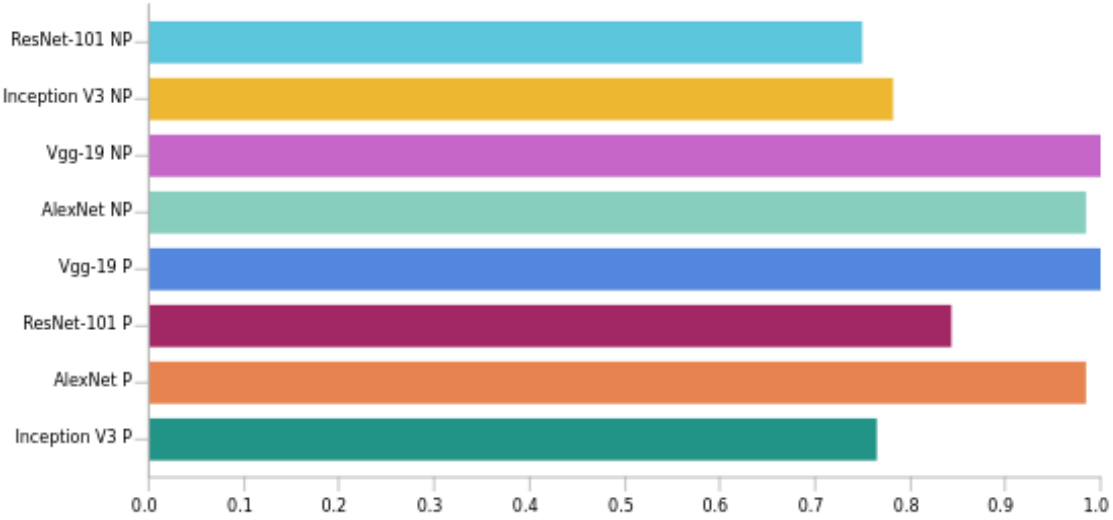


Figura 26. Exactitud cuantitativa alcanzada por cada modelo pre-entrenado

Si bien el valor de la exactitud da una idea clara sobre el desempeño de un modelo pre-entrenado como se observa en la Figura 26 donde los modelos VGG-19 y AlexNet, tanto con entrada No Preprocesada (NP) y Preprocesada (P), obtienen mejores resultados. Evaluar otras métricas como la precisión, exhaustividad y el Valor-F1 indican el beneficio del aprendizaje por transferencia para reducir la probabilidad de *overfitting* y aumentar la probabilidad de coincidencia, los cuales a su vez se ratifican con el análisis de las curvas de aprendizaje y la visualización de las matrices de confusión, respectivamente. La Tabla 12 muestra una comparación de todos los modelos usando todas las métricas establecidas.

Tabla 12
Resultados en términos de exactitud, precisión, exhaustividad y Valor F-1

Modelo	Entrada	Métrica									
		Exactitud	Precisión			Exhaustividad			Valor-F1		
			Micro	Macro	Promedio	Micro	Macro	Promedio	Micro	Macro	Promedio
AlexNet	No preprocesada	**0.97**	**0.97**	**0.97**	**0.97**	**0.97**	**0.97**	**0.97**	**0.97**	**0.97**	**0.97**
	Preprocesada	**0.97**	**0.97**	**0.97**	**0.97**	**0.97**	**0.97**	**0.97**	**0.97**	**0.97**	**0.97**
VGG-19	No preprocesada	**0.97**	**0.97**	**0.97**	**0.97**	**0.97**	**0.97**	**0.97**	**0.97**	**0.97**	**0.97**
	Preprocesada	**0.97**	**0.97**	**0.97**	**0.97**	**0.97**	**0.97**	**0.97**	**0.97**	**0.97**	**0.97**
Inception V3	No preprocesada	0.74	0.74	0.74	0.77	0.74	0.76	0.74	0.74	0.74	0.75
	Preprocesada	0.63	0.63	0.63	0.81	0.63	0.53	0.63	0.63	0.57	0.7
ResNet-101	No preprocesada	0.78	0.78	0.78	0.79	0.78	0.78	0.78	0.78	0.78	0.78
	Preprocesada	0.78	0.78	0.78	0.79	0.78	0.80	0.78	0.78	0.78	0.78

Los valores obtenidos en la exactitud, precisión, exhaustividad y Valor-F1 como medidas de rendimiento y evaluación, ilustran que los modelos VGG-19 y AlexNet obtuvieron un mejor desempeño. Por lo contrario, los modelos Inception V3 y ResNet-101 al tener más capas realizan transformaciones más complejas de las requeridas por la naturaleza del *dataset*. Los resultados concuerdan con lo señalado por Anubha *et al*. (2019) respecto al desempeño sobresaliente de VGG-19 sobre otros modelos pre-entrenados para tareas relacionadas al procesamiento de imágenes de hoja, siendo las imágenes no preprocesadas las que se desenvuelven mejor en todos los modelos. Por lo tanto, los siguientes experimentos se presentaron únicamente analizando aquellos casos donde la entrada fue de imágenes no segementadas (NP).

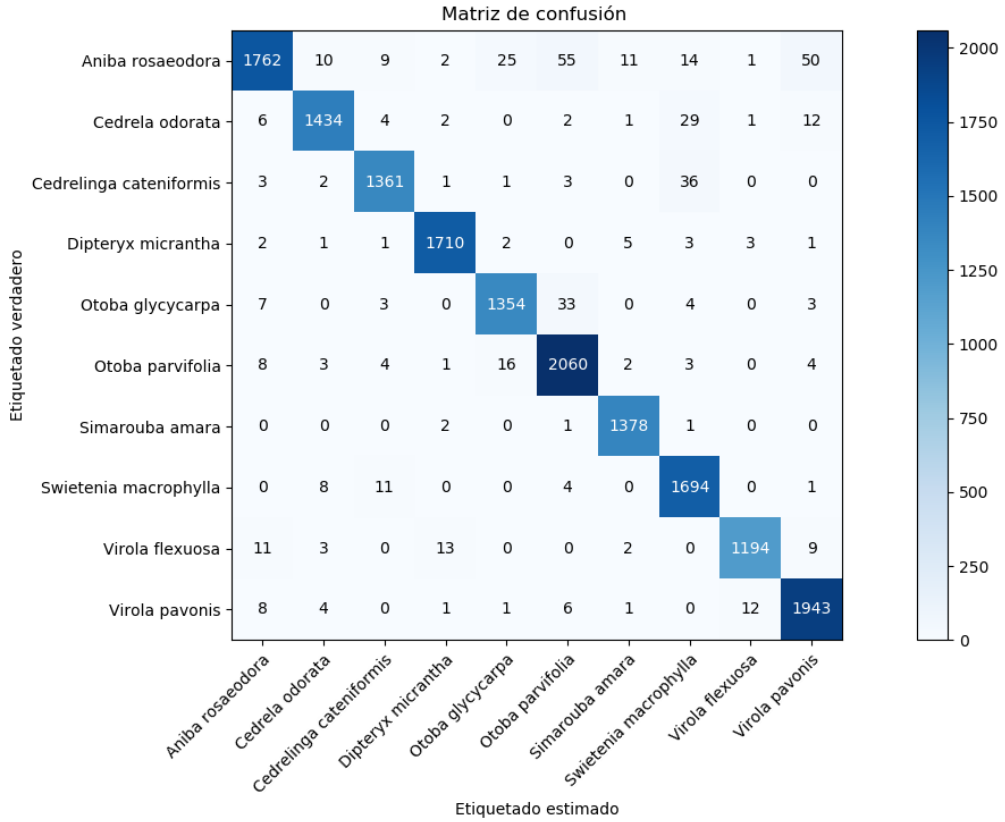
La Tabla 13 presenta el desempeño de cada modelo para cada especie utilizando una entrada no preprocesada. Por lo cual, se analizó los mejores resultados en cada modelo pre-entrenado teniendo en cuenta la precisión, exahustividad y valor F-1 como medidas cuantitativas resaltando de negrita el valor con mejor resultado para reconocer la especie que obtuvo mayor asertividad. La especie *Dipteryx micrantha* obtuvo 8 resultados altos de los 4 modelos pre-entrenados siendo los valores igual a 0.99, 0.99, 0.99, 0.99, 0.99, 0.93, 0.88 y 0.95. Seguido de la especie *Simarouba amara* con 4 resultados altos 0.99, 1.00, 0.99 y 0,96. Gracias a este resultado, podemos confirmar que los modelos pre-entrenados VGG-19 y AlexNet son adecuados para nuestro contexto.

Tabla 13
Resultados de los modelos para cada especie forestal maderable

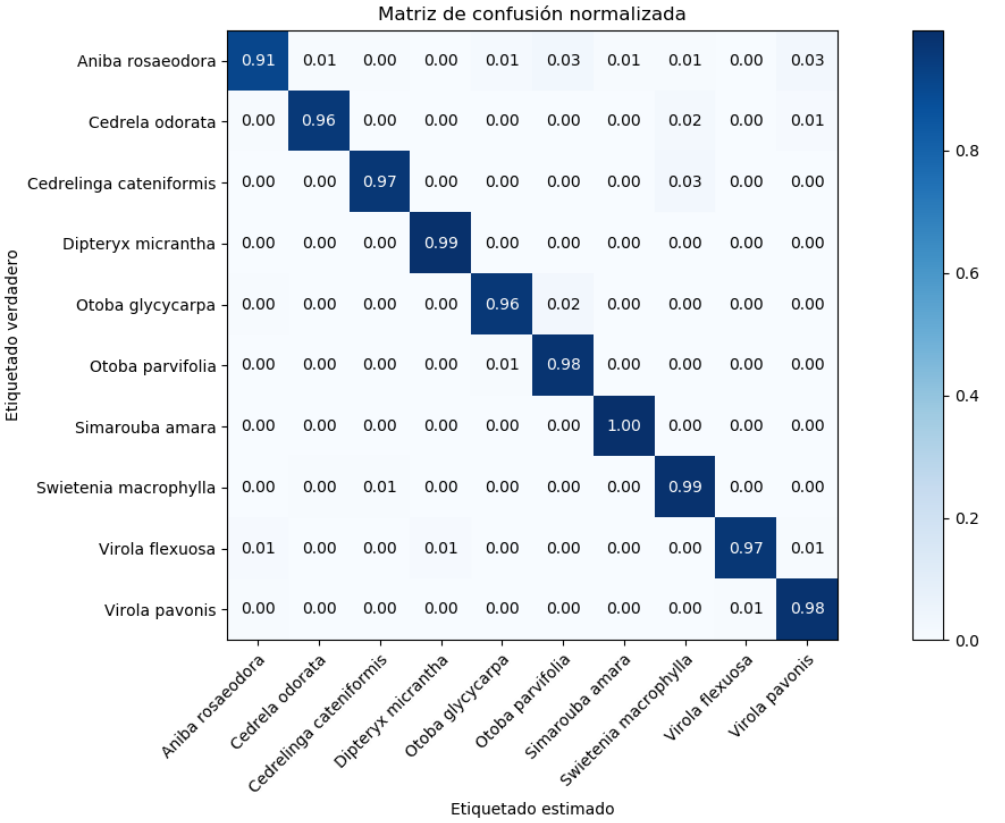
Especie	AlexNet			VGG-19			Inception V3			ResNet-101		
	Precisión	Exhaus-tividad	Valor-F1	Precisión	Exhaus-tividad	Valor-F1	Precisión	Exhaus-tividad	Valor-F1	Precisión	Exhaus-tividad	Valor-F1
Aniba rosaeodora	0.94	0.97	0.96	0.91	0.98	0.94	0.39	0.82	0.53	0.62	0.76	0.68
Cedrela odorata	0.94	0.98	0.96	0.93	0.98	0.96	0.65	0.77	0.71	0.85	0.69	0.76
Cedrelinga cateniformis	0.98	0.97	0.97	0.96	0.96	0.96	0.58	0.76	0.66	0.68	0.73	0.70
Dipteryx micrantha	**0.99**	**0.99**	**0.99**	0.99	**0.99**	**0.99**	0.83	**0.93**	**0.88**	0.80	**0.95**	0.87
Otoba glycycarpa	0.97	0.95	0.96	0.98	0.94	0.96	0.66	0.70	0.68	0.71	0.90	0.79
Otoba parvifolia	0.97	0.97	0.97	0.98	0.94	0.96	0.82	0.62	0.71	**0.90**	0.73	0.81
Simarouba amara	**0.99**	0.98	0.98	**1.00**	0.98	**0.99**	**0.96**	0.69	0.80	0.82	0.82	0.82
Swietenia macrophylla	0.98	0.95	0.96	0.96	0.94	0.95	0.88	0.66	0.75	0.75	0.70	0.72
Virola flexuosa	0.98	0.98	0.98	0.98	**0.99**	0.98	0.81	0.88	0.84	0.81	**0.95**	**0.88**
Virola pavonis	0.98	0.97	0.97	0.98	0.97	0.98	0.80	0.74	0.77	0.86	0.77	0.81

Habiendo presentado los resultados cuantitativos en términos de las métricas establecidas para el presente trabajo se procedió a realizar un análisis mucho más profundo con los dos modelos con mejor desempeño VGG-19 y AlexNet con la finalidad de observar su probabilidad de coincidencia por especie. La Figura 27 muestra las matrices de confusión del modelo VGG-19 con entradas no preprocesadas, en 27a la matriz presenta los resultados en cantidades enteras y 27b presenta la matriz con confusión normalizada para tener los resultados cuantitativos aptos para realizar una comparación.

Del mismo modo, la Figura 28 muestra las matrices de confusión en función a las entradas no preprocesadas para el modelo pre-entrenado AlexNet, en 28a se presenta el desempeño del modelo en cantidades enteras, y 27b presenta la matriz de confusión normalizada. Las matrices de confusión grafican las predicciones acertadas para todas las especies forestales maderables.

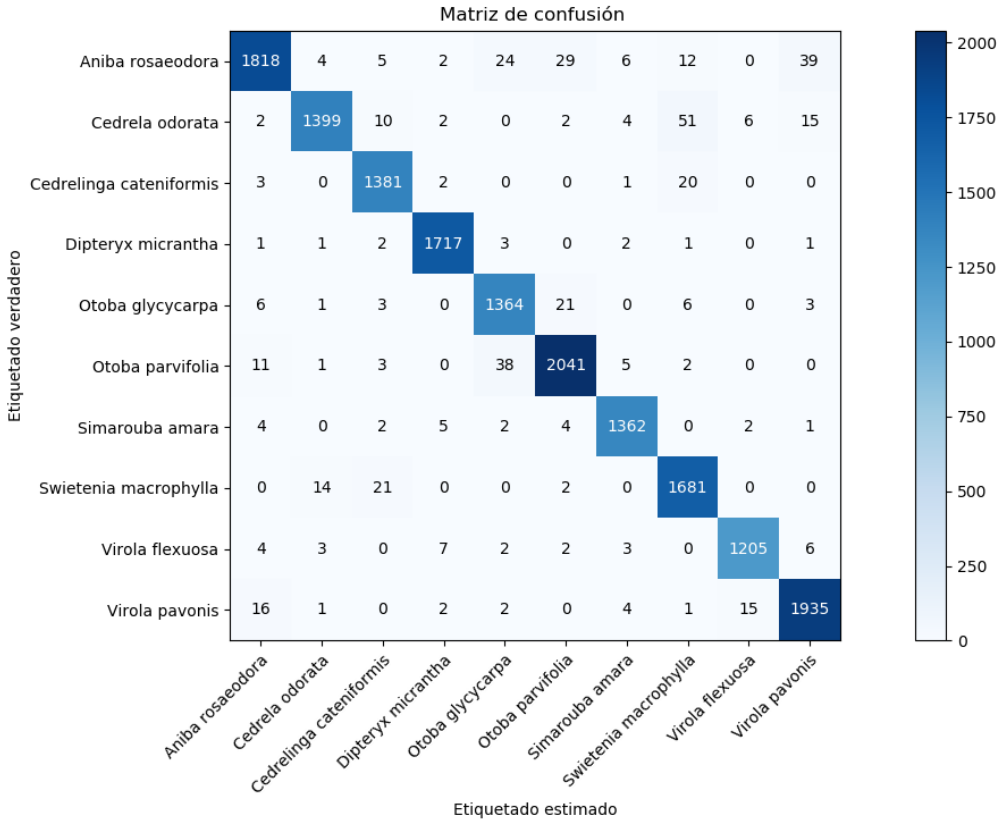


(a) Matriz de confusión para el modelo VGG-19

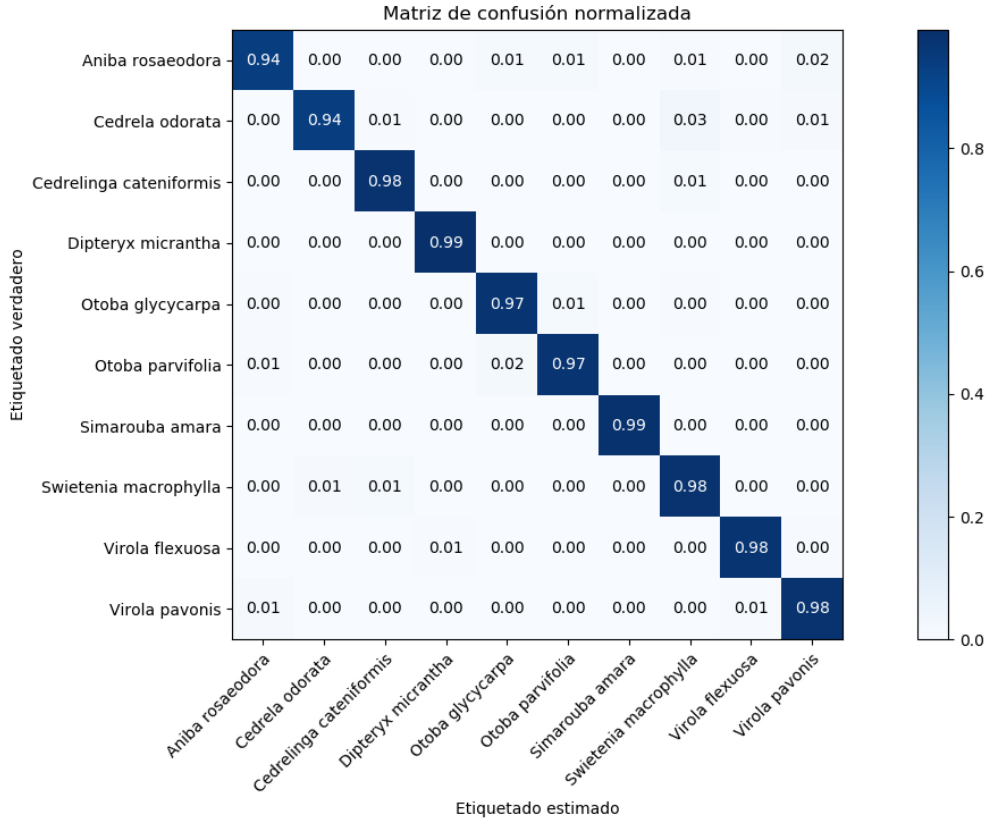


(b) Matriz de confusión normalizada para el modelo VGG-19

Figura 27. Matrices de confusión para el modelo VGG-19



(a) Matriz de confusión para el modelo AlexNet

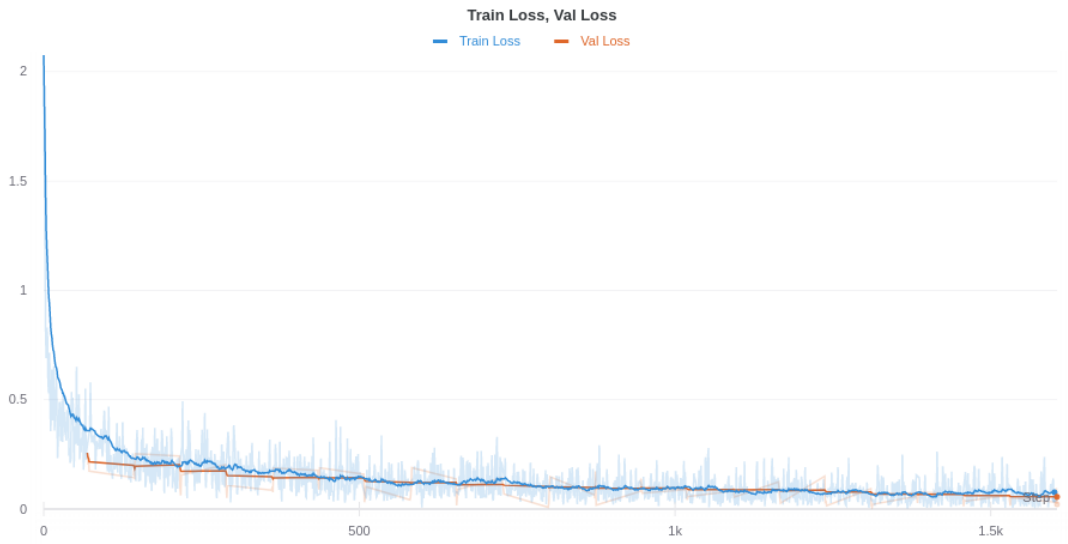


(b) Matriz de confusión normalizada para el modelo AlexNet

Figura 28. Matrices de confusión para el modelo AlexNet

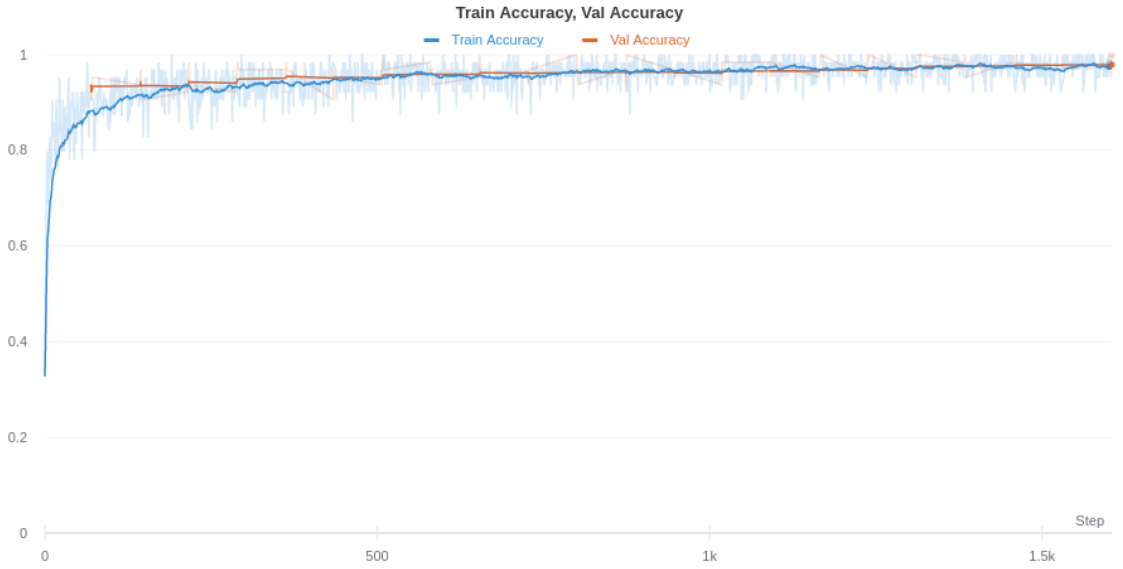
Gracias a los gráficos obtenidos por las matrices de confusión de ambos modelos se confirmó que, por lo regular, para todas las especies se realizaron predicciones acertadas. Sin embargo, a diferencia de la Tabla 13 en las matrices de confusión la especie *Simarouba amara* obtiene un mejor desempeño con 1.00 seguido por la especie *Dipteryx micrantha* con 0.99. A pesar que los resultados presentan una ligera variación respecto a la ponderación en ambas evaluaciones los dos mejores resultados son coherentes entre sí, posicionando a las especies *Simarouba amara* y *Dipteryx micrantha* como las especies forestales maderables con mayor probabilidad de una clasificación acertada.

Los resultados anteriores, también, muestran una ligera superioridad del modelo VGG-19 ante AlexNet. Con el objeto de fortalecer lo señalado y detectar la probabilidad de *overfitting* en los modelos la representación gráfica de la curva de ROC (Receiver Operating Characteristic Curve) para las dos redes pre-entrenadas con mejor desempeño se muestran en las Figuras 29 y 30 para su análisis. En ambos casos (modelos) los conjuntos de datos representan la exactitud y la pérdida registrada durante el entrenamiento y la validación. A partir de la curva de progreso del entrenamiento y validación, es notable que el modelo VGG-19 presenta un mejor desempeño que Alexnet al alcanzar el nivel máximo de precisión para todos los conjuntos de datos en tan solo 18 épocas.

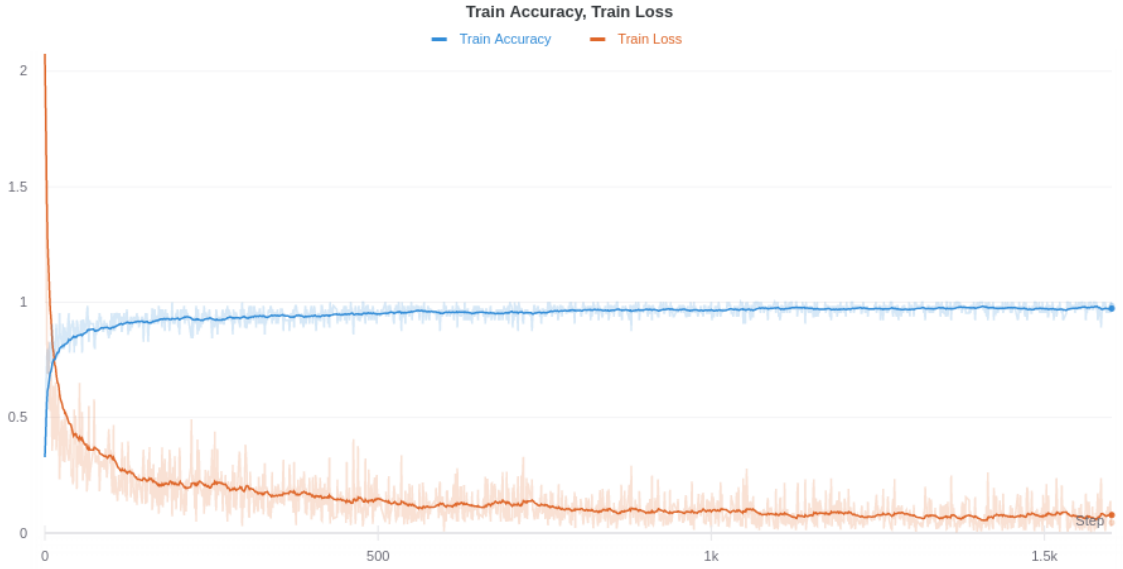


(a) VGG-19: loss de entrenamiento vs loss de validación

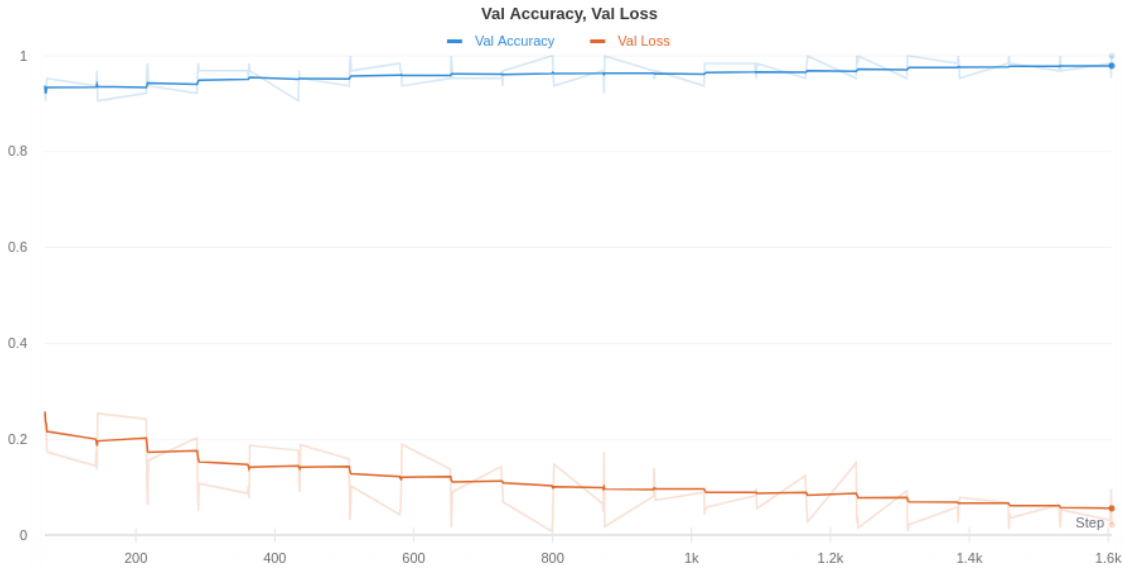
Figura 29. Curva de progreso del entrenamiento y curva de ROC de VGG-19 (1)



(b) VGG-19: exactitud de entrenamiento vs exactitud de validación

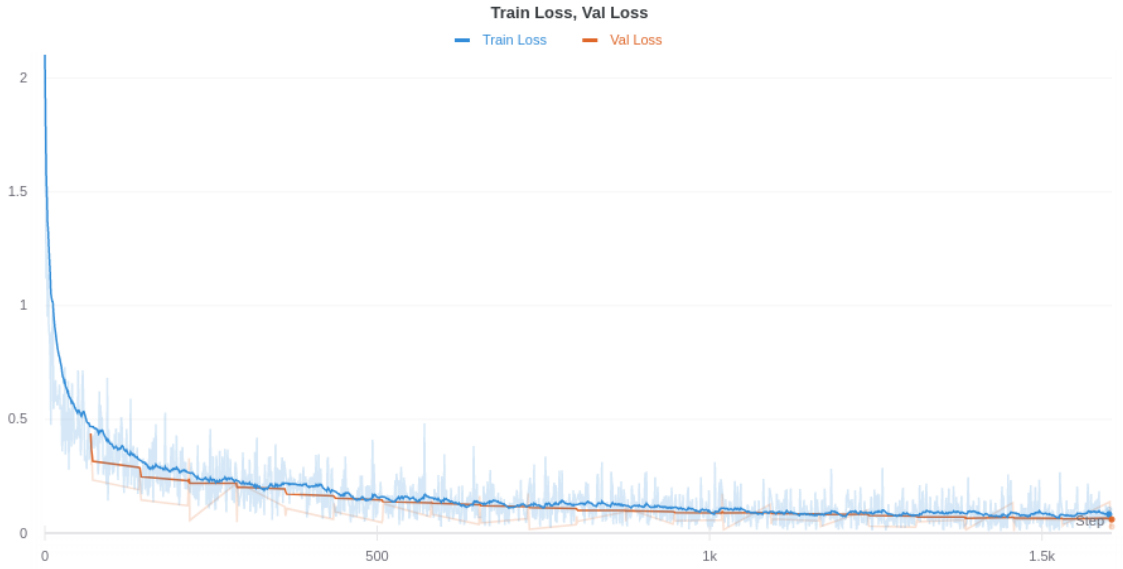


(c) VGG-19: loss vs exactitud de entrenamiento

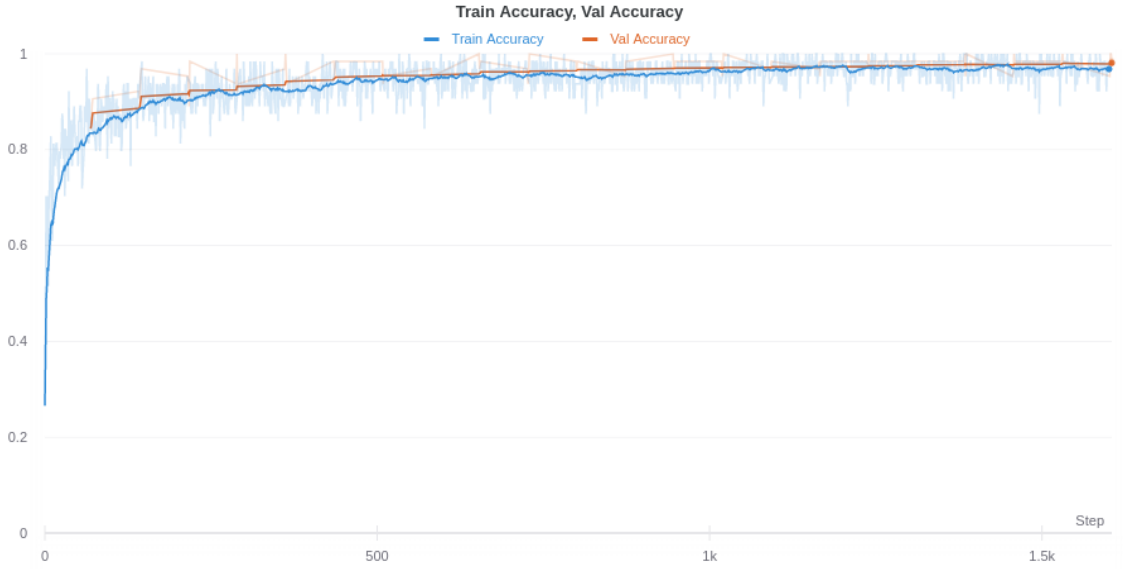


(d) VGG-19: loss vs exactitud de validacion

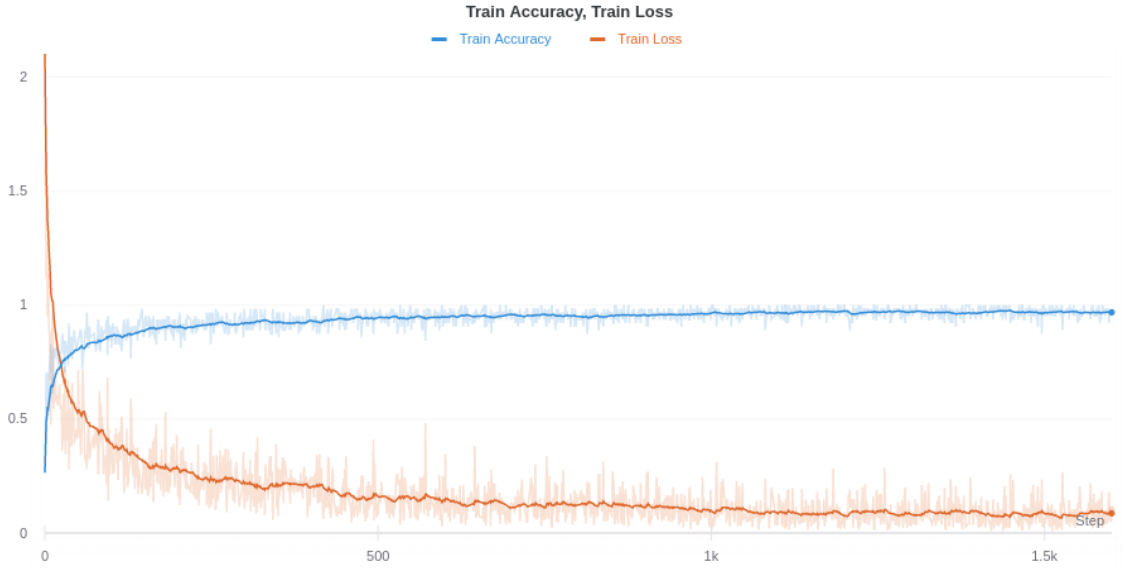
Figura 29. Curva de progreso del entrenamiento y curva de ROC de VGG-19 (2)



(a) AlexNet: loss de entrenamiento vs loss de validación

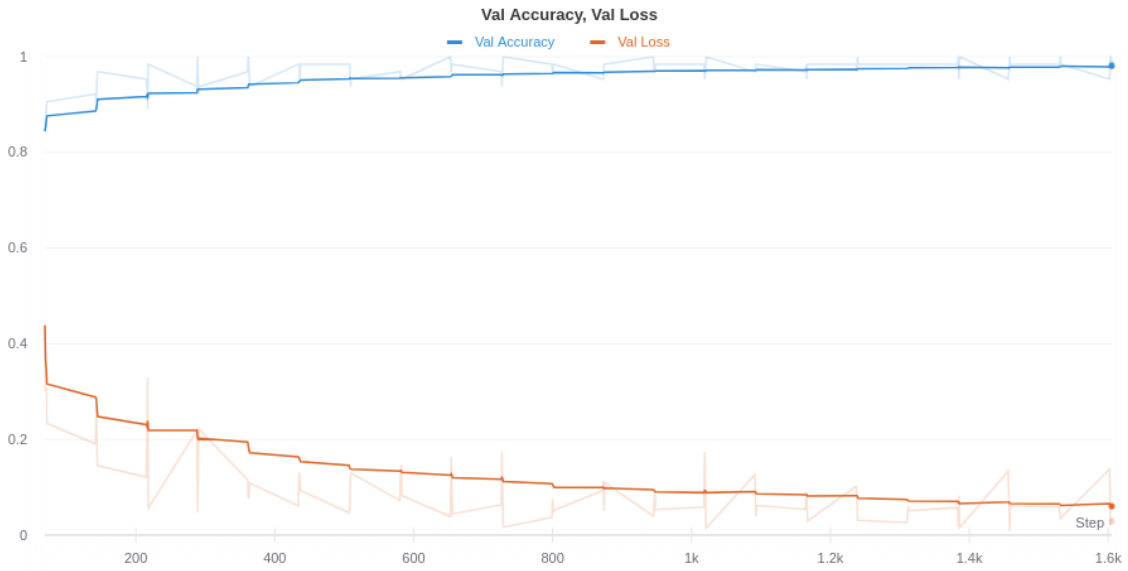


(b) AlexNet: exactitud de entrenamiento vs exactitud de validación



(c) AlexNet: loss vs exactitud de entrenamiento

Figura 30. Curva de progreso del entrenamiento y curva de ROC de AlexNet (1)



(d) AlexNet: loss vs exactitud de validacion

Figura 30. Curva de progreso del entrenamiento y curva de ROC de AlexNet (2)

4.3.2 Resultados cualitativos

La evaluación cualitativa puede llevarse a acabo usando la interpretabilidad de un modelo mediante una disección intrínseca de la red (Li *et al*., 2021). El cual, consiste en realizar una interpretación visual de las características (Barré *et al*., 2017) por las que la red aprende a clasificar. En tal sentido, se aplicó los métodos de Integrated Gradients y SmoothGrad (Smilkov *et al*., 2017; Sundararajan *et al*., 2017) en cada modelo pre-entrenado. Así, el modelo traza una serie de puntos sobre lo que considera relevante en la imagen de entrada. De este modo, las áreas con mayor densidad u aglomeración de puntos son aquellas características que el modelo pondera para realizar la clasificación.

La Figura 31 ilustra la representación visual intrínseca de las características que cada modelo pondera. Las entradas presentan dos casos cuando las imágenes son no segmentadas (a) y segmentadas (b). De los resultados obtenidos con el análisis cuantitativo las redes con mejor desempeño, VGG-19 y AlexNet, aprenden características morfológicas de la hoja de alto nivel como las venas y bordes. Por otro lado, Inception V3 y ResNet-101 aprenden a clasificar en funciones laterales casi ignorando la hoja resaltando su bajo desempeño al momento de clasificar imágenes de hojas.

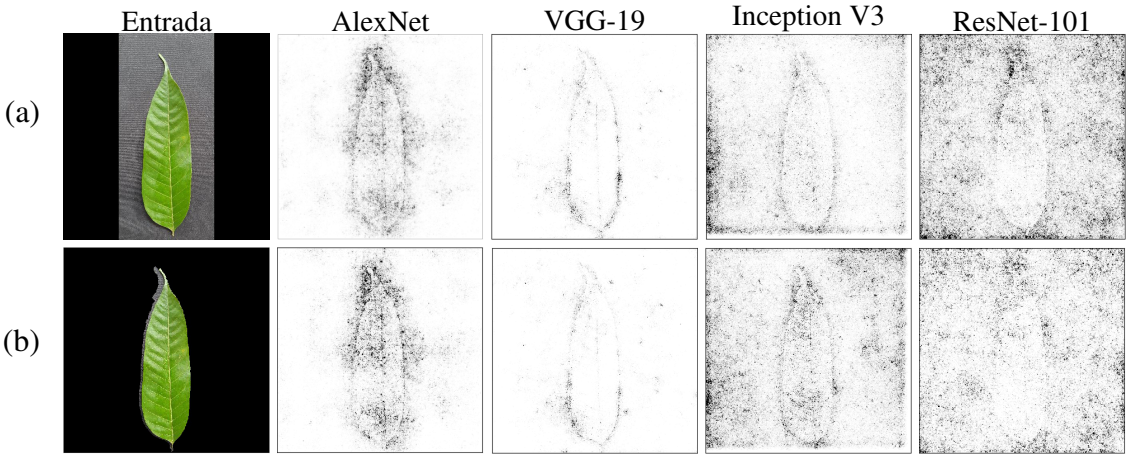


Figura 31. Interpretabilidad de los modelos entrenados
La Figura 31(a) imágenes no preprocesadas, o (b) imágenes preprocesadas.

A partir de los resultados cuantitativos y cualitativos, se pudo observar que VGG-19 es capaz de clasificar correctamente las especies forestales maderables que conforman el *dataset* con una alta probabilidad de predicción verdadera y una baja tasa de falsos positivos. Además, los modelos entrenados con imágenes no preprocesadas se ajustan mejor. El segundo modelo pre-entrenado con mejores resultados fue AlexNet obteniendo resultados similares a VGG-19; al contrario, Inception V3 y ResNet-101 presentaron un desempeño deficiente continuamente.

4.4 Discusión

La primera Subsección de discusión ilustra una comparación del modelo pre-entrenado (con mejores resultados) con otros trabajos de vanguardia existentes sobre la clasificación de especies forestales maderables de la Amazonia. La comparación se ha realizado en términos de exactitud como métrica, característica, cantidad de especies e imágenes en total, es decir, no se comparó aspectos como la partición de datos o tiempo de ejecución. La segunda Subsección, presenta la discusión en sí de los modelos pre-entrenados según su desempeño utilizando el *dataset* denominado The Peruvian Amazon Forestry Dataset.

4.4.1 Comparación de modelos orientados a la clasificación de especies forestales de la Amazonia

Las medidas de desempeño usadas en la Tabla 12 indican que los modelos AlexNet y VGG-19 demostraron ser mejores logrando alcanzar el valor de 97,00 %. Sin embargo, tras los experimentos realizados VGG-19 demostró ser superior a AlexNet. La comparación del modelo VGG-19 con otros trabajos existentes se da en la Tabla 14. De los datos tabulares, los datos resaltados con negrita son aquellas propias del presente estudio extraídos de la Tabla 11 respecto a las entradas no segmentadas.

Tabla 14
Comparación del mejor modelo con otros trabajos en la clasificación de especies forestales maderables

Nro.	Dataset	Caracterís -tica	Nro. de Especies	Nro. de Imágenes	CNN	Exactitud	Región
1	Propio	hoja	1	2 800	Propio	93,00 %	Amazonia Peruana
2	Propio	madera	16	355	Propio	91,00 %	Amazonia Peruana
3	lifeCLEF 2019	planta: todos los órganos	10 000	258 909	Inception-ResNet	96,81 %	Amazonia Guayana
4	**Peruvian Amazon Foresry Dataset**	**hoja**	**10**	**59 441**	**VGG-19**	**97.02 %**	**Amazonia Peruana**

Para la comparación y evaluación se utilizaron los resultados rescatados a partir de datos publicados por los respectivos autores. Aun así, los conjuntos de datos poseen similitud contextual al ser de especies forestales de la Amazonia y los modelos utilizados son CNNs. Con respecto a los *dataset*, Peruvian Amazon Forestry Dataset a pesar de tener una menor cantidad de especies en comparación a lifeCLEF 2019 presenta una mejor relación con respecto al número de imágenes ya que para que para realizar el entrenamiento se requiere una gran cantidad de imágenes por clase. Las investigaciones utilizaron diversas CNNs para la clasificación de especies, VGG-19 obtuvo el mejor resultado respecto a la exactitud en comparación con los otros trabajos existentes en el estado del arte (Apolinario *et al*., 2019; Chulif *et al*., 2019; Reátegui & Velasco, 2018) presentando una exactitud del 97.02 %.

El modelo pre-entrenado VGG-19 usado en la presente investigación para la clasificación de especies forestales maderables ofrece varios beneficios en contraste a sus antecedentes. En primer lugar, ofrece un máximo nivel en la exactitud para el *dataset* conformado. En segundo lugar, no es necesario realizar un preprocesamiento a las imágenes de entrada siendo posible su clasificación con imágenes en bruto y amigables con el medio ambiente. Tercero, a diferencia de otros trabajos (Rizk, 2019), al hacer uso de CNNs pre-entrenadas por transferencia se elimina la necesidad de extraer cualquier característica.

4.4.2 Desempeño de los modelos

La gran acogida de los modelos de aprendizaje profundo en el procesamiento de imágenes permitió avanzar en la investigación y en la aplicación del reconocimiento y clasificación de especies forestales mediante el uso de imágenes (Barré *et al*., 2017; Saini *et al*., 2020). Se desean modelos precisos, ligeros y rápidos para la clasificación de especies forestales maderables, de modo que se pueda aplicar en la realidad de manera adecuada. Un estudio reciente en aprendizaje profundo ha demostrado que los modelos más profundos como Inception V3 y ResNet-101 son más precisos y eficientes de entrenar (Kornblith *et al*., 2019). Sin embargo, nuestras evaluaciones cuantitativas y cualitativas evidencian que VGG-19 y AlexNet son superiores para el *dataset* compuesto, The Peruvian Amazon forestry dataset. Obteniendo ambos modelos una mayor probabilidad de coincidencia, por ejemplo, durante el entrenamiento alcanzaron resultados superiores al 97 % respecto a la exactitud superando hasta en un 34,28 % y 33,92 % a ResNet-101 e Inception V3, respectivamente.

Gracias a la interpretabilidad de los modelos podemos señalar que Alexnet aprende de la forma, textura y la venación de la hoja y presenta ruido ligero. Mientras, VGG-19 presenta escasamente ruido, y aprende de la forma y venación de la hoja, método ya utilizado por Rizk (2019). Por lo tanto, VGG-19 es el modelo pre-entrenado con mejor desempeño tareas referentes a la clasificación de imágenes de hojas, resultado que coincide con los de otros estudios (Rizk, 2019; Tan *et al*., 2018).

Por lo contrario, Inception V3 y ResNet-101 se desempeñaron relativamente bien en comparación a VGG-19, como se ilustra en la Tabla 11. Esto prueba que las CNNs

funcionan bien cuando estas no son tan profundas para la clasificación de especies forestales maderables. Del mismo modo, es más fácil entrenar AlexNet en comparación con el resto de los modelos seleccionados. VGG-19, funciona bien, aunque lleva mucho tiempo entrenar en comparación a AlexNet. De manera similar, Inception V3 y ResNet-101 son computacionalmente costosos en términos de ejecución, además presentan otros tipos de desafíos como el sobreajuste.

Las conexiones residuales permiten explorar diferentes niveles de capas siendo útiles en *datasets* complejos (Szegedy *et al*., 2015). Las conexiones por *skip* pasan por alto singularidades rompiendo con la dependencia lineal de una red (K. He *et al*., 2016). Por consiguiente, estas dos características podrían agregar ruido influyendo en los resultados cuantitativos y cualitativos de Inception V3 y ResNet-101.

Adicionalmente, el uso de algoritmos para imágenes preprocesadas es una práctica común en el aprendizaje profundo ya que busca que la red se centre en el objeto y no en el ambiente o fondo. Sin embargo, respecto al *dataset* utilizado el uso de imágenes preprocesadas no aportan mejoras en la clasificación. Son las imágenes no preprocesadas las que obtienen un mejor desempeño, ya que el modelo logra aprender a distinguir la hoja del fondo al realizar la clasificación de especies forestales maderables.

4.5 Prueba de Hipótesis

La prueba de hipótesis del modelo pre-entrenado para clasificar especies forestales maderables de la Amazonia peruana, se planteó de la siguiente manera:

1. Planteamiento de la prueba de hipótesis

$$H_0 : \mu \leq \mu_0$$

$$H_1 : \mu > \mu_0$$

La hipótesis nula H_0 nos señala que μ es menor o igual que μ_0 y la hipótesis alterna H_1 nos indica que μ mayor μ_0.

2. Nivel de significancia

   Para comprobar la prueba de hipótesis, se eligió el nivel de significancia de 0,05 o 5 % de error:

$$\alpha = 0,05 = 0,5\,\%$$

   Se utilizó la distribución t de student para el área de una cola en relación al grado de libertad $GL, n-1 = 9$, donde n es la muestra, por lo tanto, el valor de t es:

$$t_\alpha = t_{0,05} = 1,83$$

   Se usó la desviación estándar de muestra s, para las 10 especies forestales maderables se dio según su promedio de asertividad por especie x, el promedio de asertividad de las cuatro redes neuronales $\bar{x}$ y el total de especies n, así, s es dada por:

$$s = \sqrt{\frac{\sum(x-\bar{x})^2}{n-1}} \times 100 = \sqrt{\frac{418,72\,\%}{9}} \times 100 = 6,82$$

3. Zona de rechazo y regla de decisión

   Se eligió como método de prueba de hipótesis la prueba t-Student o Test-T, para $n = 10; n < 30$ y $\bar{x} =$ promedio de asertividad de los modelos. Se utilizó como media de la hipótesis nula el valor μ :

$$\mu = (\bar{x} - t_\alpha \frac{S}{\sqrt{n}}) \times 100$$

   Por lo que, $\mu = (86,49\,\% - 1,83\frac{6,82\,\%}{\sqrt{10}}) \times 100 = 82,54$, el cual se ha usado para calculo de la prueba estadística.

4. Estadística de prueba

   Se usó el estadístico de prueba t de student para las siguientes estadísticas:

$$t = \frac{\bar{x} - \mu}{\frac{s}{\sqrt{n}}}$$

Respecto al modelo pre-entrenado AlexNet

H_0: La red neuronal convolucional AlexNet no obtiene los mejores resultados de asertividad en la clasificación de especies forestales maderables de la Amazonia peruana.

H_1: La red neuronal convolucional AlexNet obtiene los mejores resultados de asertividad en la clasificación de especies forestales maderables de la Amazonia peruana.

Tabla 15
Prueba de muestra única en función del modelo AlexNet

Prueba de muestra única				
Modelo	**Nivel de asertividad**			
AlexNet	N especies	X (%) exactitud	S (%)	μ (%)
	10	96.61	6,82	82.54

$t = \frac{96,61-82,54}{\frac{6,82}{\sqrt{10}}} = 6,52$; se usó los datos de la Tabla 15 en el reemplazo de la ecuación. Rechazamos la hipótesis nula (el puntaje t de $6,52$ está en área de rechazo); y se acepta la hipótesis alterna. Lo que señala que el modelo AlexNet obtiene los mejores resultados de asertividad en la clasificación de especies forestales maderables de la Amazonia peruana.

Respecto al modelo pre-entrenado VGG-19

H_0: La red neuronal convolucional VGG-19 no obtiene los mejores resultados de asertividad en la clasificación de especies forestales maderables de la Amazonia peruana.

H_1: La red neuronal convolucional VGG-19 obtiene los mejores resultados de asertividad en la clasificación de especies forestales maderables de la Amazonia peruana.

Tabla 16
Prueba de muestra única en función del modelo VGG-19

Prueba de muestra única				
Modelo	**Nivel de asertividad**			
VGG-19	N especies	X (%) exactitud	S (%)	μ (%)
	10	97.02	6,82	82.54

De la Tabla 16, $t = \frac{97,02-82,54}{\frac{6,82}{\sqrt{10}}} = 6,71$. Rechazamos la hipótesis nula (el puntaje t de $6,71$ está en área de rechazo); y se acepta la hipótesis alterna. Lo que afirma que el modelo VGG-19 obtiene los mejores resultados de asertividad en la clasificación de especies

forestales maderables de la Amazonia peruana.

Respecto al modelo pre-entrenado Inception V3

H_0: La red neuronal convolucional Inception V3 no obtiene los mejores resultados de asertividad en la clasificación de especies forestales maderables de la Amazonia peruana.

H_1: La red neuronal convolucional Inception V3 obtiene los mejores resultados de asertividad en la clasificación de especies forestales maderables de la Amazonia peruana.

Tabla 17
Prueba de muestra única en función del modelo Inception V3

Prueba de muestra única				
Modelo	**Nivel de asertividad**			
Inception V3	N especies	X (%) exactitud	S (%)	μ (%)
	10	74.11	6,82	82.54

Se usó los datos proporcionados por la Tabla 18, para la ecuación $t = \frac{74,11-82,54}{\frac{6,82}{\sqrt{10}}} = -3,91$. Aceptamos la hipótesis nula (el puntaje t de $-3,91$ está dentro del área de no rechazo); y se rechaza la hipótesis alterna. Así, modelo Inception V3 no obtiene los mejores resultados de asertividad en la clasificación de especies forestales maderables de la Amazonia peruana.

Respecto al modelo pre-entrenado ResNet-101

H_0: La red neuronal convolucional ResNet-101 no obtiene los mejores resultados de asertividad en la clasificación de especies forestales maderables de la Amazonia peruana.

H_1: La red neuronal convolucional ResNet-101 obtiene los mejores resultados de asertividad en la clasificación de especies forestales maderables de la Amazonia peruana.

Tabla 18
Prueba de muestra única en función del modelo ResNet-101

Prueba de muestra única				
Modelo	**Nivel de asertividad**			
ResNet-101	N especies	X (%) exactitud	S (%)	μ (%)
	10	78.22	6,82	82.54

$t = \frac{78,22-82,54}{\frac{6,82}{\sqrt{10}}} = -2,00$; se usó los datos proporcionados por la Tabla 18 en el reemplazo de la ecuación. Aceptamos la hipótesis nula (el puntaje t de $-2,00$ está dentro del área de no rechazo); y se rechaza la hipótesis alterna. Lo que implica que el modelo ResNet-101 no obtiene los mejores resultados de asertividad en la clasificación de especies forestales maderables de la Amazonia peruana.

Finalmente, de las Tablas 15 - 18, se puede observar que el modelo VGG-19 es la red convolucional pre-entrenada con aprendizaje por transferencia que obtiene mejores resultados de asertividad en la clasificación de especies forestales maderables de la Amazonia peruana.

CONCLUSIONES

En la presente tesis, se realizó la clasificación de imágenes de hojas de especies forestales maderables utilizando las redes pre-entrenadas AlexNet, VGG-19, Inception V3 y ResNet-101 con el objetivo general de determinar la red neural convolucional con mejor desempeño para el conjunto de datos conformado, alcanzando las siguientes conclusiones:

1. La red neuronal convolucional que obtuvo mejores resultados en la clasificación de especies forestales maderables en la Amazonia peruana ha sido **VGG-19** logrando un mejor desempeño con imágenes no segmentadas, el **97.61 %** de exactitud en el entrenamiento, en la validación el **98.87 %** de exactitud, y un **97.02 %** de exactitud en el test.

2. Se conformó el conjunto de datos Peruvian Amazon Forestry Dataset, compuesto por 59,441 imágenes de hojas correspondientes a 10 especies forestales maderables en peligro de extinción con importancia económica de la Amazonia peruana. El conjunto de datos es significativo, dado que las redes neuronales que aprendieron de él lograron generalizar sus resultados hacia ejemplos no vistos durante el entrenamiento.

3. Se implementaron las redes neuronales AlexNet, VGG-19, Inception V3 y ResNet-101 para clasificar especies forestales maderables utilizando el conjunto de datos conformado. Para el entrenamiento, se configuró cada uno los modelos con 16 mini-batches y se aplicó la función de activación ReLU, asimismo, se usaron dos tipos de entradas (imágenes no segmentadas y segmentadas). Los criterios de entrenamiento fueron apropiados puesto que las redes neuronales obtuvieron resultados sobresalientes en los distintos parámetros de evaluación.

4. Se establecieron dos tipos de evaluación para valorar el desempeño de las cuatro redes neuronales: cuantitativa y cualitativamente. Las métricas utilizadas fueron apropiadas debido a que permitieron hacer una comparación detallada y complementaria del desempeño de cada red neuronal, mencionando los principales hallazgos a continuación:

- **Evaluación cuantitativa** las redes neuronales convolucionales fueron evaluadas según el tipo de entrada en términos de exactitud, precisión, exhaustividad y valor-F1. Según su desempeño el orden de las redes es: VGG-19 >AlexNet >ResNet-101 >Inception V3.
- **Evaluación cualitativa** se aplicó un algoritmo de interpretabilidad en los cuatro modelos. Se observó que AlexNet y VGG-19 aprendieron características morfológicas de la hoja de alto nivel como la forma y vena, lo cual consolida su sobresaliente desempeño. Sin embargo, Inception V3 y ResNet-101 se enfocaron en el ambiente la imagen lo que redujo la probabilidad de acierto.

RECOMENDACIONES

Se recomienda para trabajos posteriores relacionados con la clasificación de especies forestales maderables utilizar redes neuronales menos profundas como VGG-19. En vista que la red es adecuada respecto a su exactitud y puede integrarse a una aplicación móvil para su empleo en el mundo real.

Se recomienda usar el conjunto de datos Peruvian Amazon Forestry Dataset conformado e implementar un laboratorio especializado para la toma de imágenes de hojas a fin de aumentar la cantidad de clases y ejemplos de alta calidad.

Se recomienda entrenar las redes neuronales convolucionales por transferencia con imágenes no preprocesadas para obtener resultados sobresalientes en la clasificación de especies forestales maderables. Respecto a la extracción automática de características, se sugiere priorizar la atención en la forma y venación de la hoja.

Para futuras investigaciones relacionadas a la implementación de modelos de aprendizaje profundo, es recomendable explorar la relación de la exactitud y los algoritmos de interpretabilidad, ya que permiten ver la conexión de los resultados de predicción y otros factores del entrenamiento como la robustez o el preprocesamiento que pueden ser optimizados desde la intervención humana.

BIBLIOGRAFÍA

Adit, V., Rubesh, C., Bharathi, S., Santhiya, G. & Anuradha, R. (2020). A Comparison of Deep Learning Algorithms for Plant Disease Classification. *Advances in Cybernetics, Cognition, and Machine Learning for Communication Technologies* (pp. 153-161). Springer. https://doi.org/hnrq

Álvarez, J. (2007). Reserva Nacional Allpahuayo-Mishana: una joya natural al lado de Iquitos.

Anubha, S., Sathiesh, V. & Harini, S. (2019). A study on plant recognition using conventional image processing and deep learning approaches. *Journal of Intelligent & Fuzzy Systems*, *36*(3), 1997-2004. https://doi.org/10.3233/JIFS-169911

Apolinario, M., Paredes, D. & Bustamante, S. G. (2019). Open Set Recognition of Timber Species Using Deep Learning for Embedded Systems. *IEEE Latin America Transactions*, *17*(12), 2005-2012. https://doi.org/hnrv

Ashqar, B., Abu-Nasser, B. & Abu-Naser, S. (2019). Plant Seedlings Classification Using Deep Learning.

Azlah, M., Chua, L., Rahmad, F., Abdullah, F. & Wan, S. (2019). Review on Techniques for Plant Leaf Classification and Recognition. *Computers*, *8*(4), 77. https://doi.org/hnrs

Barré, P., Stöver, B., Müller, K. & Steinhage, V. (2017). LeafNet: A computer vision system for automatic plant species identification. *Ecological Informatics*, *40*, 50-56. https://doi.org/gbtbcb

Bendezú, Y. (2016). Clave Dendrologica para la Identificación de los Principales Árboles de la Región Ucayali.

Bengio, Y., Goodfellow, I. & Courville, A. (2017). *Deep learning* (Vol. 1). MIT press.

Biewald, L. (2020). Experiment tracking with weights and biases. *Software available from wandb. com*, *2*.

Bonnet, P., Goëau, H., Hang, S., Lasseck, M., Šulc, M., Malécot, V., Jauzein, P., Melet, J.-C., You, C. & Joly, A. (2018). Plant identification: experts vs. machines in the era of deep learning. *Multimedia Tools and Applications for Environmental & Biodiversity Informatics* (pp. 131-149). Springer, Cham. https://doi.org/hnr2

Brownlee, J. (2018). *Better deep learning: train faster, reduce overfitting, and make better predictions*. Machine Learning Mastery.

Carpentier, M., Giguère, P. & Gaudreault, J. (2018). Tree species identification from bark images using convolutional neural networks. *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1075-1081.

Carranza-Rojas, J. & Mata-Montero, E. (2016). Combining leaf shape and texture for Costa Rican plant species identification. *CLEI Electronic journal*, *19*(1), 7-7.

Castillo, A. (2010). Manual dendrológico de las principales especies de interés comercial actual y potencial de la zona del Alto Huallaga. *Perú, Camara Nacional Forestal*.

Chu, P.-S., Yu, Z.-P. & Hastenrath, S. (1994). Detecting climate change concurrent with deforestation in the Amazon Basin: Which way has it gone? *Bulletin of the american Meteorological Society*, *75*(4), 579-584.

Chulif, S., Jing, K., Wei, T., Al, M. & Chang, Y. (2019). Plant identification on amazonian and guiana shield flora: Neuon submission to lifeclef 2019 plant. *CLEF*.

CNF, OIMT & SERFOR. (2016). Cartilla de precios de productos y servicios forestales.

Cossío, R., Menton, M., Cronkleton, P. & Larson, A. (2014). Manejo forestal comunitario en la Amazonía peruana.

Davies, R. (2017). *Computer vision: principles, algorithms, applications, learning*. Academic Press.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K. & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. *2009 IEEE conference on computer vision and pattern recognition*, 248-255. https://doi.org/cvc7xp

Deng, L. & Liu, Y. (2018). *Deep learning in natural language processing*. Springer.

Ellis, B. (2009). *Manual of leaf architecture*. Published in association with the New York Botanical Garden.

Elnemr, H. (2019). Convolutional neural network architecture for plant seedling classification. *International Journal of Advanced Computer Science and Applications*, *10*(8), 319-325.

Esmaeili, H. & Phoka, T. (2018). Transfer learning for leaf classification with convolutional neural networks. *2018 15th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, 1-6. https://doi.org/hnrr

Fang, M., Yue, G. & Yu, Q. (2009). The study on an application of otsu method in canny operator. *Proceedings. The 2009 International Symposium on Information Processing (ISIP 2009)*, 109.

Farfan-Rios, W., Garcia-Cabrera, K., Salinas, N., Raurau-Quisiyupanqui, M. & Silman, M. (2015). Lista anotada de árboles y afines en los bosques montanos del sureste peruano: la importancia de seguir recolectando. *Revista peruana de biología*, *22*(2), 145-174.

Geetharamani, G. & Pandian, A. (2019). Identification of plant leaf diseases using a nine-layer deep convolutional neural network. *Computers & Electrical Engineering*, *76*, 323-338. https://doi.org/hnr3

Goeau, H., Bonnet, P. & Joly, A. (2017). Plant identification based on noisy web data: the amazing performance of deep learning (LifeCLEF 2017). *CLEF: Conference and Labs of the Evaluation Forum*, (1866).

Goëau, H., Bonnet, P. & Joly, A. (2019). Overview of lifeclef plant identification task 2019: diving into data deficient tropical countries.

Góes, E. (2019). O rio Amazonas: fonte de diversidade. *Revista del Museo de La Plata*.

Goodfellow, I., Bengio, Y. & Courville, A. (2016). *Deep learning*. MIT press.

He, G., Xia, Z., Zhang, Q., Zhang, H. & Fan, J. (2018). Plant species identification by bi-channel deep convolutional networks. *Journal of Physics: Conference Series*, *1004*(1), 012015.

He, K., Zhang, X., Ren, S. & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *Proceedings of the IEEE international conference on computer vision*, 1026-1034.

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

Hemanth, D. & Estrela, V. (2017). *Deep learning for image processing applications* (Vol. 31). IOS Press.

Hinton, G., Srivastava, N., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

Ibrahim, Z., Sabri, N. & Mangshor, N. N. A. (2018). Leaf Recognition using Texture Features for Herbal Plant Identification. *Indonesian Journal of Electrical Engineering and Computer Science*, *9*(1), 152-156. https://doi.org/hnr4

ITP/CITEmadera. (2018). La Industria de la Madera en el Perú.

Kadir, A., Nugroho, L., Susanto, A. & Santosa, P. (2012). Performance improvement of leaf identification system using principal component analysis. *International Journal of Advanced Science and Technology*, *44*, 113-124.

Kaur, T. & Gandhi, T. K. (2020). Deep convolutional neural networks with transfer learning for automated brain image classification. *Machine Vision and Applications*, *31*, 1-16. https://doi.org/hnrx

Keenan, R., Reams, G., Achard, F., de Freitas, J., Grainger, A. & Lindquist, E. (2015). Dynamics of global forest area: Results from the FAO Global Forest Resources Assessment 2015. *Forest Ecology and Management*, *352*, 9-20.

Kingma, D. & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv:1412.6980*.

Kornblith, S., Shlens, J. & Le, Q. (2019). Do better imagenet models transfer better? *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2661-2671. https://doi.org/hnkr

Krizhevsky, A. (2009). Learning Multiple Layers of Features from Tiny Images. *Master's thesis, University of Tront*.

Krizhevsky, A., Sutskever, I. & Hinton, G. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 1097-1105.

Kumar, N., Belhumeur, P., Biswas, A., Jacobs, D., Kress, W., Lopez, I. & Soares, J. (2012). Leafsnap: A computer vision system for automatic plant species identification. *European conference on computer vision*, 502-516.

Li, X., Xiong, H., Li, X., Wu, X., Zhang, X., Liu, J., Bian, J. & Dou, D. (2021). Interpretable Deep Learning: Interpretation, Interpretability, Trustworthiness, and Beyond.

Lin, M., Chen, Q. & Yan, S. (2013). Network in network. *arXiv preprint arXiv:1312.4400*.

Maeda, V., Galván, C., Zanella, L., Celaya, J., Galván, J., Gamboa, H., Luna, H., Magallanes, R., Guerrero, C. & Olvera, C. (2020). Comparison of Convolutional Neural Network Architectures for Classification of Tomato Plant Diseases. *Applied Sciences*, *10*(4), 1245. https://doi.org/gnjsqt

McCulloch, W. & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, *5*(4), 115-133.

Mehrotra, K., Mohan, C. & Ranka, S. (1997). *Elements of artificial neural networks*. MIT press. https://doi.org/djsbj6

MINAGRI. (2014). Reglamento de Organización y Funciones del Servicio Nacional Forestal y de Fauna Silvestre – SERFOR.

Montenegro, R. (2018). *Clasificación de especies forestales maderables de la Amazonía Peruana aplicando análisis Clúster con algoritmo Clara*. [Tesis de maestría, Universidad Nacional Agraria La Molina]. Repositorio Institucional Universidad Nacional Agraria La Molina. https://hdl.handle.net/20.500.12996/3759

Neves, E. (2019). El río Amazonas: fuente de diversidad. *Revista del Museo de La Plata*, *4*(2), 385-400.

O'Shea, K. & Nash, R. (2015). An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*.

OSINFOR. (2017). Protocolo para la evaluación de individuos maderables-Proceso de convergencia 2.

OSINFOR. (2018). Fichas De Identificación De Especies Forestales Maderables Y No Maderables De La Amazonía Peruana.

OSINFOR. (s.f.). Fichas de identificación de especies forestales maderables y no maderables de la Amazonía peruana.

Otter, D., Medina, J. & Kalita, J. (2020). A survey of the usages of deep learning for natural language processing. *IEEE Transactions on Neural Networks and Learning Systems*. https://doi.org/ghnd2g

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., … Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, *32*, 8026-8037.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V. *et al*. (2011). Scikit-learn: Machine learning in Python. *the Journal of machine Learning research*, *12*, 2825-2830.

Pham, T. (2020). A Comprehensive Study on Classification of COVID-19 on Computed Tomography with Pretrained Convolutional Neural Networks.

Pizer, S., Amburn, E., Austin, J., Cromartie, R., Geselowitz, A., Greer, T., Ter, B., Zimmerman, J. & Zuiderveld, K. (1987). Adaptive histogram equalization and its va-

riations. *Computer vision, graphics, and image processing*, *39*(3), 355-368. https://doi.org/cbvkxd

Qian, W., Huang, Y., Liu, Q., Fan, W., Sun, Z., Dong, H., Wan, F. & Qiao, X. (2020). UAV and a deep convolutional neural network for monitoring invasive alien plants in the wild. *Computers and Electronics in Agriculture*, *174*, 105519. https://doi.org/gh3dk2

Rawat, W. & Wang, Z. (2017). Deep convolutional neural networks for image classification: A comprehensive review. *Neural computation*, *29*(9), 2352-2449. https://doi.org/gbvf4p

Reátegui, A. & Velasco, M. (2018). *Aplicación informática para reconocimiento de la especie camu camu (Myrciaria Dubia) a través de redes neuronales convolucionales, en Iquitos Perú, durante el año 2017*. [Tesis de maestría, Universidad Nacional de la Amazonia Peruana]. Repositorio Institucional Digital Universidad Nacional de la Amazonia Peruana.

Resolución de Dirección Ejecutiva Nro. 230-2019-MINAGRI-SEFOR-DE. (2019).

Resolución de Dirección Ejecutiva Nro.118-2019-MINAGRI-SEFOR-DE. (2019).

Rizk, S. (2019). *Plant leaf classification using dual path convolutional neural networks*. [Tesis de maestría, Notre Dame University-Louaize]. Repositorio Dspace Notre Dame University-Louaize. http://ir.ndu.edu.lb/123456789/990

Saikia, A., Bora, K., Mahanta, L. & Das, A. (2019). Comparative assessment of CNN architectures for classification of breast FNAC images. *Tissue and Cell*, *57*, 8-14. https://doi.org/hnrz

Saini, G., Khamparia, A. & Luhach, A. K. (2020). Classification of Plants Using Convolutional Neural Network. *First International Conference on Sustainable Technologies for Computational Intelligence*, 551-561. https://doi.org/hnrn

Simonyan, K. & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Smilkov, D., Thorat, N., Kim, B., Viégas, F. & Wattenberg, M. (2017). Smoothgrad: removing noise by adding noise. *arXiv preprint arXiv:1706.03825*.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, *15*(1), 1929-1958.

Su, D., Zhang, H., Chen, H., Yi, J., Chen, P.-Y. & Gao, Y. (2018). Is Robustness the Cost of Accuracy?–A Comprehensive Study on the Robustness of 18 Deep Image Classification Models. *Proceedings of the European Conference on Computer Vision*.

Sun, Y., Liu, Y., Wang, G. & Zhang, H. (2017). Deep learning for plant identification in natural environment. *Computational intelligence and neuroscience*, *2017*. https://doi.org/gj6bms

Sundararajan, M., Taly, A. & Yan, Q. (2017). Axiomatic attribution for deep networks. *International Conference on Machine Learning*, 3319-3328.

Syarief, M. & Setiawan, W. (2020). Convolutional neural network for maize leaf disease image classification. *Telkomnika*, *18*(3). https://doi.org/hnrw

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. & Rabinovich, A. (2015). Going deeper with convolutions. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1-9.

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

Tan, J., Chang, S.-W., Kareem, S., Yap, H. & Yong, K.-T. (2018). Deep learning for plant species classification using leaf vein morphometric. *IEEE/ACM transactions on computational biology and bioinformatics*. https://doi.org/hnrp

Ter, H., Prado, P., de Lima, R., Pos, E., de Souza, L., de Andrade, D., Salomão, R. P., Amaral, I., de Almeida, F. & Castilho, C. (2020). Biased-corrected richness estimates for the Amazonian tree flora. *Scientific Reports*, *10*(1), 1-13. https://doi.org/hnr6

Thanh, T., Truong, Q., Truong, Q. & Xuan, H. (2018). Depth learning with convolutional neural network for leaves classifier based on shape of leaf vein. *Asian Conference on Intelligent Information and Database Systems*, 565-575. https://doi.org/hnrt

Toma, A., Stefan, L.-D. & Ionescu, B. (2017). UPB HES SO@ PlantCLEF 2017: Automatic Plant Image Identification using Transfer Learning via Convolutional Neural Network. *CLEF (Working Notes)*.

Too, E., Yujian, L., Njuki, S. & Yingchun, L. (2019). A comparative study of fine-tuning deep learning models for plant disease identification. *Computers and Electronics in Agriculture*, *161*, 272-279. https://doi.org/ggqrs7

Valqui, M., Feather, C. & Llanos, R. (2016). *Haciendo Visible Lo Invisible: Perspectivas Indígenas Sobre la Deforestación en la Amazonía Peruana: Causas Y Alternativas*. Asociación Interétnica de Desarrollo de la Selva Peruana (AIDESEP).

Voulodimos, A., Doulamis, N., Doulamis, A. & Protopapadakis, E. (2018). Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience*. https://doi.org/gfkdks

Wang, W., Yang, Y., Wang, X., Wang, W. & Li, J. (2019). Development of convolutional neural network and its application in image classification: a survey. *Optical Engineering*, *58*(4), 040901. https://doi.org/gmgw3p

Wick, C. & Puppe, F. (2017). Leaf identification using a deep convolutional neural network. *arXiv preprint arXiv:1712.00967*.

Wittmann, F., Schöngart, J., Montero, J. C., Motzer, T., Junk, W., Piedade, M., Queiroz, H. & Worbes, M. (2006). Tree species composition and diversity gradients in white-water forests across the Amazon Basin. *Journal of biogeography*, *33*(8), 1334-1347. https://doi.org/c8hcvj

Wu, J. (2017). Introduction to convolutional neural networks. *National Key Lab for Novel Software Technology. Nanjing University. China*, *5*, 23.

Wu, S., Bao, F., Xu, E., Wang, Y.-X., Chang, Y.-F. & Xiang, Q.-L. (2007). A leaf recognition algorithm for plant classification using probabilistic neural network. *2007 IEEE international symposium on signal processing and information technology*, 11-16.

Zárate, R., Alvarado, L. & Maco, J. (2012). Inventario de cumalas (Myristicaceae) en la Amazonía peruana. *Folia Amazónica*, *21*(1-2), 7-22.

Zhang, H., He, G., Li, F., Xia, Z., Liu, B. & Peng, J. (2021). Plant taxonomy-guided path-based tree classifier for large-scale plant species identification. *Journal of Electronic Imaging*, *30*(2), 023019. https://doi.org/hnr8

Zhang, S., Huang, W., Huang, Y.-a. & Zhang, C. (2020). Plant species recognition methods using leaf image: overview. *Neurocomputing*, *408*, 246-272. https://doi.org/hnr7

Zhu, L., Li, Z., Li, C., Wu, J. & Yue, J. (2018). High performance vegetable classification from images based on alexnet deep learning model. *International Journal of Agricultural and Biological Engineering*, *11*(4), 217-223.

ANEXOS

Anexo 1. Metadatos del dataset

Los Metadatos de las imágenes del *dataset* se organizaron según el estándar Plian Core. A continuación, el ejemplo presenta los metadatos correspondientes a la imagen digital $virolaFlex_A30_0001.jpg$.

```
File
     ImageLeaf_Id
         virolaFlex_A30_0001
    File_Date
        26/02/20-19:20:46
Metadata
     Dataset_ID
         virolaFlex
     dateStamp
          DateTimeOriginal = 2020:01:04 14:05:33
         DateTimeDigitized = 2020:01:04 14:05:33
     Image
          ExifImageWidth = 4608
          ExifImageHeight = 2128
          ImageLength = 2128
          Orientation = 6
         XResolution = (72, 1)
         YResolution = (72, 1)
    Camera
        Flash = 0
        FocalLength = (392, 100)
         DigitalZoomRatio = (100, 100)
         FocalLengthIn35mmFilm = 26
         Make = samsung
         Model = SM-A305G
         ExposureTime = (1, 60)
         FNumber = (170, 100)
         ISOSpeedRatings = 125
    citation
         Virola flexuosa - A.C. Sm.by Instituto de Investigaciones de la Amazonia Peruana
                2020
    dataset
         Amazon Forest Leaf
    References
        From IIAP-GESCON
    TaxonRecord
    RecordMetadata
        Language
        EN, ES
         TargetAudiences
        Scientific Community
    BaseElements
        TaxonRecordID
        0001
        Abstract(EN)
         Tree that reaches 35 meters in height. Straight and cylindrical trunk that can
        reach a commercial height of 20 m. Simple, alternate, oblong or narrow elliptical
        leaves, 5 to 20 cm long, acute apex, corded base, hairless or tomentose with
        sessile trichomes, secondary veins 40-60 pairs, straight, notorious on the underside.
        Inflorescence in panicles 4-9 cm long, densely tomentose, flowers 10-15 per fascicle,
        pedicels 1 mm long. Fruits in capsule immature subglobose, star-shaped, ferruginous.
        Abstract(ES)
         Árbol que alcanza 35 metros de altura. Tronco recto y cilíndrico que puede alcanzar
        una altura comercial de 20 m. Hojas simples, alternas, oblongas o elípticas
        estrechas, de 5 a 20 cm de longitud, ápice agudo, base cordada, glabras o tomentosas
        con tricomas sésiles, venas secundarias 40-60 pares, rectas, notorias por el envés.
        Inflorescencia en paniculas de 4 a 9 cm de largo, densamente tomentosas, flores
        10 - 15 por fascículo, pedicelos de 1 mm de largo. Frutos en cápsula inmaduras
        subglobosas, estrelladotomentosas de color ferrugineo.
    Nomenclature And Classification
     TaxonRecordName
        Virola flexuosa
     Synonyms
```

It does not have a synonym
CommonNames
Caupuri de altura(Spanish,Peru)
cumala(Spanish,Peru)
cumala blanca(Spanish,Peru)
cumala negra(Spanish,Peru)
virola amarilla(Spanish,Peru)
FlatHierarchy
Kingdom: Plantae
Phylum: Magnoliophyta
Class: Magnoliopsida
Order: Magnoliales
Family: Myristicaceae
HabitatAndDistribution
Habitat(EN)
It lives in forests on the mainland, in forests on terraces, forests on hills. It may be dispersed by monkeys and birds.
Habitat(ES)
Habita en bosques de tierra firme, en bosques de terrazas, bosques de colinas. Es posible que se disperse por monos y aves.
Distribution
Ecuador
Bolivia
Peru
Colombia
Brasil.
About
Project_Name
Solución computacional para reconocimiento online de especies forestales maderables de importancia económica y ecológica para la amazonia peruana (vulnerables a la tala ilegal), a través de técnicas de redes neuronales y computación de alto desempeño
Institute_Name
Instituto de Investigaciones de la Amazonia Peruana
Acknowledgments
The present work was carried out thanks to the research funds of the "Fondo Nacional de Desarrollo Científico, Tecnológico y de Innovación Tecnológica (FONDECYT)", an initiative of the "Consejo Nacional de Ciencia, Tecnología e Innovación Tecnológica (CONCYTEC)", grant 022-2018-FONDECYT-BM-IADT-AV.

Anexo 2. Modelos importados

Las redes pre-entrenadas AlexNet, VGG-19, Inception V3 y ResNet-101 sobre el *dataset* Imagenet son accesibles a través de Pytorch. A continuación, se presentan los modelos con aprendizaje por transferencia importados. El *output* nos ilustra sobre la estructura (capas) de cada modelo.

AlexNet (8 capas)

```
################################################
# 1. Importar la red pre-entrenada AlexNet
################################################

model = models.alexnet(pretrained=True)
print(model)

################################################
#OUTOUT
################################################

AlexNet(
(features): Sequential(
(0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
(1): ReLU(inplace=True)
(2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
(3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
(4): ReLU(inplace=True)
(5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
(6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(7): ReLU(inplace=True)
(8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(9): ReLU(inplace=True)
(10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(11): ReLU(inplace=True)
(12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
)
(avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
(classifier): Sequential(
(0): Dropout(p=0.5, inplace=False)
(1): Linear(in_features=9216, out_features=4096, bias=True)
(2): ReLU(inplace=True)
(3): Dropout(p=0.5, inplace=False)
(4): Linear(in_features=4096, out_features=4096, bias=True)
(5): ReLU(inplace=True)
(6): Linear(in_features=4096, out_features=1000, bias=True))
)
```

VGG-19 (19 capas)

```
################################################
# 1. Importar la red pre-entrenada VGG-19
################################################

model = models.vgg19(pretrained=True)
print(model)

################################################
#OUTOUT
################################################

VGG(
(features): Sequential(
(0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(1): ReLU(inplace=True)
(2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(3): ReLU(inplace=True)
```

```
(4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
(5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(6): ReLU(inplace=True)
(7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(8): ReLU(inplace=True)
(9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
(10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(11): ReLU(inplace=True)
(12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(13): ReLU(inplace=True)
(14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(15): ReLU(inplace=True)
(16): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(17): ReLU(inplace=True)
(18): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
(19): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(20): ReLU(inplace=True)
(21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(22): ReLU(inplace=True)
(23): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(24): ReLU(inplace=True)
(25): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(26): ReLU(inplace=True)
(27): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
(28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(29): ReLU(inplace=True)
(30): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(31): ReLU(inplace=True)
(32): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(33): ReLU(inplace=True)
(34): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(35): ReLU(inplace=True)
(36): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
)
(avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
(classifier): Sequential(
(0): Linear(in_features=25088, out_features=4096, bias=True)
(1): ReLU(inplace=True)
(2): Dropout(p=0.5, inplace=False)
(3): Linear(in_features=4096, out_features=4096, bias=True)
(4): ReLU(inplace=True)
(5): Dropout(p=0.5, inplace=False)
(6): Linear(in_features=4096, out_features=1000, bias=True)
)
)
```

Inception V3 (48 capas)

```
######################################################
# 1. Importar la red pre-entrenada Inception_V3
######################################################

model = models.inception_v3(pretrained=True)
print(model)

######################################################
#OUTOUT
######################################################

Inception3(
  (Conv2d_1a_3x3): BasicConv2d(
    (conv): Conv2d(3, 32, kernel_size=(3, 3), stride=(2, 2), bias=False)
    (bn): BatchNorm2d(32, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
  )
  (Conv2d_2a_3x3): BasicConv2d(
    (conv): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), bias=False)
    (bn): BatchNorm2d(32, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
  )
  (Conv2d_2b_3x3): BasicConv2d(
    (conv): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
```

```
  )
  (maxpool1): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  (Conv2d_3b_1x1): BasicConv2d(
    (conv): Conv2d(64, 80, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn): BatchNorm2d(80, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
  )
  (Conv2d_4a_3x3): BasicConv2d(
    (conv): Conv2d(80, 192, kernel_size=(3, 3), stride=(1, 1), bias=False)
    (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
  )
  (maxpool2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  (Mixed_5b): InceptionA(
    (branch1x1): BasicConv2d(
      (conv): Conv2d(192, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
    )
    (branch5x5_1): BasicConv2d(
      (conv): Conv2d(192, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn): BatchNorm2d(48, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
    )
    (branch5x5_2): BasicConv2d(
      (conv): Conv2d(48, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2), bias=False)
      (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
    )
    (branch3x3dbl_1): BasicConv2d(
      (conv): Conv2d(192, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
    )
    (branch3x3dbl_2): BasicConv2d(
      (conv): Conv2d(64, 96, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn): BatchNorm2d(96, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
    )
    (branch3x3dbl_3): BasicConv2d(
      (conv): Conv2d(96, 96, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn): BatchNorm2d(96, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
    )
    (branch_pool): BasicConv2d(
      (conv): Conv2d(192, 32, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn): BatchNorm2d(32, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (Mixed_5c): InceptionA(
    (branch1x1): BasicConv2d(
      (conv): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
    )
    (branch5x5_1): BasicConv2d(
      (conv): Conv2d(256, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn): BatchNorm2d(48, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
    )
    (branch5x5_2): BasicConv2d(
      (conv): Conv2d(48, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2), bias=False)
      (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
    )
    (branch3x3dbl_1): BasicConv2d(
      (conv): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
    )
    (branch3x3dbl_2): BasicConv2d(
      (conv): Conv2d(64, 96, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn): BatchNorm2d(96, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
    )
    (branch3x3dbl_3): BasicConv2d(
      (conv): Conv2d(96, 96, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn): BatchNorm2d(96, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
    )
    (branch_pool): BasicConv2d(
      (conv): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
.
.
```

```
.
  (Mixed_7c): InceptionE(
    (branch1x1): BasicConv2d(
      (conv): Conv2d(2048, 320, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn): BatchNorm2d(320, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
    )
    (branch3x3_1): BasicConv2d(
      (conv): Conv2d(2048, 384, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn): BatchNorm2d(384, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
    )
    (branch3x3_2a): BasicConv2d(
      (conv): Conv2d(384, 384, kernel_size=(1, 3), stride=(1, 1), padding=(0, 1), bias=False)
      (bn): BatchNorm2d(384, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
    )
    (branch3x3_2b): BasicConv2d(
      (conv): Conv2d(384, 384, kernel_size=(3, 1), stride=(1, 1), padding=(1, 0), bias=False)
      (bn): BatchNorm2d(384, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
    )
    (branch3x3dbl_1): BasicConv2d(
      (conv): Conv2d(2048, 448, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn): BatchNorm2d(448, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
    )
    (branch3x3dbl_2): BasicConv2d(
      (conv): Conv2d(448, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn): BatchNorm2d(384, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
    )
    (branch3x3dbl_3a): BasicConv2d(
      (conv): Conv2d(384, 384, kernel_size=(1, 3), stride=(1, 1), padding=(0, 1), bias=False)
      (bn): BatchNorm2d(384, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
    )
    (branch3x3dbl_3b): BasicConv2d(
      (conv): Conv2d(384, 384, kernel_size=(3, 1), stride=(1, 1), padding=(1, 0), bias=False)
      (bn): BatchNorm2d(384, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
    )
    (branch_pool): BasicConv2d(
      (conv): Conv2d(2048, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn): BatchNorm2d(192, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (dropout): Dropout(p=0.5, inplace=False)
  (fc): Linear(in_features=2048, out_features=1000, bias=True)
)
```

ResNet-101 (101 capas)

```
################################################
# 1. Importar la red pre-entrenada ResNet-101
################################################

model = models.resnet101(pretrained=True)
print(model)

################################################
#OUTOUT
################################################

ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
```

```
      (downsample): Sequential(
        (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
  )
  (layer2): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (3): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
  )
.
.
.
  (layer4): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
```

```
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=2048, out_features=1000, bias=True)
)
```

Anexo 3. Configuración de los modelos

Los modelos pre-entrenados Alexnet, VGG-19, Inception V3 y ResNet-101, tras ser importados, es preciso que sean definidos nuevamente como clasificadores, utilizando activaciones ReLU y el *dataset* de especies forestales maderables conformado. Se presenta cada modelo con su respectiva nueva configuración y *ouput* referente al cambio realizado.

AlexNet

```
###################################################################
# 2. Nueva red AlexNet
# el input_size coincide con las in_features del modelo pre-entrenado
###################################################################

classifier = nn.Sequential(
    OrderedDict(
        [
            ("dp", nn.Dropout(p=0.5)),
            ("fc1", nn.Linear(9216, 4096)),
            ("relu", nn.ReLU()),
            ("dp", nn.Dropout(p=0.5)),
            ("fc2", nn.Linear(4096, 1000)),
            ("relu", nn.ReLU()),
            ("fc3", nn.Linear(1000, 10)),
            ("output", nn.LogSoftmax(dim=1)),
        ]
    )
)

device = "cuda:0"
model = model.to(device)

# Ubicamos el classificador definido en la red cargada
model.classifier = classifier

# Guarda el aprendizaje de las primeras capas
for param in model.parameters():
    param.requires_grad = False
for param in model.classifier.parameters():
    param.requires_grad = True

print(classifier)

#####################################################
#OUTOUT
#####################################################

Sequential(
  (dp): Dropout(p=0.5, inplace=False)
  (fc1): Linear(in_features=9216, out_features=4096, bias=True)
  (relu): ReLU()
  (fc2): Linear(in_features=4096, out_features=1000, bias=True)
  (fc3): Linear(in_features=1000, out_features=10, bias=True)
  (output): LogSoftmax(dim=1)
)
```

VGG-19

```
###################################################################
# 2. Nueva red VGG-19
# el input_size coincide con las in_features del modelo pre-entrenado
###################################################################

classifier = nn.Sequential(
    OrderedDict(
        [
```

```
            ("dp", nn.Dropout(p=0.5)),
            ("fc1", nn.Linear(25088, 4096)),
            ("relu", nn.ReLU()),
            ("dp", nn.Dropout(p=0.5)),
            ("fc2", nn.Linear(4096, 1000)),
            ("relu", nn.ReLU()),
            ("fc3", nn.Linear(1000, 10)),
            ("output", nn.LogSoftmax(dim=1)),
        ]
    )
)

device = "cuda:0"
model = model.to(device)

# Ubicamos el classificador definido en la red cargada
model.classifier = classifier

# Guarda el aprendizaje de las primeras capas
for param in model.parameters():
    param.requires_grad = False
for param in model.classifier.parameters():
    param.requires_grad = True

print(classifier)

#######################################################
#OUTOUT
#######################################################

Sequential(
  (dp): Dropout(p=0.5, inplace=False)
  (fc1): Linear(in_features=25088, out_features=4096, bias=True)
  (relu): ReLU()
  (fc2): Linear(in_features=4096, out_features=1000, bias=True)
  (fc3): Linear(in_features=1000, out_features=10, bias=True)
  (output): LogSoftmax(dim=1)
)
```

Inception V3

```
#####################################################################
# 2. Nueva red Inception-V3
# el input_size coincide con las in_features del modelo pre-entrenado
#####################################################################

classifier = nn.Sequential(
    OrderedDict(
     [
         ("fc1", nn.Linear(2048, 10)),
         ("output", nn.LogSoftmax(dim=1))
         ])
)

device = "cuda:0"
model = model.to(device)

# Ubicamos el classificador definido en la red cargada
model.classifier = classifier

# Guarda el aprendizaje de las primeras capas
for param in model.parameters():
    param.requires_grad = False
for param in model.classifier.parameters():
    param.requires_grad = True

print(classifier)

#####################################################
#OUTOUT
#####################################################
```

```
Sequential(
  (fc1): Linear(in_features=2048, out_features=10, bias=True)
  (output): LogSoftmax(dim=1)
)
```

ResNet-101

```
######################################################################
# 2. Nueva red RexNet-101
# el input_size coincide con las in_features del modelo pre-entrenado
######################################################################

classifier = nn.Sequential(
    OrderedDict(
        [
            ("dp", nn.Dropout(p=0.5)),
            ("fc1", nn.Linear(2048, 1000)),
            ("relu", nn.ReLU()),
            ("dp", nn.Dropout(p=0.5)),
            ("fc2", nn.Linear(1000, 512)),
            ("relu", nn.ReLU()),
            ("fc3", nn.Linear(512, 10)),
            ("output", nn.LogSoftmax(dim=1)),
        ]
    )
)

device = "cuda:0"
model = model.to(device)

# Ubicamos el classificador definido en la red cargada
model.classifier = classifier

# Guarda el aprendizaje de las primeras capas
for param in model.parameters():
    param.requires_grad = False
for param in model.classifier.parameters():
    param.requires_grad = True

print(classifier)

######################################################
#OUTOUT
######################################################

Sequential(
  (dp): Dropout(p=0.5, inplace=False)
  (fc1): Linear(in_features=2048, out_features=1000, bias=True)
  (relu): ReLU()
  (fc2): Linear(in_features=1000, out_features=512, bias=True)
  (fc3): Linear(in_features=512, out_features=10, bias=True)
  (output): LogSoftmax(dim=1)
)
```

Anexo 4. Entrenamiento

El proceso de entrenamiento se llevo a cabo utilizando dos tipos de entradas de imágenes de hojas (segmentadas y no segmentadas), y establecieron criterios de aprendizaje orientados a la clasificación de especies forestales maderables.

El código en Python utilizado para el entrenamiento se divide en tres partes. La primera parte presenta las librerías utilizadas para los experimentos. La segunda, muestra el código para cargar el *dataset* del entrenamiento, validación y test. Finalmente, la tercera parte presenta la función entrenar con el uso del optimizador ADAM, la tasa de aprendizaje y el número de épocas, definidas.

```
############################################################
# Librerias Importadas
############################################################

import torch
from torchvision import datasets, transforms, models, utils
from torch import nn, optim
import torch.nn.functional as F
import torchvision
import numpy as np
from collections import OrderedDict
from torch.optim import lr_scheduler
from torch.autograd import Variable
import torch.nn as nn
import time
import json
import copy
import wandb
import os
from os.path import exists
from dataloader import device, data_transforms, pth, type, experiment, cross, wandb_on

############################################################
# Cargar dataset del entrenamiento, validacion y test
############################################################

train_dir = type + "dataset/train"
valid_dir = type + "dataset/validation"
test_dir = type + "dataset/test"
dirs = {"train": train_dir, "validation": valid_dir, "test": test_dir}

pretrained = Path(PRETRAINED_PATH)
trained = Path(TRAINED_PATH)

# Usar del dataset imagenes normalizadas y definidas

image_datasets = {
    x: DataLoaderClass(dirs[x], transform=data_transforms[x])
    for x in ["train", "validation", "test"]
}

dataset_sizes = {x: len(image_datasets[x]) for x in ["train", "validation", "test"]}
class_names = image_datasets["train"].classes

# Cargar archivo .JSON de categoria y nombre de especie

with open("dataset/cat_to_name.json", "r") as f:
    label_map = json.load(f)

dataloaders = {
    x: torch.utils.data.DataLoader(
        image_datasets[x], batch_size=64, shuffle=True, num_workers=32
    )
    for x in ["train", "validation", "test"]
}

############################################################
# Entrenamiento del modelo
```

```
############################################################

# Funccion entrenar
def train_model(model, criteria, optimizer, scheduler, num_epochs=22, device="cuda"):
    since = time.time()

    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    for epoch in range(num_epochs):
        print("Epoch {}/{}".format(epoch, num_epochs - 1))
        print("-" * 10)

        # Cada epoca hace un entrenamiento y validacion
        for phase in ['train', 'val']:
            if phase == 'train':

                model.train()  # Modelo de entrenamiento
            else:
                model.eval()   # Modelo de validacion

            running_loss = 0.0
            running_corrects = 0

            # Repetir los datos
            for inputs, labels in dataloaders[phase]:
                inputs = inputs.to(device)
                labels = labels.to(device)

                # Gradiente cero en los parametroszero the parameter gradients
                optimizer.zero_grad()

                # historial del entrenamiento
                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    _, preds = torch.max(outputs, 1)
                    loss = criteria(outputs, labels)

                    # backward + optimize only if in training phase
                    if phase == 'train':
                        #sched.step()
                        loss.backward()

                        optimizer.step()

                # estadisticas
                running_loss += loss.item() * inputs.size(0)
                running_corrects += torch.sum(preds == labels.data

        epoch_loss = running_loss / dataset_sizes[phase]
        epoch_acc = running_corrects.double() / dataset_sizes[phase]

            print('{} Loss: {:.4f} Acc: {:.4f}'.format(
                phase, epoch_loss, epoch_acc))

            # copia el modelo
            if phase == 'val' and epoch_acc > best_acc:
                best_acc = epoch_acc
                best_model_wts = copy.deepcopy(model.state_dict())

            if phase == "train":
                scheduler.step()

        print()

            time_elapsed = time.time() - since

    # carga los mejores resultados del modelo

    model.load_state_dict(best_model_wts)
    print("Training complete in {:.0f}m {:.0f}s".format(
            time_elapsed // 60, time_elapsed % 60))
    print("Best val Acc: {:4f}".format(best_acc))
```

```
    return model

# Se usa Criteria NLLLoss ya que la capa final es Softmax
 criteria = nn.NLLLoss().to(device)
# Para optimizar los parametros se usa ADAM
 optim = optim.Adam(model.parameters(), lr=0.00001)
# La tasa de aprendizaje LR varia en descenso 0.8 cada 3 epocas
 sched = lr_scheduler.StepLR(optim, step_size=3, gamma=0.8)
# Numero de epocas
 eps = 22
```

Anexo 5. Métricas

El siguiente código en Python se divide en dos partes principales. La primera parte, se refiere al cálculo de predicción de las redes en términos de de exactitud, precisión y el Valor-F1. Se importan las librerías SkLearn, WandB para el cálculo de los valores, la impresión de la matriz de confusión, y el seguimiento del entrenamiento a través de las curvas de ROC a tiempo real. La segunda parte, presenta el algoritmo de interpretabilidad visual.

```
##########################################
# calculo de metricas
# micro/macro: accuracy, recall, f1-score
##########################################

#librerias

from plotcm import plot_confusion_matrix, get_all_preds, calc_metrics, names
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.metrics import classification_report
import wandb

import torchvision
from torchvision import models
from torchvision import transforms
from captum.attr import IntegratedGradients
from captum.attr import GradientShap
from captum.attr import Saliency
from captum.attr import NoiseTunnel
from captum.attr import visualization as viz

#activar wandB para el seguimiento a tiempo real y curvas de ROC

if wandb_on is False:
    os.environ["WANDB_MODE"] = "dryrun"

wandb.init(project="d_leaf")
wandb.watch(model)

if __name__ == "__main__":
    if not trained.is_file():
        model = train_model(model, criteria, optim, sched, eps, device)
        torch.save(model.state_dict(), TRAINED_PATH)

    calc_accuracy(model, "test", device)

    #metricas

    print("\nAccuracy: {:.2f}\n".format(accuracy_score(preds, labels)))
    print(
        "Micro Precision: {:.2f}".format(
            precision_score(preds, labels, average="micro")
        )
    )
    print("Micro Recall: {:.2f}".format(recall_score(preds, labels, average="micro")))
    print("Micro F1-score: {:.2f}\n".format(f1_score(preds, labels, average="micro")))

    print(
        "Macro Precision: {:.2f}".format(
            precision_score(preds, labels, average="macro")
        )
    )
    print("Macro Recall: {:.2f}".format(recall_score(preds, labels, average="macro")))
    print("Macro F1-score: {:.2f}\n".format(f1_score(preds, labels, average="macro")))

    print(
        "Weighted Precision: {:.2f}".format(
            precision_score(preds, labels, average="weighted")
        )
    )
    print(
        "Weighted Recall: {:.2f}".format(
```

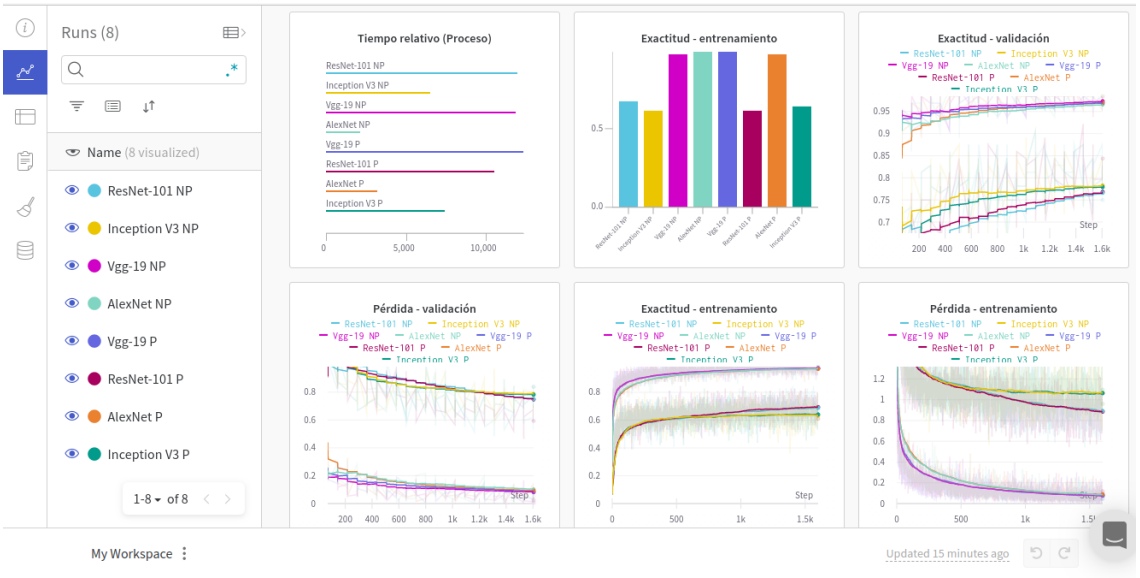
```
            recall_score(preds, labels, average="weighted")
        )
    )
    print(
        "Weighted F1-score: {:.2f}".format(f1_score(preds, labels, average="weighted"))
    )

##########################################
# Reporte de clasificación por el nombre de clases (especies)
##########################################

print("\nClassification Report\n")
    print(
        classification_report(
            preds,
            labels,
            target_names=[
                "Aniba rosaeodora",
                "Cedrela odorata",
                "Cedrelinga cateniformis",
                "Dipteryx micrantha",
                "Otoba glycycarpa",
                "Otoba parvifolia",
                "Simarouba amara",
                "Swietenia macrophylla",
                "Virola flexuosa",
                "Virola pavonis",
            ],
        )
    )

##########################################
# Matriz de confusión
##########################################

#Llamado a la matriz
preds, labels = get_all_preds(model, "test", device, dataloaders)
cm = calc_metrics(preds, labels)
print(cm)

# Ploteo de la matriz
plot_confusion_matrix(cm, names)

##########################################
# Interpretabilidad visual
##########################################

model = "vgg"
image = type + "dataset/test/Virola flexuosa/20200104_140656.jpg"

TRAINED_PATH = "d_train/" + pth + "/" + m + "_post.pth"

model.load_state_dict(torch.load(TRAINED_PATH))
model = model.eval()

transform = transforms.Compose(
    [
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]),
    ])

with open("dataset/cat_to_name.json") as json_data:
    idx_to_labels = json.load(json_data)

transformed_img = transform(img)
input = transformed_img.unsqueeze(0)

output = model(input)
output = torch.exp(output)
prediction_score, pred_label_idx = torch.topk(output, 1)

print("Predicted:", predicted_label, "(", prediction_score.squeeze().item(), ")")
```

```
# Create IntegratedGradients object and get attributes
integrated_gradients = IntegratedGradients(model)
attributions_ig = integrated_gradients.attribute(input, target=pred_label_idx, n_steps=200, internal_bat

noise_tunnel = NoiseTunnel(integrated_gradients)

# create custom colormap for visualizing the result
default_cmap = LinearSegmentedColormap.from_list(
    "custom blue", [(0, "#ffffff"), (0.25, "#000000"), (1, "#000000")], N=256)

# visualize the results using the visualize_image_attr helper method

transformed_img = transformed_img / torch.max(transformed_img)
_ = viz.visualize_image_attr_multiple(
    np.transpose(attributions_ig_nt.squeeze().cpu().detach().numpy(), (1, 2, 0)),
    np.transpose(
        transforms.ToTensor()(img).squeeze().cpu().detach().numpy(), (1, 2, 0)
    ),
    methods=["original_image", "heat_map"],
    signs=["all", "positive"],
    cmap=default_cmap,
    show_colorbar=True,
)
```

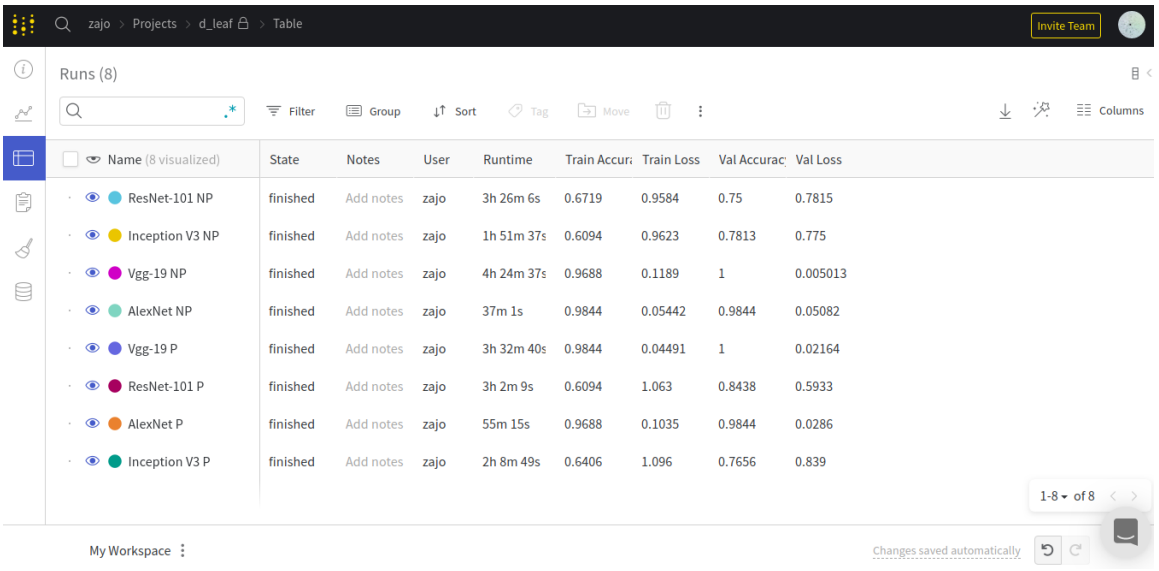
Anexo 6. Interfaz WandB

La plataforma *online* Weights & Biases permite el seguimiento a tiempo real del entrenamiento, validación y test nuestros modelos. Algunos de los servicios son:

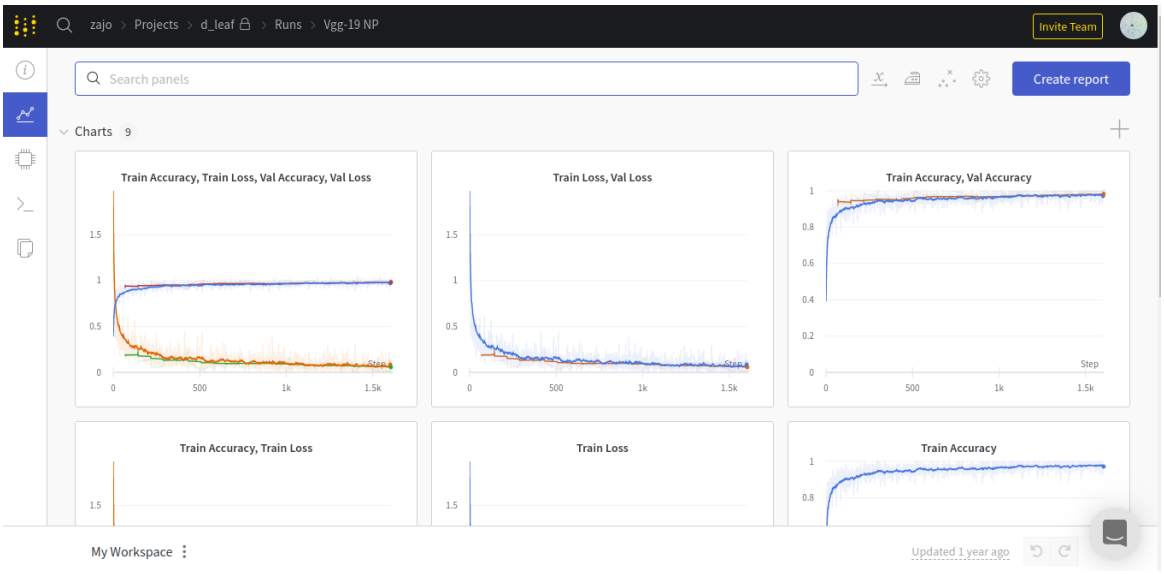
Permite realizar comparaciones a tiempo real y de manera sencilla.



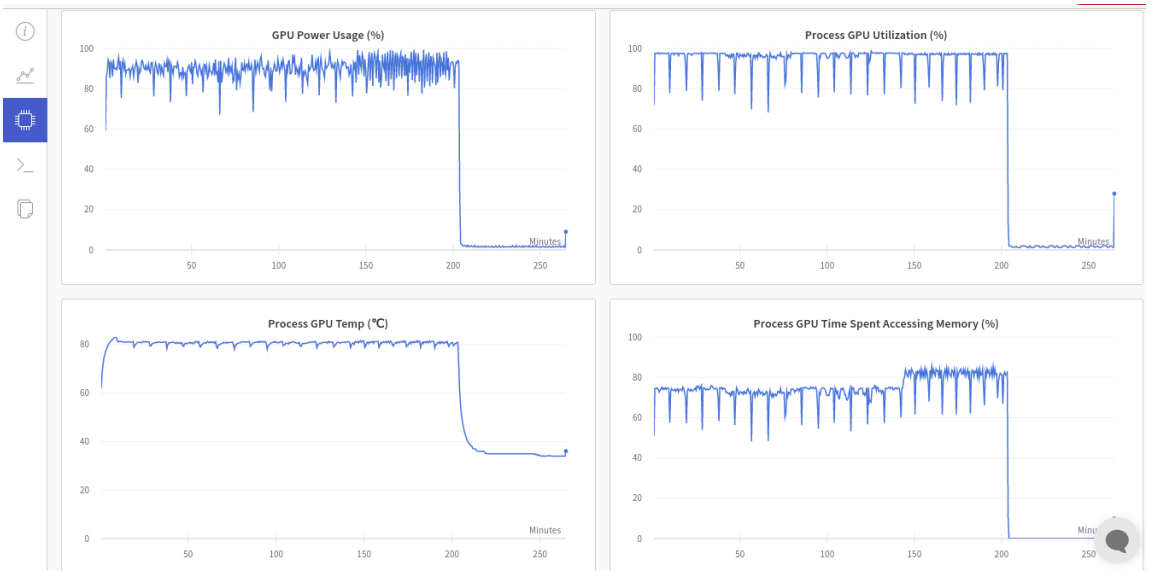
Proporciona un resumen tabulado sobre lo más relevante de nuestro modelo durante el aprendizaje.



WandB grafica curvas ROC según los parámetros que se le sean insertados.



También, muestra gráficos referentes al rendimiento de nuestra memoria, CPU, GPU, tráfico de red, energía y entre otros.



Finalmente, WandB guarda en versiones todo lo referente a nuestro modelo como el *output*.