Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales

Clasificación de especies en bosques utilizando imágenes capturadas con vehículos aéreos no tripulados y aprendizaje profundo

Tesis de Licenciatura en Ciencias de Datos

Catherine Sophie Louys Sanso

Director: Dr. Pablo De Cristóforis
Codirector: Ing. Francisco Raverta Capua
Buenos Aires, 2025

CLASIFICACIÓN DE ESPECIES EN BOSQUES UTILIZANDO IMÁGENES CAPTURADAS CON VEHÍCULOS AÉREOS NO TRIPULADOS Y APRENDIZAJE PROFUNDO

En este trabajo se aborda el desafío de clasificar especies arbóreas a partir de imágenes aéreas RGB mediante técnicas de aprendizaje profundo. El objetivo principal fue realizar segmentación semántica de copas de árboles utilizando tecnología accesible y de bajo costo, como drones equipados con cámaras RGB, enfrentando limitaciones propias de este tipo de datos, como la variabilidad en la iluminación, la oclusión parcial de copas y la similitud fenotípica entre especies.

Como parte de esta tesis se replicó el trabajo de Cloutier *et al.* [9], donde se utilizó una U-Net. A partir de dicha base, se implementaron modificaciones sobre la arquitectura original de la U-Net. Se implementaron 3 tipos de *schedulers* distintos, los cuales permitieron mejorar el desempeño de la red. El mejor F1 obtenido con la U-Net fue de 0.7392, superando el valor reportado en el *paper* original.

Además, se trabajó con otra red, una DeepLabV3 con ResNet-50 de *backbone*, que superó el desempeño de la anterior. Se probaron distintas configuraciones de esta red, también con los mismos *schedulers* que se utilizaron para la U-Net. El mejor resultado obtenido fue un *F1 score* de 0.7515 en promedio entre las especies.

Esta tesis se enmarcó dentro de un proyecto de mapeo de bosques en el *Laboratorio de Robótica y Sistemas Embebidos* (LRSE) del ICC-DC. Para la realización de este trabajo se utilizó el servidor del área de Imágenes y Robótica del ICC-DC, pues a través de él pudo accederse a dos GPUs que fueron de utilidad para reducir los tiempos de cómputo de entrenamiento de redes. Asimismo, la realización de esta tesis supuso el manejo de *Docker* [12], *software* para el encapsulamiento del entorno de ejecución en un contenedor. Para la implementación de las redes neuronales se utilizó la librería PyTorch [25]. El dataset utilizado consistió en imágenes reales tomadas en áreas forestales de Canadá.

Palabras claves: clasificación de especies arbóreas, segmentación semántica, U-Net, DeepLabV3, redes neuronales convolucionales, aprendizaje profundo.

TREE SPECIES CLASSIFICATION USING FOREST IMAGERY CAPTURED WITH UNMANNED AEREAL VEHICLES (UAVS) AND DEEP LEARNING

This work addresses the challenge of classifying tree species from aerial RGB images using deep learning techniques. The main objective was to perform semantic segmentation of tree canopies using accessible and low-cost technology, such as drones equipped with RGB cameras, while overcoming limitations inherent to this type of data, including illumination variability, partial occlusion of canopies, and phenotypic similarity between species.

As part of this thesis, the work done by Cloutier *et al.* [9], where a U-Net was used for the tree species classification in a forest in Canada, was replicated. Based on that approach, modifications were made on the original architecture. Three different types of schedulers were assessed, achieving comparable or even better performance than the results reported in the original paper. The best score was 0.7392, improving the one found in the original work.

Furthermore, another convolutional network was employed, DeepLabV3 with a ResNet-50 backbone. This network showed a higher F1 score than the previous models and configurations. As well as with the U-Net, the same three different schedulers were experimented on this network. The best average F1-score achieved was 0.7515.

This thesis was developed in a forest mapping project under the *Laboratorio de Robótica y Sistemas Embebidos* (LRSE), ICC-DC. For the making of this work, a server from the Imagery & Robotics area of the ICC-DC was used, allowing access to two GPUs that were crucial for lowering the computation time in model training. Moreover, the making of this thesis required knowledge in Docker [12] container management, as it helped encapsulating the runtime environment. In order to program the previously mentioned convolutional networks, PyTorch [25] library was used. The dataset analyzed was consisted of real RGB images of forest areas in Canada.

Keywords: tree species classification, semantic segmentation, U-Net, DeepLabV3, convolutional neural networks, deep learning.

AGRADECIMIENTOS

Quiero agradecerles a mi director Pablo y mi codirector Fran por acompañarme durante todo el proceso, y por leer muchísimas veces mi tesis. Gracias Fran por responder mis consultas a cualquier horario. Quiero agradecerle también a mi familia y a mis amigos por su apoyo. Este trabajo no habría sido posible sin todos ustedes.

A mi hermano Jean Paul, que es mi profe particular desde que me enseñó a sumar.

Índice general

1. INTRODUCCIÓN

1.1. Motivación

Los bosques son ecosistemas complejos que desempeñan un papel trascendental en la salud del planeta. Su contribución a la biodiversidad, la regulación del clima, la producción de oxígeno, el ciclo hidrológico, la conservación del suelo, el desarrollo económico y el bienestar humano subrayan la importancia crítica de su preservación para garantizar un futuro sostenible. Resulta de suma importancia entonces la gestión y el manejo efectivo de los recursos forestales.

Si bien las imágenes satelitales han demostrado su utilidad para el monitoreo del cambio de uso del suelo y la evaluación de la funcionalidad forestal a escala regional, su resolución espacial limita su aplicación en el análisis detallado a nivel predial. Por otro lado, el relevamiento de datos a través del trabajo de campo tradicional llevado adelante por ecólogos, agrónomos e ingenieros forestales, aunque proporciona información de alta precisión, presenta limitaciones significativas en términos de eficiencia costo-superficie y resulta impracticable para el monitoreo de grandes extensiones forestales.

En los últimos años, ha surgido un creciente interés a nivel global por la utilización de vehículos aéreos no tripulados (VANTs) como plataformas de sensores remotos para el monitoreo ambiental. Estos sistemas ofrecen una serie de ventajas sustanciales sobre las imágenes satelitales y los métodos de relevamiento manual, incluyendo la flexibilidad en el tiempo de revisita, una alta resolución espacial, la independencia de la cobertura nubosa y un menor costo y riesgo operativo. En Argentina, a pesar de la existencia de iniciativas puntuales, la adopción de esta tecnología en el ámbito forestal aún se encuentra en una etapa incipiente, lo que resalta la necesidad de impulsar su desarrollo y aplicación.

La presente investigación se centra en la problemática de la clasificación de especies arbóreas a partir de imágenes RGB obtenidas mediante el uso de drones, empleando técnicas de aprendizaje profundo. La motivación fundamental de este trabajo reside en la necesidad de desarrollar una metodología eficiente y precisa para la identificación de las especies que componen un determinado bosque. Esta capacidad de clasificación reviste una importancia crucial por diversas razones. En primer lugar, permite llevar a cabo un inventario detallado de la abundancia de cada especie, información esencial para la detección y gestión de especies invasoras en bosques nativos. Esto sirve a su vez para realizar una evaluación de la disponibilidad de madera para distintos usos, como por ejemplo: carbón, leña, aserraderos para muebleria, etcétera, dependiendo de la especie; y para la detección de especies con valor especial de conservación, por ejemplo, alerces milenarios. La implementación de un monitoreo periódico de la composición de especies a lo largo del tiempo podría proporcionar entonces información valiosa para evaluar el estado de preservación de un área forestal específica, facilitando la toma de decisiones informadas para su explotación sustentable, conservación y restauración; dependiendo el caso y el tipo de bosque con el que se trate. Asimismo, el conocimiento de la composición de especies se relaciona directamente con el estado de conservación y la gestión del bosque, considerando las diferentes susceptibilidades de las especies a factores como los incendios forestales.

En resumen, este trabajo de investigación busca contribuir al estudio y análisis de los ecosistemas forestales, con un enfoque particular en la identificación y clasificación

automatizada de especies arbóreas mediante la integración de la tecnología de drones como sensores remotos y el aprendizaje profundo para el procesamiento de los datos. Se espera que los resultados de esta investigación aporten herramientas valiosas para la gestión sostenible de los recursos forestales y la conservación de la biodiversidad.

1.2. Objetivos

El objetivo general de este trabajo de investigación es abordar el desafío de la clasificación de especies arbóreas a partir de imágenes aéreas, mediante la aplicación de técnicas de aprendizaje profundo. Esta tarea implica superar complejidades inherentes al procesamiento de imágenes aéreas, como la variabilidad en la iluminación, los ángulos de visión, la oclusión de copas, la similitud visual entre especies y la necesidad de extraer características discriminativas robustas.

Para cumplir con este objetivo se adoptó el enfoque basado en la segmentación semántica de imágenes aéreas RGB capturadas con drones, utilizando redes neuronales convolucionales profundas.

Entre los objetivos específicos, podemos mencionar: en primer lugar, la revisión del estado del arte y la búsqueda de conjuntos de datos (*datasets*) afines al problema de clasificación de bosques; en segundo lugar, la replicación del trabajo de Cloutier *et al.* [9] introduciendo mejoras a la red utilizada en dicho trabajo; y por último, la implementación de redes alternativas para mejorar aún más los resultados obtenidos.

1.3. Estructura de la tesis

En esta sección se describen los contenidos cubiertos por los siguientes capítulos.

En el Capítulo 2, se presenta un estudio sobre el estado del arte. Se aporta la base del trabajo y la literatura que se tuvo en cuenta. Se explican algunos conceptos básicos de las redes neuronales convolucionales así como también se mencionan trabajos previos del laboratorio realizados en la misma área de investigación, y trabajos de otros grupos de investigación también abocados a esta temática. En la segunda sección, se explican algunas arquitecturas de red utilizadas específicamente en el contexto de procesamiento de imágenes, y se enfatiza sobre las redes utilizadas en este trabajo en particular. Además, se delimitan las métricas para evaluar el desempeño de la red y el funcionamiento de distintas modificaciones a la red, como la introducción de *schedulers*.

En el Capítulo 3, se describe el procedimiento del trabajo y los materiales utilizados para realizarlo. Se divide en dos secciones. En la primera, se describe el conjunto de datos utilizado y las herramientas para procesarlo; y en la segunda se mencionan las distintas configuraciones de redes empleadas.

En el Capítulo 4, se grafican, discuten y comparan los resultados obtenidos en las distintas configuraciones de red para cada una de las redes utilizadas.

En el Capítulo 5, se brindan las conclusiones y resumidamente se nombran los objetivos cumplidos a lo largo del trabajo. Por último, se proponen líneas de investigación para continuar como trabajo futuro dentro del *Laboratorio de Robótica y Sistemas Embebidos* (LRSE), del ICC-DC.

2. ESTADO DEL ARTE Y MARCO TEÓRICO

En este capítulo se abordarán los trabajos previos realizados en el *Laboratorio de Robótica y Sistemas Embebidos* (LRSE) del ICC-DC, laboratorio en el cual se enmarca esta tesis. Además, se mencionarán otros trabajos en la misma área de investigación.

En las siguientes dos secciones, se explicarán algunos conceptos básicos de redes neuronales y redes neuronales convolucionales. Luego, se mencionarán las redes neuronales convolucionales más populares. Finalmente, se comentaran algunas herramientas utilizadas comúnmente en redes, ya sea para mejorar los resultados obtenidos o para acelerar la convergencia de la misma.

2.1. Análisis y clasificación de datos forestales

El *Laboratorio de Robótica y Sistemas Embebidos* (LRSE) del ICC-DC cuenta con trabajos previos en esta temática. En primer lugar, la obtención de datos reales etiquetados sigue siendo costosa y requiere de trabajo de clasificación manual, y es en este contexto que el LRSE desarrolló un simulador capaz de generar escenas forestales sintéticas, con el objetivo de evaluar si estas pueden ser utilizadas para entrenar redes neuronales profundas en tareas de segmentación semántica de nubes de puntos reales [27]. Además, se propuso una metodología simplificada para generar DTMs (en inglés, *Digital Terrain Models*) precisos a partir de datos fotogramétricos de escenas de bosques obtenidos con VANTs. En el trabajo, se comparan diferentes configuraciones de vuelo y estrategias de adquisición de datos, demostrando cómo distintas decisiones como la altura de vuelo y el tipo de cámara impactan significativamente en la calidad del modelo generado [24].

Más alla de los trabajos realizados en este laboratorio, diversos estudios han demostrado que la combinación de datos provenientes de diferentes sensores puede mejorar significativamente los resultados en la clasificación de especies forestales. En [11], se comparan múltiples algoritmos de clasificación aplicados a datos combinados de escaneo láser aerotransportado (ALS) y fotografías RGB. Este enfoque multimodal permitió alcanzar una mayor precisión en la identificación de especies a nivel de árbol individual, destacando el valor de integrar distintas fuentes de información y técnicas de análisis. La investigación subraya además la importancia de elegir algoritmos adecuados según las características de los datos y los objetivos del trabajo, evidenciando que la clasificación forestal efectiva requiere no solo datos de calidad, sino también estrategias computacionales robustas y adaptadas al contexto. En [33], se combina información obtenida a través de imágenes RGB y tecnología LiDAR (en inglés, *Light Detection and Ranging*). Se realizaron distintas combinaciones de las características obtenidas con un algoritmo de *Random Forest* para clasificar 10 tipos distintos de especies de árboles en áreas urbanas.

Dentro de la clasificación de especies arbóreas solamente a partir de imágenes RGB obtenidas mediante VANTs, una línea de trabajo se centra en la detección de copas individuales utilizando técnicas basadas en *bounding boxes* o cajas delimitadoras. Por ejemplo, [14] emplea redes convolucionales para clasificar especies sobre imágenes de alta resolución. De forma similar, en [2] se aplica un enfoque supervisado que combina detección de copas individuales mediante *bounding boxes* con la posterior identificación de especies, demostrando buenos resultados incluso en bosques heterogéneos. No obstante, estos métodos

presentan limitaciones en escenarios donde las copas se superponen o no están claramente definidas, lo que ha motivado el desarrollo de enfoques más detallados.

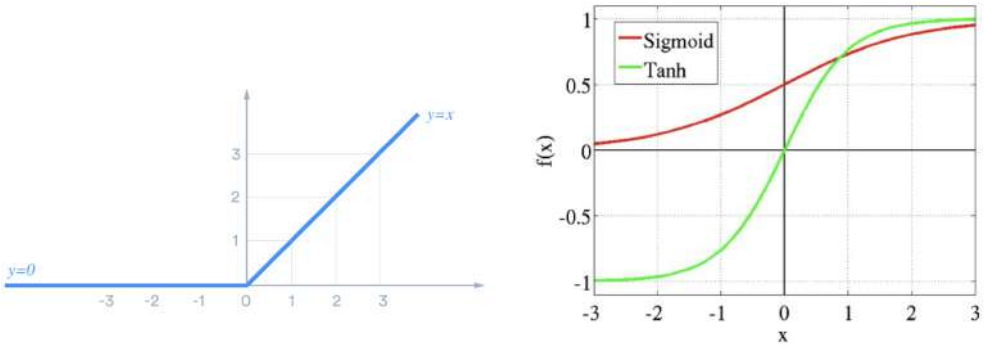
Una alternativa más precisa es la segmentación semántica, que permite delinear con mayor exactitud la forma de cada copa o estructura arbórea a nivel de píxel. En este sentido, [22] se enfoca en segmentar automáticamente una especie específica mediante una red profunda similar a la DeepLabV3+, destacando la capacidad de este tipo de modelos para identificar patrones morfológicos particulares. Por otro lado, [35] propone una arquitectura basada en Mask R-CNN mejorada, que permite la segmentación e identificación simultánea de múltiples especies en bosques mixtos, superando las limitaciones de los enfoques basados exclusivamente en detección. Finalmente, [6] introduce una variante del modelo Res-UNet, adaptada a imágenes aéreas, que logra una segmentación semántica precisa y una clasificación de especies robusta, reforzando el potencial de las arquitecturas profundas en contextos forestales complejos.

Sin embargo, pocos de estos trabajos son realizados en Sudamérica y en Argentina en particular: de los anteriores trabajos de clasificación enmarcados fuera del LRSE, [11] utiliza bosques de Japón, [14] de Alemania, [2] de Suiza, [35] y [6] de China, y solamente [22] utiliza bosques de Perú (en Sudamérica). Otro trabajo destacado en Sudamérica es [10], llevado a cabo en Brasil, que ilustra cómo estas tecnologías permiten detectar y modelar árboles invasores en bosques subtropicales, un desafío crítico para la conservación y manejo forestal regional. La anterior investigación resalta la efectividad del uso de imágenes UAV de alta resolución junto con algoritmos de aprendizaje automático para identificar especies con impactos ecológicos negativos, mostrando el potencial de estas metodologías para apoyar decisiones de gestión ambiental en áreas complejas y de difícil acceso.

El manejo efectivo de la variabilidad fenológica, es decir, los cambios que se producen en una especie físicamente a lo largo del año, ha sido otro foco importante en la generación de *datasets* para el entrenamiento de modelos de aprendizaje profundo. El trabajo [17] se distingue por la construcción de conjuntos de datos que incorporan imágenes capturadas en diferentes estaciones, permitiendo así reflejar los cambios fenológicos naturales de las especies arbóreas. Asimismo, [21] demuestra que la incorporación de imágenes obtenidas en distintos momentos del año permite entrenar modelos capaces de realizar tareas como el conteo de árboles, la segmentación de copas y la predicción de altura a escala nacional. Este trabajo destaca que incorporar esta diversidad estacional en los datos de entrada mejora significativamente la capacidad de los modelos para generalizar en contextos forestales con alta heterogeneidad espacial y temporal. Finalmente, [9] evidencia que los cambios en la coloración y textura de las hojas durante el período otoñal pueden modificar significativamente el rendimiento de los modelos de segmentación basados en redes neuronales profundas. A modo de aclaración, en esta tesis no se pretende replicar el trabajo completo de Cloutier *et al.* [9], sino la implementación de la red de segmentación semántica y el análisis de las imágenes tomadas en uno de los períodos.

2.2. Redes neuronales

Las redes neuronales surgieron como inspiración del funcionamiento del cerebro humano. En su concepción más básica, están compuestas por unidades llamadas neuronas artificiales, organizadas en capas: una capa de entrada, una o más capas ocultas y una capa de salida. Cada neurona recibe información, la procesa mediante una función matemática y transmite el resultado a otras neuronas conectadas.



(a) Función ReLU. *(b)* Función sigmoide y tanh.

Fig. 2.1: Funciones de activación. Imágenes extraídas de [32].

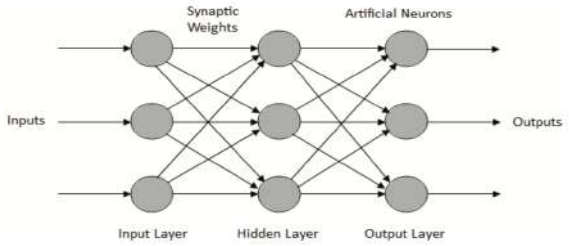


Fig. 2.2: Arquitectura de un MLP. Imagen extraída de [30].

Una de las arquitecturas más básicas y ampliamente utilizadas es la red neuronal multicapa o MLP (en inglés, *Multi-Layer Perceptron*). En este tipo de red, cada capa está completamente conectada con la siguiente, y se utilizan funciones de activación no lineales, como la ReLU, para introducir no linealidades que permiten a la red aprender relaciones complejas entre los datos. En la Fig. 2.1 se presentan algunas de las funciones de activación más populares y en la Fig. 2.2 se representa un gráfico de la arquitectura de la red MLP.

Durante el entrenamiento, las redes ajustan sus pesos internos mediante un optimizador que minimiza una función de pérdida (en inglés, *loss function*). Para esto se utiliza, por un lado, el método de retropropagación del error (en inglés, *backpropagation*) para estimar el gradiente con la regla de la cadena; y por otro lado, el método del descenso por el gradiente, para modificar los pesos en función del error cometido en la predicción.

Entre los optimizadores más populares se encuentra Adam (en inglés, *Adaptive Moment Estimation*) [20], que detallamos a continuación. Adam combina las ventajas de dos métodos de optimización populares: *Momentum* y *AdaGrad*. Utiliza promedios móviles sobre una ventana de primer y segundo orden de los gradientes para adaptar la tasa de aprendizaje de cada parámetro. El algoritmo tiene los siguientes hiperparámetros: α, que es la tasa de aprendizaje; β_1, que decaimiento exponencial para el promedio del gradiente; β_2, que es el decaimiento exponencial para el promedio del cuadrado del gradiente; y ϵ, que es un valor pequeño destinado a evitar divisiones por cero. Los parámetros son m_t, el vector del primer momento; v_t, el vector del segundo momento; y θ_t, que contiene los pesos del modelo.

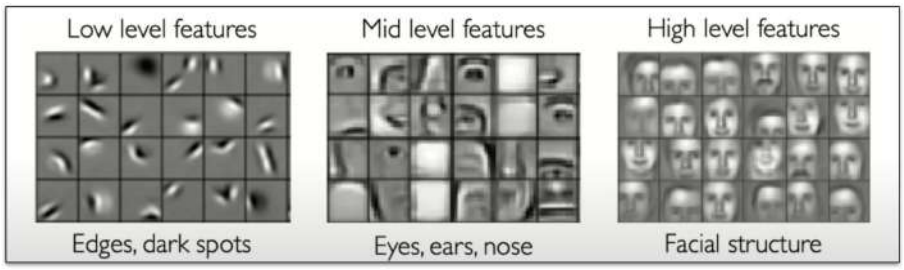


Fig. 2.3: Distintos tipos de características o *features*.

La forma propuesta en [20] para inicializar el algoritmo en $t = 0$ es con los hiperparámetros $\alpha = 0{,}001$, $\beta_1 = 0{,}9$, $\beta_2 = 0{,}999$ y $\epsilon = 10^{-8}$. Además, los parámetros m_t y v_t se inicializan en 0. El pseudocódigo de Adam es el siguiente:

Pseudocódigo del optimizador Adam

1: **Requiere:** α: tasa de aprendizaje; β_1 y $\beta_2 \in [0, 1)$, decaimientos exponenciales para las estimaciones de las derivadas; $f(\theta)$, la función objetivo con parámetro θ
2: **Para** cada iteración con $t = 1, 2, \ldots$
3: $g_t = \nabla_\theta f_t(\theta_{t-1})$
4: $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$
5: $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
6: $\hat{m}_t = \frac{m_t}{1-\beta_1^t}$
7: $\hat{v}_t = \frac{v_t}{1-\beta_2^t}$
8: $\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon}$

En la línea 3 del pseudocódigo se calcula el gradiente, en este caso mediante *backpropagation*. En la línea 4, se actualiza la estimación del primer momento o de la primera derivada. Lo mismo sucede para el segundo momento en la línea 5. Luego, en las restantes líneas se corrige, en orden, el sesgo para el primer y segundo momento, y se actualizan los parámetros de la red.

2.3. Redes neuronales convolucionales

Las redes neuronales convolucionales (o CNNs, del inglés *Convolutional Neural Networks*) son una clase especial de redes neuronales diseñadas para trabajar con datos que tienen una estructura espacial, como las imágenes. Las CNNs utilizan filtros o *kernels* que recorren la entrada para detectar patrones locales. Son altamente eficaces en tareas de procesamiento de imágenes y visión por computadora, como clasificación de imágenes, detección de objetos y segmentación semántica.

Estas redes están compuestas por distintos tipos de capas: las más características son las capas convolucionales, que realizan operaciones de filtrado sobre regiones locales de la imagen, y las capas de *pooling*. Gracias a esta estructura, las capas iniciales tienden a detectar patrones simples como bordes o texturas, mientras que las capas más profundas pueden capturar estructuras más complejas y específicas, como formas u objetos completos (Fig. 2.3).

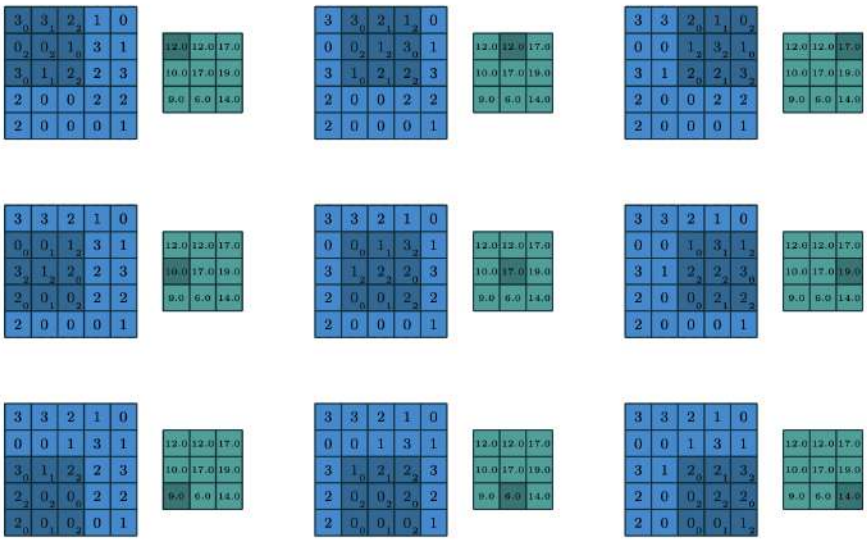


Fig. 2.4: Pasos de una convolución con *stride* = 1. Imagen extraída de [13]. En azul se presenta una matriz de tamaño 5×5. Superpuesto en la matriz en azul oscuro se encuentra el filtro de 3×3. En verde, se grafica el resultado obtenido en la convolución, y en verde oscuro el elemento que está siendo calculado para un paso en particular.

2.3.1. Operaciones básicas en redes neuronales convolucionales

En redes convolucionales, una convolución consiste en aplicar un filtro (o *kernel*) sobre una imagen o un mapa de activación para extraer características. En la Fig. 2.4 se presenta cómo se realiza la operación de convolución al desplazar el filtro sobre la entrada usando un parámetro llamado *stride*, que indica cuántos píxeles se mueve el filtro en cada paso, y se multiplica al sector de la entrada en el que se encuentra. Aclaración: si bien hay una diferencia entre convolución y *cross-correlation*, se utilizó el término convolución para ser consistente con la literatura.

Una convolución en dos dimensiones con filtro de tamaño impar y *stride* = 1 está representada por la Ecuación 2.1. Las variables i y j corresponden al número de la fila y columna de la imagen original X donde se está centrando el filtro, es decir, indican al elemento de X que se multiplica por el elemento central del filtro. Luego, Y es la imagen resultado y K el filtro de $L \times M$. Las variables l y m recorren las filas y columnas del filtro respectivamente, para aplicar la convolución.

$$Y[i,j] = \sum_{l=0}^{L-1} \sum_{m=0}^{M-1} X[i+l, j+m] K[l,m] \tag{2.1}$$

Una consecuencia de aplicar una convolución es la disminución del tamaño espacial de la salida, a excepción de que se utilice un *kernel* de 1×1 con *stride* = 1. Si el *stride* es mayor a 1, el tamaño espacial se reduce aún más. Una doble convolución aplica dos capas convolucionales consecutivas y suele tener funciones de activación, como ReLU, entre las dos capas. Las CNNs aprenden los pesos del *kernel* mediante retropropagación o *backpropagation*.

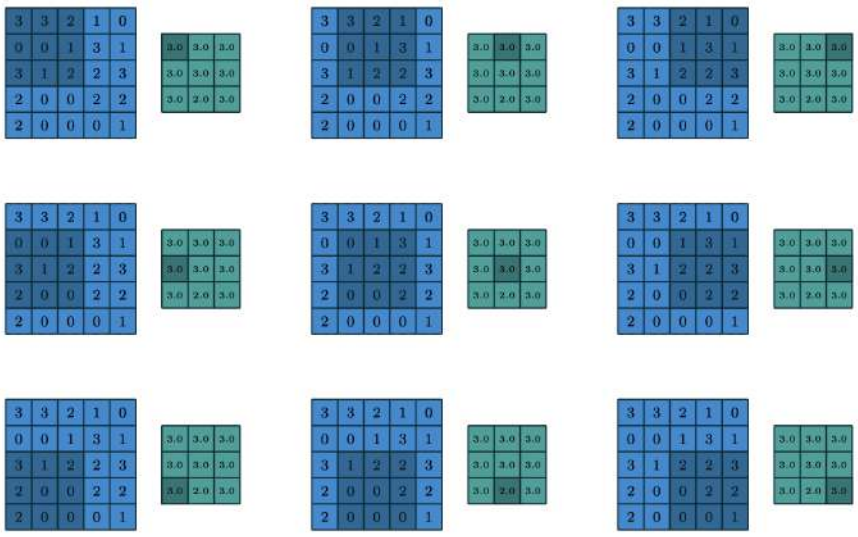


Fig. 2.5: Pasos de una operación de *max pooling* con *stride* = 1. Imagen extraída de [13].

Es común en redes neuronales convolucionales encontrar también operaciones de *downsampling* y *upsampling*. El *downsampling* reduce la resolución espacial de una imagen, y se logra comúnmente mediante operaciones como *pooling* o convoluciones. Esta operación permite disminuir el tamaño de los datos, capturar patrones más generales y reducir el costo computacional, aunque también implica pérdida de información espacial.

Pooling es una técnica particular de *downsampling*, a través de la cual se obtiene un número representativo de un sector de la imagen. Este número representativo se obtiene considerando un criterio particular, como el promedio (en inglés, *average*) o el máximo. Por ejemplo, en el *max pooling*, se divide el mapa de activación en regiones y se conserva únicamente el valor máximo de cada una. En la Fig. 2.5 puede verse el proceso de *max pooling* paso a paso. Se presenta en azul una matriz de tamaño 5×5. En azul oscuro, se delimita la región de 3×3 de la cual se está calculando para un paso particular el número representativo, que será el máximo valor. En verde, se grafica el resultado obtenido en la operación de *pooling*.

Si una convolución reduce la dimensión espacial pero se pretende obtener una salida del mismo tamaño que la entrada, se debe utilizar *padding*. Se le llama *padding* a la adición de valores extra, habitualmente ceros (en ese caso, *zero-padding*), alrededor de la imagen o mapa de características (Fig. 2.6). Al añadir este reborde, se permite que el filtro se aplique también en los bordes, manteniendo así las dimensiones espaciales originales y asegurando que los píxeles del borde se utilicen tanto como los del centro.

Por otro lado, el *upsampling* aumenta la resolución espacial de una representación. Esto es fundamental en arquitecturas como las U-Net (ver Sección 2.4.1 para más detalles), donde es necesario recuperar la forma original de la imagen para producir una salida de la misma dimensión. El *upsampling* se puede realizar con una interpolación o con capas de convoluciones transpuestas.

Las llamadas convoluciones transpuestas (o *transposed convolutions*) aumentan la resolución aprendiendo cómo distribuir la información a lo largo de un espacio mayor. En la Fig. 2.7 se muestran los primeros 4 pasos de una convolución transpuesta sobre una matriz de 3×3 (en azul). Se rodeó cada elemento de la matriz original con ceros, formando una

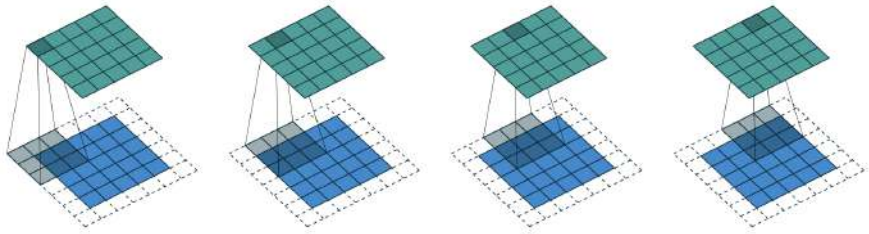


Fig. 2.6: Convolución con *stride* = 1 y con *padding* de tamaño 1 (líneas punteadas) para cada uno de los bordes de una matriz de 5×5 (azul). La matriz resultante se observa en verde. Imagen extraída de [13].

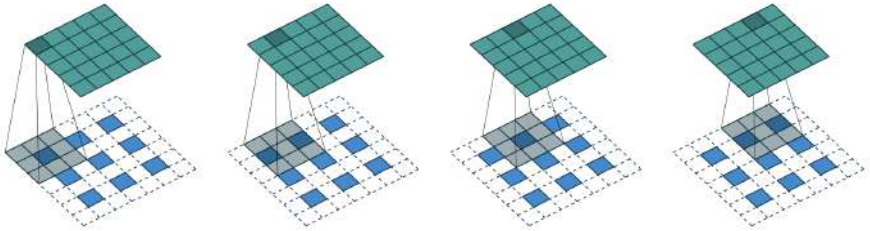


Fig. 2.7: Primeros pasos de una convolución transpuesta. Imagen extraída de [13].

imagen de 7×7, y se utilizó un filtro de 3×3 con *stride* = 2 para generar la matriz de 5×5.

La interpolación por otro lado, no utiliza parámetros aprendidos sino que estima los valores faltantes basándose en los píxeles vecinos. Un ejemplo es la interpolación bicúbica, que calcula cada nuevo píxel como una combinación de los 16 píxeles más cercanos. Se realiza típicamente primero una interpolación cúbica [19] en el eje x, y luego otra en el eje y.

2.4. Redes neuronales convolucionales para la segmentación semántica

La segmentación semántica es una técnica del procesamiento de imágenes que consiste en clasificar cada píxel en función de la categoría a la que pertenece. A diferencia de otros problemas como la clasificación de imágenes, donde se asigna una sola etiqueta a toda la imagen, o la detección de objetos, donde se localizan y etiquetan instancias individuales con *bounding boxes*, la segmentación semántica es píxel a píxel.

En segmentación semántica, todos los píxeles que pertenecen a una misma clase reciben la misma etiqueta, sin hacerse una distinción entre individuos de la misma clase. El resultado final es una máscara de segmentación, que es una imagen con la misma resolución que la imagen original donde cada píxel está etiquetado con su clase correspondiente.

Entre las principales redes que permiten realizar esta tarea de clasificación se encuentran las *Fully Convolutional Networks* (FCN), pioneras en segmentación semántica [29]; y SegNet, que además de ser una FCN, posee un bloque codificador y un decodificador, e introduce un manejo novedoso del *upsampling* que permite recuperar resolución perdida en las operaciones de *pooling* de extracción de características [1].

Luego, se incluye U-Net [28] y DeepLab [7]. La U-Net es una red que combina un codificador y un decodificador simétricos conectados mediante *skip connections*, que se detallará

más adelante (ver Sección 2.4.1). DeepLab es un modelo de segmentación semántica desarrollado por Google Research que combina redes convolucionales profundas con técnicas que mejoran la resolución espacial de las predicciones, usando una arquitectura basada en VGG-16 como extractor de características, y reemplaza las capas convolucionales estándar por convoluciones dilatadas. Para mejorar la precisión de los bordes de los objetos segmentados, DeepLab incorpora un CRF (*Conditional Random Field*) que ajusta los contornos según la información de color y cercanía en la imagen, como se detallará más adelante (ver Sección 2.4.3).

PSPNet (*Pyramid Scene Parsing Network*) es una arquitectura diseñada para mejorar la segmentación semántica capturando el contexto global de la imagen [36]. Su principal innovación es el módulo de pirámide de pooling (*Pyramid Pooling Module*), que permite extraer información a múltiples escalas espaciales. Gracias a esta estrategia, PSPNet logra una segmentación más precisa en escenas complejas y con objetos de tamaños variados.

También se emplean variantes como DeepLabV3, sucesora de la DeepLab, que utiliza convoluciones dilatadas e incorpora módulos de otros tipos de *pooling* [8] para capturar información contextual de distintos tamaños.

Existen otras redes conocidas como Faster R-CNN que, si bien están diseñadas para detección de objetos y no para segmentación semántica, pueden adaptarse mediante variantes como Mask R-CNN [15]. Por otro lado, redes como ResNet suelen funcionar como *backbones* en otras arquitecturas, como DeepLab o FCN. El *backbone* de una red es el bloque destinado a la extracción de características o *features*. En algunas ocasiones se lo utiliza preentrenado, para lograr una convergencia más rápida de la red al reducir la función de pérdida (o *loss function*).

En el último tiempo, la aparición de novedosas arquitecturas como SegFormer, que combina un codificador basado en *transformers* jerárquicos con un decodificador pequeño sin necesidad de utilizar *positional embeddings*, mejoraron resultados de segmentación semántica tanto en precisión como en eficiencia computacional [34]. En esta tesis no se llegaron a considerar este tipo de redes pero se proponen como trabajo futuro.

En esta tesis se trabajó con una UNet y una ResNet como backbone de una red DeepLabV3.

2.4.1. U-Net

La U-Net es una arquitectura de red neuronal convolucional diseñada específicamente para tareas de segmentación semántica a nivel de píxel y fue propuesta originalmente por O. Ronneberger *et al.* [28] para segmentar imágenes biomédicas. Su nombre proviene de la forma de "U" que describe su estructura simétrica: un bloque de compresión (codificador o *encoder*) y otro de expansión (decodificador o *decoder*) unidos por *skip connections*.

La Fig. 2.8 muestra la arquitectura de una U-Net. Del lado izquierdo se encuentran las capas del *encoder*, del lado derecho las del *decoder*. En el medio, el punto más bajo de la "U", se ve el *bottleneck*. Las *skip connections* están marcadas en gris.

El bloque de *encoder* captura el contexto de la imagen. Su función es extraer las características a medida que se reduce la resolución espacial, incorporando información contextual. Es un bloque de *contracción*, ya que el tamaño de la imagen se va reduciendo en cada paso, mientras aumenta su profundidad, pues aumenta la cantidad de canales.

Dentro del *encoder*, se encuentran bloques más pequeños, cada uno de los cuales está compuesto por dos capas convolucionales de 3×3, para procesar los mapas de características y detectar patrones visuales; funciones de activación ReLU, para introducir una no

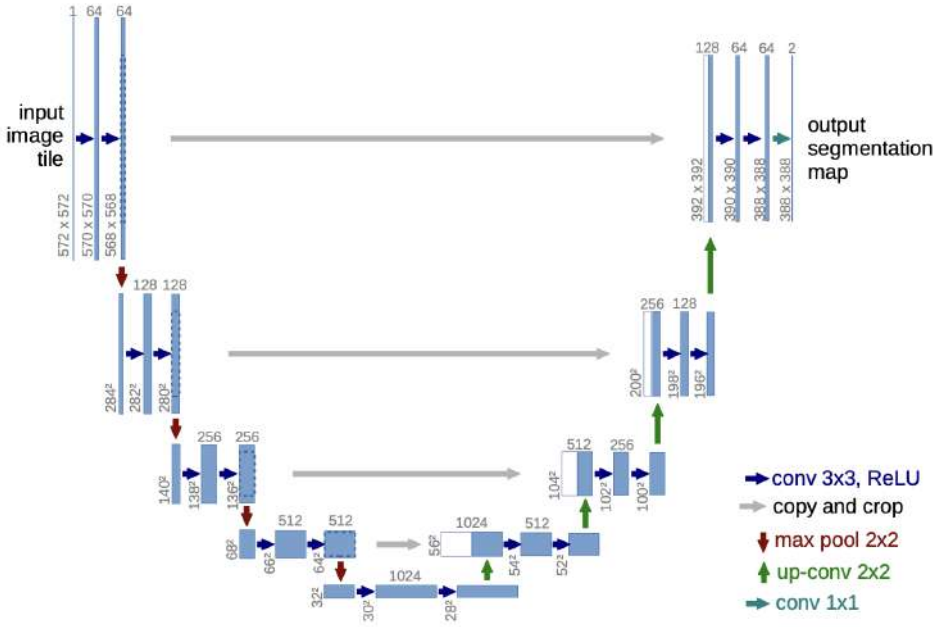


Fig. 2.8: Arquitectura de la U-Net en su implementación original. Imagen extraída de [28].

linealidad y permitir que la red aprenda relaciones más complejas; y una capa de *max pooling*, para achicar la resolución espacial a la mitad y retener las características más fuertes. Al finalizar la parte del *encoder* de la red, se obtiene una representación comprimida de las *features* de la imagen.

Para conectar al *encoder* con el *decoder*, la red emplea un cuello de botella o *bottleneck*. En este punto se realiza también una doble convolución de 3×3 con funciones ReLU de activación y se llega a la cantidad máxima de canales.

El bloque que continúa es el del *decoder*. El objetivo de este segmento de la red es aumentar la resolución e ir recuperando los detalles espaciales perdidos en las capas anteriores. Se divide en bloques, cada uno de los cuales posee una capa de *upsampling*, seguido de una *skip connection*, una doble convolución, y una función ReLU de activación. La capa de *upsampling* sirve para duplicar el tamaño espacial del mapa de características, reduciendo a la mitad la cantidad de canales.

En la literatura se refiere a una *skip connection* como una concatenación entre el mapa de características obtenido en el *encoder* y el obtenido en el *decoder* con esa misma resolución. De esta forma se recuperan detalles que el *encoder* perdió en las operaciones de *pooling*, permitiendo una reconstrucción más precisa de las formas. Hay cuatro de estas conexiones, entre el mapa obtenido luego de realizar cada doble convolución en el *encoder* y antes de realizar cada una en el *decoder*.

La salida de la última capa de la U-Net es una máscara de segmentación que asigna una clase a cada píxel de la imagen original. Para esto, se utiliza una convolución de 1×1 que reduce la profundidad o el número de canales a la cantidad de clases o categorías posibles. Cada píxel está representado como un vector, donde en la i-ésima posición se obtiene la probabilidad de que ese píxel pertenezca a la i-ésima clase. Para obtener esas probabilidades, se emplea normalmente una función *softmax* o una *binary cross entropy*, para las clasificaciones multiclase y binarias respectivamente. Cuando se utiliza *softmax*

para la clasificación de múltiples especies, la salida de la red para una imagen será un tensor con tantos canales como especies. En cada uno de estos canales, se representa, píxel a píxel, la probabilidad de que dicho píxel pertenezca a la clase (especie) correspondiente a ese canal. Esto proporciona una distribución de probabilidad por píxel, donde la suma de las probabilidades para cada píxel a través de los canales es igual a 1.

Como previamente se mencionó, la U-Net se creó para segmentar células en imágenes de microscopía biomédica y no se utilizaba *padding* en las capas convolucionales. Entonces, en cada convolución de 3×3 del *encoder* se redujo el tamaño espacial del mapa de características. La consecuencia de esto es que el tamaño del mapa se reduce con cada capa de convolución, además de reducirse por las operaciones de *pooling*. Luego, se debe hacer un arreglo para que las dimensiones de los mapas las capas del *encoder* coincidan con las del *decoder* a la hora de concatenarlas con una *skip connection*. En el diseño original se recortaron los mapas de características del *encoder* para así operar con información de la imagen sin añadir un reborde de ceros, es decir, sin hacer *padding*.

Ahora bien, en la tarea de segmentación semántica, trabajar sin *padding* puede conllevar problemas. La salida de la red puede ser mucho más chica que la imagen original (como puede verse en las anotaciones en gris de las dimensiones de la salida de la U-Net en la Fig. 2.8) y entonces no cubrirla completamente, dejando sin segmentar los bordes de la imagen. Luego, se realiza *padding* para abordar la segmentación semántica con la U-Net.

2.4.2. ResNet-50

ResNet-50 [16] es una red neuronal convolucional profunda, propuesta por He, K. *et al* (Fig. 2.9). Su principal innovación, junto al de otras redes de la misma familia de ResNets, es el uso de bloques residuales. Estos bloques permiten entrenar redes muy profundas sin que el rendimiento se degrade y evitando problemas como el desvanecimiento del gradiente, que sucede cuando el gradiente de la función de pérdida se vuelve extremadamente pequeño a medida que se propaga hacia atrás a través de las capas iniciales de la red durante el entrenamiento.

ResNet fue diseñada y evaluada para tareas de clasificación de imágenes en datasets como ImageNet. Su concepto de diseño residual fue tan influyente que desde entonces ha sido adoptada en casi todas las arquitecturas modernas, incluyendo tareas de detección y segmentación, por ejemplo.

Antes de ResNet, aumentar la profundidad o la cantidad de canales en una red llevaba a una mayor tasa de error, por las dificultades al propagar el gradiente hacia las capas iniciales. ResNet soluciona este problema permitiendo que el flujo de información y de gradientes "salte" capas a través de conexiones residuales. Estas conexiones son las *skip connections*, que se mencionaron previamente al describir la arquitectura de la U-Net.

La ResNet-50 en particular cuenta con 50 capas, divididas en 5 etapas. En cada etapa se aplica *batch normalization* luego de cada convolución. Se explicará esta herramienta más adelante en el Apartado 2.5.1.

En la etapa inicial, se realiza una convolución de 7×7 con $stride = 2$ y $padding = 3$. Luego, se aplica la función ReLU para añadir no linealidades, y se realiza un *max pooling* de 3×3 con $stride = 2$ para extraer *features* o características.

En las siguientes cuatro etapas (denominadas en el trabajo original [16] como "conv2_x" a "conv5_x"), se repite un bloque residual una determinada cantidad de veces, dependiendo del número de etapa. Para la etapa 2 y la 5, se repite 3 veces. Para la etapa 3, se repite

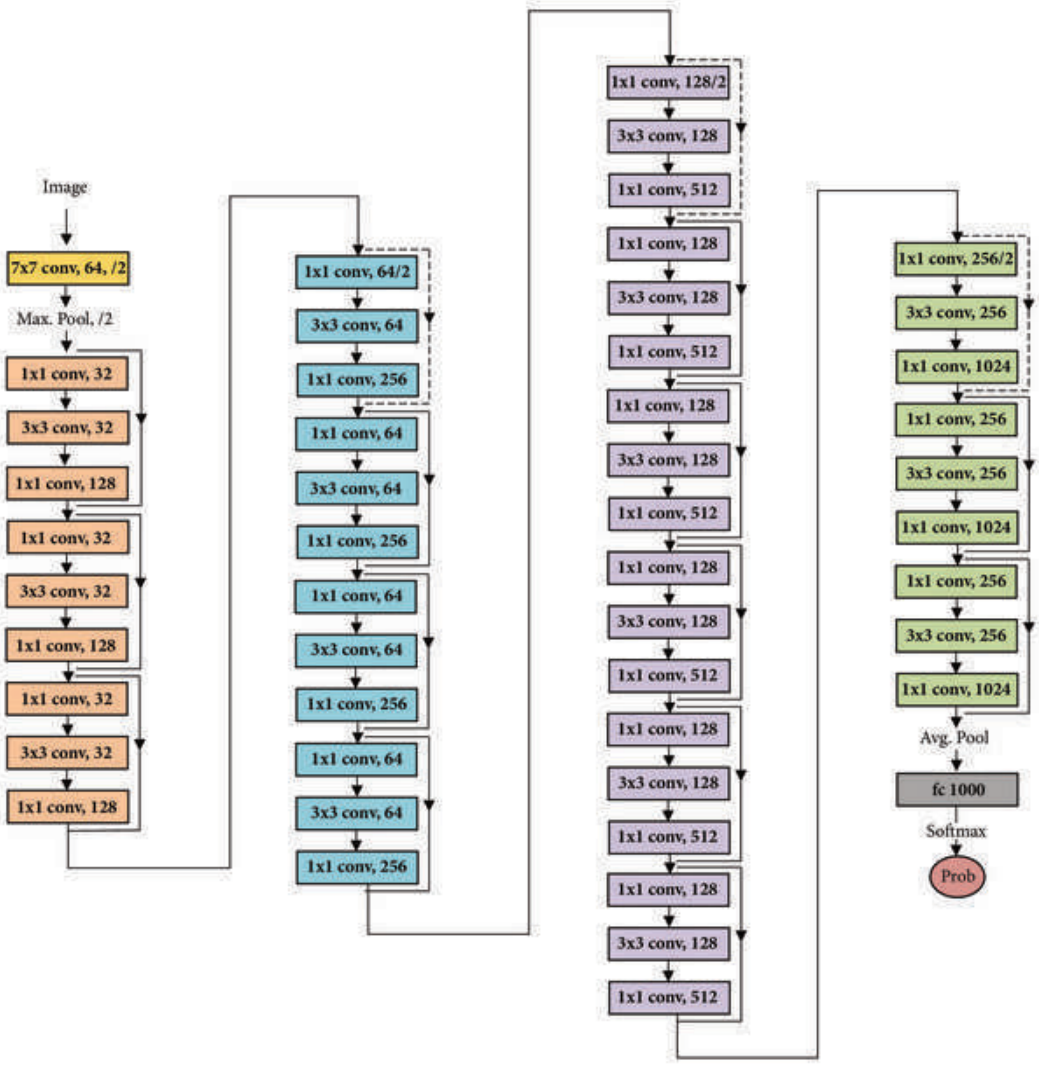


Fig. 2.9: Arquitectura de una ResNet-50. Imagen extraída de [4].

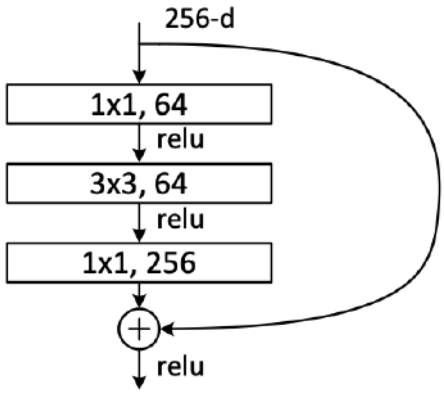


Fig. 2.10: Arquitectura del bloque residual en una ResNet-50. Imagen extraída de [16].

4 veces. Por último, para la etapa 4 se repite 6 veces.

Este bloque residual mencionado previamente consiste en 3 convoluciones: una convolución de 1×1 para reducir la dimensionalidad de la imagen, otra de 3×3 para procesar y obtener las *features* y otra de 1×1 para restaurar la imagen a su cantidad de canales original. Luego de cada convolución, excepto en la última de cada bloque, se agrega una no linealidad con ReLU.

Cada bloque residual (o *bottleneck block*, Fig. 2.10) suma su entrada a la salida del bloque convolucional a través de una *skip connection*. Cuando la dimensión cambia, se aplica una convolución 1×1 en el salto para adaptar dimensiones antes de la suma.

Para finalizar, en esta red se realiza un *average pooling* global, seguido de una *fully connected layer* y *softmax* para obtener probabilidades de clasificación. La función final de *softmax* permite obtener una predicción basada en probabilidades para la clasificación.

2.4.3. DeepLabV3 con ResNet-50 de *backbone*

DeepLabV3 es una red de segmentación semántica que se basa en un esquema de tipo *encoder*, pero sin un *decoder* explícito. Fue diseñada específicamente para capturar información contextual a múltiples escalas, lo cual es esencial en tareas donde los objetos pueden tener tamaños muy variados dentro de una imagen.

Para extraer las características de la imagen, se utiliza una arquitectura ResNet-50 como *backbone*. La ResNet-50 se modifica reemplazando algunas operaciones de *down-sampling* por convoluciones dilatadas (o *atrous convolutions*) en los bloques finales. Este cambio permite mantener la resolución espacial de los mapas de activación, a la vez que se amplía el campo receptivo de cada neurona, lo cual facilita una mejor captura de información contextual sin incrementar significativamente la cantidad de parámetros ni el costo computacional. Respecto de una convolución tradicional, se agregan filas y columnas de valor 0 entre los valores del filtro, y luego se lo aplica a una matriz (Fig. 2.11). En la Fig. 2.12 se presentan tres ejemplos de convoluciones dilatadas. Además, se controla un parámetro clave denominado *output stride*, que representa la relación entre el tamaño de la imagen de entrada y el de la imagen de salida. Reducir el *output stride* permite mantener más resolución espacial, aunque a costa de mayor costo computacional.

Otra modificación clave es la incorporación del módulo ASPP o *Atrous Spatial Pyramid*

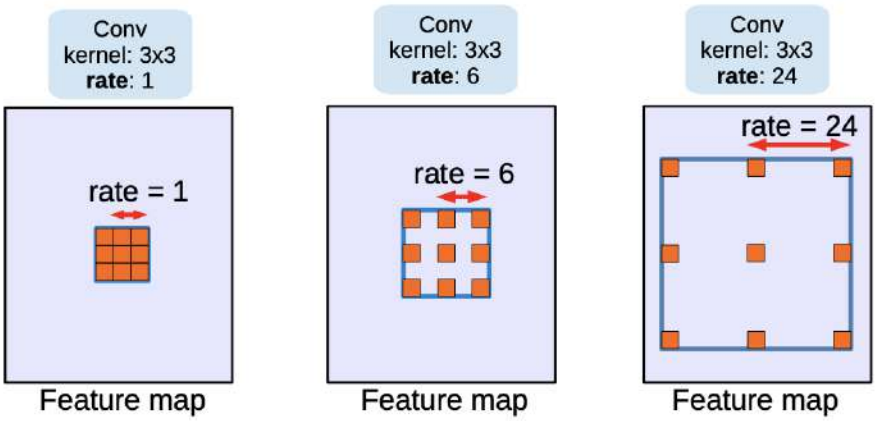


Fig. 2.11: Atrous convolutions. El parámetro *rate* controla la cantidad de columnas o filas de ceros entre uno y otro elemento del filtro. Imagen extraída de [8].

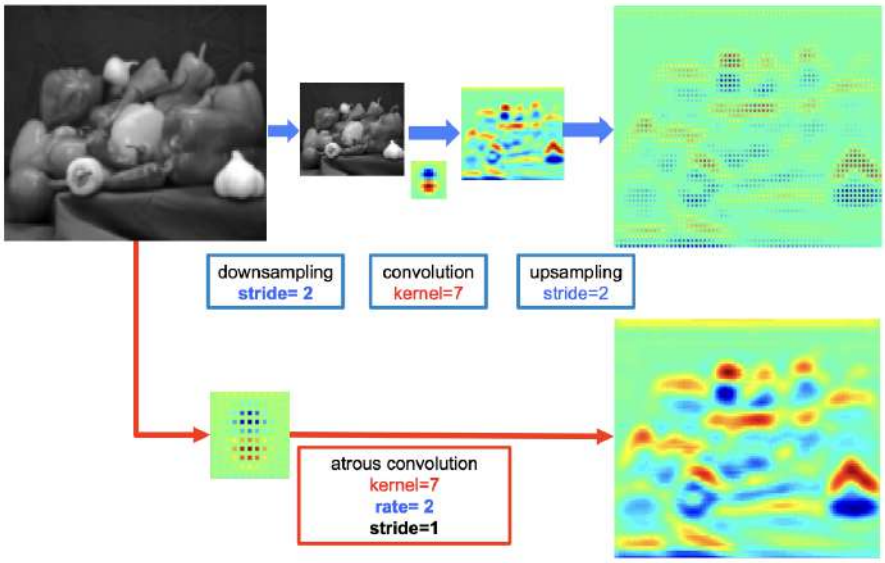


Fig. 2.12: Comparación entre una convolución dilatada y una operación de *downsampling* seguido de una convolución y una operación de *upsampling*. Imagen extraída de [7].

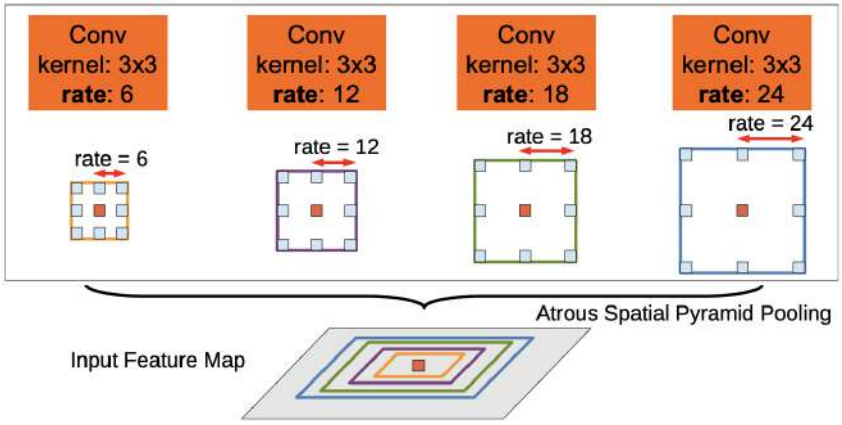


Fig. 2.13: Esquema de una ASPP para una DeepLab. Se presentan 4 convoluciones distintas que forman el módulo ASPP de esta red. Imagen extraída de [7].

Pooling (Fig. 2.13), que aplica múltiples convoluciones dilatadas en paralelo, cada una con un *rate* o tasa de dilatación distinta. Esto permite capturar patrones visuales a diferentes escalas. Además, se incluye una operación de *global average pooling* para introducir contexto global. Las salidas de todos estos caminos paralelos se concatenan y procesan para generar el mapa de segmentación final.

2.5. Herramientas adicionales para el ajuste de redes neuronales

En esta sección se mencionan algunas herramientas que en algunos casos pueden llegar a mejorar el desempeño de una red.

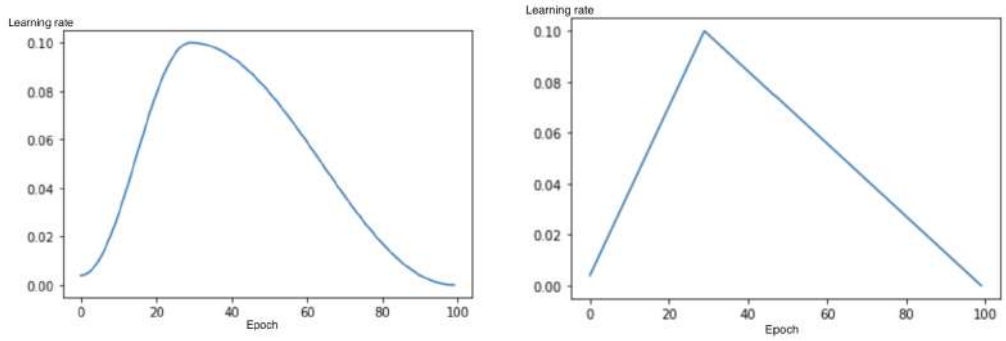
2.5.1. Normalización del *dataset* y *batch normalization*

La normalización del conjunto de datos consiste en transformar los datos de entrada para que sus características tengan una media cercana a cero y una desviación estándar cercana a uno. Esta práctica evita que al entrenar la red las distintas escalas de las características afecten el proceso de optimización y la convergencia.

Para normalizar se calcula la media y la desviación estándar para cada canal por separado para las imágenes de entrenamiento. Para un canal fijo, se suman los valores de los píxeles de todas las imágenes en cada canal en particular y se divide por el total de píxeles para obtener la media. Además, se calcula la desviación estándar de todos esos valores.

Por otro lado, *batch normalization* es una técnica que se aplica dentro de la arquitectura de la red neuronal para normalizar la salida de las funciones de activación en cada *batch* durante el entrenamiento. Denominamos *batch* al conjunto de imágenes que la red procesa simultáneamente en una iteración. *Batch normalization* ajusta y escala las activaciones obtenidas después de cada capa para que mantengan una media y varianza estables a lo largo del entrenamiento. Esto acelera la convergencia, reduce la sensibilidad a la inicialización de los pesos y puede actuar como una forma de regularización que ayuda a prevenir el sobreajuste.

Ambas técnicas permiten una mayor estabilidad y una mejor generalización del modelo.



(a) Estrategia coseno. *(b)* Estrategia lineal.

Fig. 2.14: Gráfico de la tasa de aprendizaje a lo largo de las épocas usando un `OneCycleLR()` con distintas estrategias de actualización. Imágenes extraídas de [3].

2.5.2. *Schedulers*

Los *schedulers* son algoritmos que ajustan la tasa de aprendizaje durante el entrenamiento del modelo para mejorar su convergencia y precisión.

Estos objetos en PyTorch son conectados al optimizador, quien es el encargado de ajustar los parámetros del modelo basándose en el cálculo del gradiente para minimizar la función de pérdida o *loss function*. El optimizador más utilizado en PyTorch es Adam.

El *scheduler* entonces, en cada época actualiza la tasa de aprendizaje o *learning rate* de acuerdo a una regla predefinida.

A continuación se definen algunos de los *schedulers* nativos de PyTorch:

2.5.2.1. `OneCycleLR()`

Este *scheduler* ajusta la tasa de aprendizaje en "un solo ciclo", como sugiere [31]. Sube rápidamente la tasa al máximo, y luego baja lentamente durante el resto del entrenamiento. El objetivo es que inicialmente utilice tasas grandes y luego se vaya estabilizando con tasas de aprendizaje más pequeñas para converger rápidamente. Entre los parámetros de este *scheduler* se encuentra `max_lr`, que define la tasa máxima de aprendizaje alcanzada; `epochs` y `steps_per_epoch`, que definen la duración del ciclo; y `anneal_strategy`, que puede ser lineal o coseno, según cómo se pretende que se actualice la tasa a lo largo de las épocas (Fig. 2.14).

2.5.2.2. `ReduceLROnPlateau()`

Este *scheduler* reduce la tasa de aprendizaje cuando una métrica no mejora después de cierta cantidad de épocas. A esa cantidad se la denomina *patience* (paciencia, en inglés). La métrica puede ser tanto una métrica como el F1, o la función de pérdida en el conjunto de validación, por ejemplo. De esta forma, si el modelo no mejora lo suficiente, la tasa de aprendizaje va disminuyendo hasta converger. Posee un parámetro denominado *factor*, donde se indica cuánto reducir la tasa en caso de que no mejore lo suficiente la métrica; un parámetro de `threshold`, que indica cuánto debe mejorar la métrica para que la tasa no disminuya, y `mode`, que indica si la métrica indicada debe disminuir o aumentar en la cantidad indicada por `threshold`. En el caso de la función de pérdida o *loss function*, por ejemplo, se buscaría disminuirla, mientras que en el caso de F1 se buscaría aumentarla.

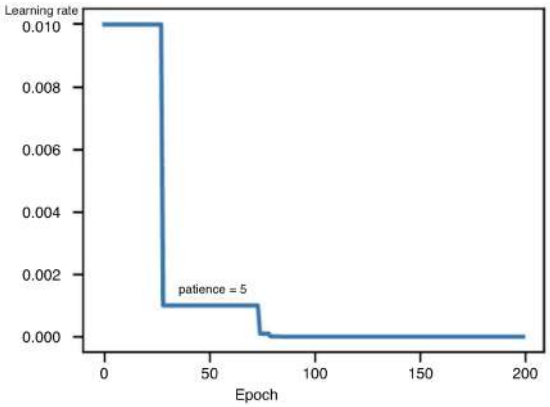


Fig. 2.15: Actualización de la tasa de aprendizaje a través de las épocas con un `ReduceLROnPlateau()`. Imagen extraída de [5].

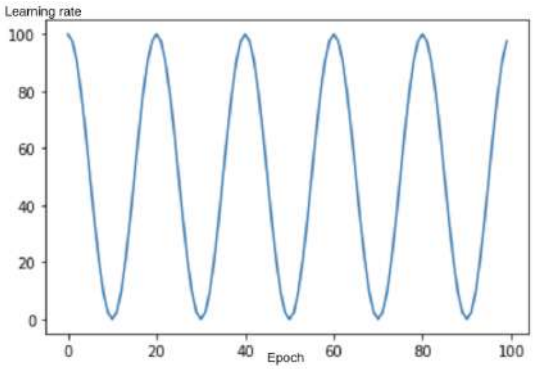


Fig. 2.16: Gráfico de la tasa de aprendizaje a lo largo de las épocas usando un `CosineAnnealingLR()`. Imagen extraída de [3].

En la Fig. 2.15 se visualiza un gráfico de la tasa de aprendizaje a lo largo de las épocas, con `patience` = 5.

2.5.2.3. `CosineAnnealingLR()`

`CosineAnnealingLR()` modula la tasa de aprendizaje siguiendo una función coseno entre un valor máximo y un mínimo. La tasa de aprendizaje según las épocas es entonces una función periódica (véase Fig. 2.16). Este *scheduler* puede ayudar a escapar de mínimos locales que no son lo suficientemente buenos. Entre los parámetros que toma este *scheduler* se encuentra `T_max`, que indica el número de pasos o épocas para completar un ciclo coseno; y `eta_min`, que indica la tasa de aprendizaje mínima.

2.5.3. *Dropouts*

Dropout [18] es una técnica de regularización para redes neuronales que ayuda a prevenir el sobreajuste. Durante el entrenamiento, el dropout consiste en desactivar aleatoriamente una fracción de las neuronas de la red en cada iteración. Esto impide que las neuronas se vuelvan demasiado dependientes unas de otras, promoviendo representaciones más robustas y reduciendo la varianza del modelo. En la fase de evaluación se utilizan

todas las neuronas.

Además, la función `nn.Dropout2d()` de PyTorch aplica *dropout* específicamente a entradas de imágenes de cuatro dimensiones (tamaño del *batch*, cantidad de canales, alto de la imagen, ancho de la imagen). A diferencia del dropout estándar, esta función anula canales completos (es decir, pone en cero mapas de activación enteros) con una probabilidad p durante el entrenamiento, lo que fuerza a la red a no depender de características específicas en ciertos canales. El parámetro p representa la probabilidad de desactivación de cada canal. A este *dropout* es al que se referirá el siguiente capítulo.

3. MATERIALES Y MÉTODOS

En este Capítulo se describe la base de datos utilizada para realizar este trabajo, así como también el preprocesamiento y las características de las redes utilizadas para la clasificación de especies arbóreas. Además, se darán a conocer las herramientas tecnológicas utilizadas a lo largo del proceso.

3.1. Obtención y preprocesamiento de datos

Si bien el objetivo original era clasificar árboles de zonas forestales de Argentina, la falta de *datasets* de imágenes de bosques argentinos y la falta de tiempo para el etiquetado manual de imágenes para el entrenamiento que involucra la generación de *datasets* propios imposibilitó la clasificación de las especies de nuestro país. Luego, se utilizó un *dataset* canadiense (Fig. 3.1), y se siguió el camino trazado por [9].

Este *dataset* contiene distintas carpetas de imágenes y máscaras, ya que el trabajo original se basa en la comparación del desempeño (o *performance*) en clasificación de una red neuronal a lo largo de las distintas estaciones del año. En esta tesis se utilizó la carpeta que mejores resultados obtuvo en las métricas del trabajo original [9], es decir, las imágenes correspondientes al 2 de septiembre de 2021.

De la carpeta mencionada se utilizaron siete archivos. El *dataset* cuenta con tres zonas de donde fueron extraídas las imágenes, por lo que tres de los archivos son ortomosaicos, tres son máscaras y el último delimita la *zona de inferencia*. Los ortomosaicos, uno por cada zona (de 39935×41984, 38911×32699 y 26624×26596 píxeles), son mapas de alta precisión creados al combinar múltiples imágenes aéreas ortocorregidas para crear una imagen con alta precisión de la superficie total terrestre relevada. Las imágenes son corregidas de forma tal que parecen ser tomadas desde una vista cenital perfecta, sin distorsiones de perspectiva o relieve. Los siguientes tres archivos utilizados corresponden a las máscaras de cada zona. Cada uno de estos archivos contiene un conjunto de polígonos que, una vez superpuestos con el ortomosaico o la imagen pertinente, permiten delimitar las porciones de la imagen pertenecientes a cada especie reconocida.

A la hora de evaluar la *performance* de una red neuronal en un conjunto de datos, es importante que no se produzca *data leakage*: esto es, que los *sets* de evaluación, validación y entrenamiento estén contaminados y contengan, por ejemplo, imágenes compartidas entre los distintos conjuntos. En caso de que se repitieran imágenes en el conjunto de evaluación y en el de entrenamiento, probablemente se obtendría un resultado sobrevalorado de cómo funciona la red clasificadora de especies. Con el objetivo de que esto no sucediera, y siguiendo la implementación del trabajo original, una de las tres zonas fue destinada solamente a la evaluación. De las otras dos, se delimitaron algunas regiones para formar parte también del conjunto de evaluación, y así obtener muestras de todas las especies en la evaluación. Estas regiones fueron obtenidas a través del séptimo y último archivo utilizado del *dataset* original, la *zona de inferencia* (Fig 3.2.a). De las imágenes extraídas no pertenecientes a la evaluación, el 80 % fue elegido aleatoriamente y destinado al entrenamiento, mientras que el restante 20 % se destinó a la validación de los modelos.

La forma que se utilizó en el trabajo original replicado para obtener imágenes a través del ortomosaico fue dividirlo en recortes o *tiles* no superpuestas de 256×256 píxeles. Luego,

Fig. 3.1: Zona de donde fueron obtenidos los datos. Imagen extraída de [9].

se implementó un programa en Python que permitió cortar un ortomosaico en imágenes de un tamaño en específico, y dividirlas según si pertenecían a la *zona de inferencia* o no. Para obtener una primera visualización de los datos, se utilizó la aplicación gratuita QGIS [26] (véase Fig 3.2).

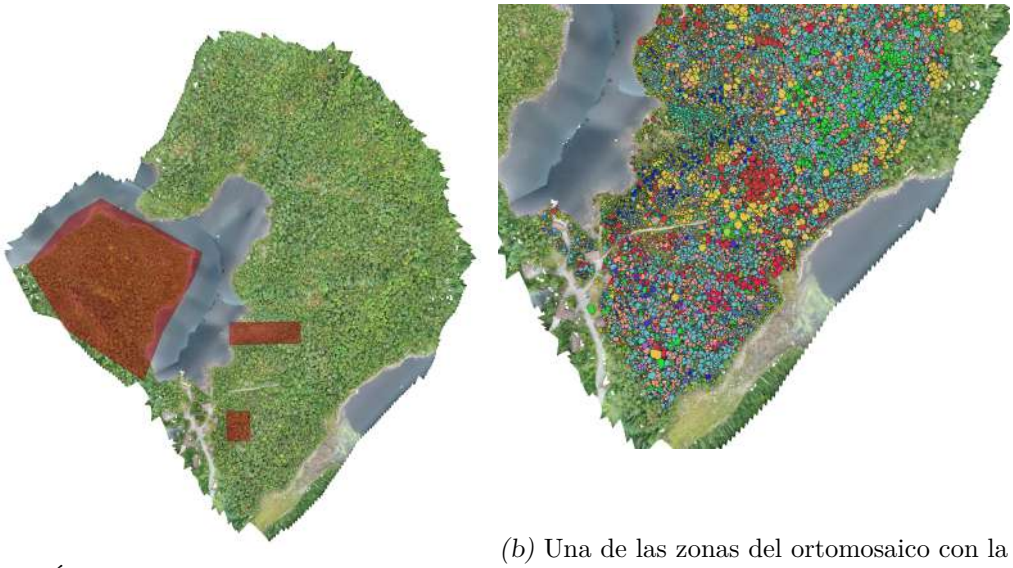
Además, se tuvieron otras consideraciones a partir del *paper* original. En primer lugar, se descartaron aquellas *tiles* cuya área estaba cubierta en menos de un 25 % por los polígonos de una máscara. Esto se hizo para que las distintas especies tuvieran una mayor proporción de píxeles en el *dataset*. Si se hubieran dejado todas las *tiles* independientemente de si tenían máscaras asociadas o no, probablemente habría habido una gran cantidad de píxeles de fondo, sin los árboles a identificar. En ese caso, una red que solamente supiera distinguir al fondo de la imagen respecto de los árboles podría haber tenido una *performance* muy alta según alguna métrica, pero no cumpliría el objetivo para el cual fue entrenada; es decir, no necesariamente distinguiría entre las distintas especies de árboles.

En segundo lugar, se agruparon algunas especies bajo un mismo género, pues eran muy similares entre sí y difíciles de distinguir. Un género agrupa a varias especies que comparten características similares y están estrechamente relacionadas, mientras que una especie es el grupo básico que incluye individuos capaces de reproducirse entre sí y tener descendencia fértil. Por ejemplo, en el género *Acer* existen distintas especies, como *Acer rubrum* y *Acer saccharum*, que son diferentes tipos de robles y se agruparon en el *dataset* para tener una mejor representación de la clase a identificar.

Por último, la red neuronal entrenada se limitó a reconocer entre 14 especies y genéros de árboles, con lo cual algunas especies menos representadas en el *dataset* fueron agrupadas junto al fondo de la imagen; es decir, la porción de la imagen donde ninguna especie fue reconocida.

En la Tabla 3.1 se mencionan las etiquetas finales luego del preprocesado del *dataset*. Además, se divisa, por cada etiqueta, el género y las especies a los que podría pertenecer cada píxel bajo esa etiqueta. Los casilleros marcados con * indican que habría varias opciones. Por ejemplo, un árbol muerto (bajo la etiqueta “Mort”) podría pertenecer a cualquier especie, incluso fuera de las demás contenidas en la tabla.

A partir del código, se obtuvieron tres imágenes por cada porción del *dataset* (Fig. 3.3). La primera consiste en el recorte del ortomosaico. La segunda corresponde al recorte



(a) Área total analizada con la zona de inferencia marcada en rojo.

(b) Una de las zonas del ortomosaico con la máscara por encima, coloreado según la clase.

Fig. 3.2: Visualizaciones del *dataset* estudiado en QGIS [26].

de la máscara de esa parte del ortomosaico. La última, en cambio, corresponde al recorte de la máscara y el ortomosaico, la primera por encima del segundo. Esta última imagen se generó para corroborar que hubieran sido correctamente recortadas las máscaras y que concordaran con el fragmento del ortomosaico.

En total, se obtuvieron 12752 imágenes solamente de porciones del ortomosaico para validación y entrenamiento, y 2744 imágenes para evaluación; conformando un total de 15496. Las Tablas 3.2 y 3.3 muestran la cantidad de píxeles por clase en el conjunto de imágenes de entrenamiento y validación, y en el de evaluación respectivamente. A través de ellas se aprecia que la clase mayoritaria en cantidad de píxeles es el fondo, y que las clases están desbalanceadas. En la última columna de las tablas se aprecia el porcentaje de píxeles de cada clase en particular sobre el total. Se considera que la diferencia de porcentajes por especie entre los distintos conjuntos no es mucha, con lo cual este factor no presentaría demasiados conflictos en la generalización de datos. La máxima diferencia absoluta es de 8.09 % en la clase Populus.

3.2. Configuraciones y uso de las redes seleccionadas

El entrenamiento de las redes fue realizado con la utilización del servidor del área de Imágenes y Robótica del ICC/DC, que posee dos GPUs RTX 3090 que disminuyeron ampliamente los tiempos de ejecución. Se realizó una conexión SSH a este servidor para poder aprovechar estas unidades de procesamiento gráfico. Se implementaron las redes en la versión 2.2.2 de PyTorch, instalada en un contenedor de Docker, junto con la versión 12.1 de CUDA, también instalada en el contenedor, para poder vincularse a las GPUs.

Los archivos de código fueron ejecutados en un contenedor aislado a través de Docker [12]. Docker es un *software de containerización* que resultó muy útil para este trabajo: permitió encapsular todo el entorno de ejecución; incluyendo dependencias, librerías y configuraciones específicas dentro de un contenedor. De esta manera, se aseguró que el

Etiqueta en el *dataset*	**Género**	**Especies**
ABBA	*Abies*	*Abies balsamea*
ACPE	*Acer*	*Acer pensylvanicum*
ACRU	*Acer*	*Acer rubrum*
ACSA	*Acer*	*Acer saccharum*
BEAL	*Betula*	*Betula alleghaniensis*
BEPA	*Betula*	*Betula papyrifera*
FAGR	*Fagus*	*Fagus grandifolia*
LALA	*Larix*	*Larix laricina*
Mort	*	*
PIST	*Pinus*	*Pinus strobus*
Picea	*Picea*	*Picea glauca*, *Picea mariana* y *Picea rubens*
Populus	*Populus*	*Populus grandidentata* y *Populus tremuloides*
THOC	*Thuja*	*Thuja occidentalis*
TSCA	*Tsuga*	*Tsuga canadensis*
Fondo	*	*

Tab. 3.1: Tabla taxonómica.

(a) Porción del ortomosaico. *(b)* Porción de la máscara **en escala de grises**. *(c)* Porción del ortomosaico con la máscara por encima.

Fig. 3.3: Ejemplo de recorte de una *tile*. Se utilizó una paleta de grises para reconocer las distintas clases, pudiendo de esta forma utilizar máscaras que sean imágenes de un solo canal. Esto permitió achicar los tiempos de cómputo.

Clase	N° de clase	Cantidad de píxeles	Porcentaje de píxeles (%)
ABBA	0	25,548,503	3.06
ACPE	1	10,768,198	1.29
ACRU	2	145,455,601	17.40
ACSA	3	46,899,382	5.61
BEAL	4	12,217,880	1.46
BEPA	5	201,815,112	24.15
FAGR	6	5,017,639	0.60
LALA	7	5,488,355	0.66
Mort	8	4,844,058	0.58
PIST	9	46,065,170	5.51
Picea	10	9,986,224	1.19
Populus	11	96,624,438	11.56
THOC	12	16,902,487	2.02
TSCA	13	1,435,936	0.17
Fondo (*background*)	14	206,646,089	24.73
Total	-	835,715,072	100.00

Tab. 3.2: Cantidad de píxeles por clase en los conjuntos de entrenamiento y validación.

Clase	N° de clase	Cantidad de píxeles	Porcentaje de píxeles (%)
ABBA	0	12,856,992	7.15
ACPE	1	3,216,226	1.79
ACRU	2	31,091,648	17.29
ACSA	3	7,762,257	4.32
BEAL	4	7,711,719	4.29
BEPA	5	33,897,965	18.85
FAGR	6	4,404,702	2.45
LALA	7	1,374,595	0.76
Mort	8	2,054,906	1.14
PIST	9	1,514,561	0.84
Picea	10	6,255,915	3.48
Populus	11	6,235,259	3.47
THOC	12	6,471,914	3.60
TSCA	13	1,103,097	0.61
Fondo (*background*)	14	53,879,028	29.96
Total	-	179,830,784	100.00

Tab. 3.3: Cantidad de píxeles por clase en el conjunto de evaluación.

comportamiento del *software* fuera idéntico tanto en el entorno local (la computadora que se conecta al servidor) como en el servidor remoto, eliminando problemas de compatibilidad de versiones de *software*, por ejemplo. Además, la utilización de Docker facilita la portabilidad y replicabilidad del proyecto. Al contenerse los archivos aislados, se mejoró la seguridad del sistema, pues los archivos fueron ejecutados en entornos aislados sin afectar otros procesos del servidor.

Para cada una de las redes y configuraciones mencionadas a continuación, se realizaron 80 épocas (o *epochs*, en inglés) de entrenamiento y se utilizó el optimizador Adam. Como función de pérdida se utilizó la función de entropía cruzada.

3.2.1. Detalles de implementación y mejoras de la U-Net

Inicialmente, se implementó una U-Net desde cero con algunos cambios respecto de su implementación tradicional. Se utilizó *batch normalization* para acelerar la convergencia de la red y mejorar la estabilidad del entrenamiento. Además, se normalizaron los datos calculando los valores propios del conjunto de entrenamiento utilizado.

Luego, en lugar de utilizar una *transposed convolution* para hacer la *up convolution*, se realizó un *upsampling* a través de una interpolación bicúbica para aumentar el tamaño de la imagen seguido de una convolución de 1x1 para dividir por la mitad la cantidad de canales. El método de *upsampling* escogido permite obtener resultados más suaves [23].

Se entrenó la red con el conjunto de entrenamiento, y se utilizó el conjunto de validación para elegir los pesos del modelo que mejor evaluaban en el *F1-score*. La red tuvo un total de 28,955,471 parámetros.

Además de esta implementación de la U-Net, se probaron otras modificaciones para mejorar los resultados obtenidos. En primer lugar, se experimentó agregar *dropouts* con distintas probabilidades y en distintas capas de la red: entre la última del *encoder* y la primera del *decoder* y entre las convoluciones dobles. No se incluyen resultados de este ensayo en el siguiente capítulo pues no mejoraron la *performance* de la red en el F1. Una posible explicación de esto es que la red no está sobreajustando, con lo cual apagar neuronas no mejoraría el rendimiento de la misma. Puede consultarse la Fig. 4.1 y su análisis para más detalles.

Luego, se utilizaron distintos tipos de *schedulers*. Estas modificaciones sí mejoraron los resultados obtenidos anteriormente, e incluso superaron a los del trabajo original [9]. Luego, los modelos de esta red que se tuvieron en cuenta en el siguiente capítulo son cuatro: sin *schedulers* adicionales, con un *scheduler* `OneCycleLR()`, con un *scheduler* `ReduceLROnPlateau()` y con un *scheduler* `CosineAnnealingLR()`.

Para los distintos *schedulers* utilizados, se probaron varios parámetros. Las siguientes fueron las mejores combinaciones que se encontraron para cada uno de ellos. Para `OneCycleLR()`, los parámetros fueron `{max_lr = 1e-3, steps_per_epoch = len(train_dataloader), anneal_strategy = 'cos'}`. Para `ReduceLROnPlateau()`, se utilizó `{mode = 'max', factor = 0.5, patience = 3, threshold = 1e-4, min_lr = 1e-6}`, maximizando el F1 en base a lo calculado en el conjunto (o *set*) de validación. Por último, para `CosineAnnealingLR()`, los parámetros fueron `{T_max = 10, eta_min = 1e-6}`.

Todas las variantes mencionadas tuvieron un tiempo de entrenamiento similar de aproximadamente 8 horas para las 80 épocas consideradas.

3.2.2. Detalles de implementación de la DeepLabV3

Con el objetivo de encontrar una mejora en las métricas evaluadas, se optó por utilizar la DeepLabV3 que se encuentra entre las redes convolucionales más recientes y utilizadas en el estado del arte para resolver este tipo de problemas de segmentación semántica. Esta red cuenta con una ResNet-50 de *backbone* y posee 39,637,327 parámetros.

En esta ocasión se utilizó la versión de la red ya implementada en la librería *torchvision* con su *backbone* preentrenado en ImageNet. Cabe remarcar que la red no fue preentrenada en su totalidad, y que el preentrenamiento del *backbone* en ImageNet sirve para acelerar la convergencia del modelo durante el entrenamiento con el *dataset* objetivo. Popularmente, a este tipo de entrenamiento realizado sobre una red preentrenada se lo conoce como *fine-tuning*.

Además, se utilizaron los valores de normalización de ImageNet durante la etapa de normalización de imágenes. Esto permite que no se rompa la distribución de los valores de entrada que el modelo esperaba, aprovechándose mejor su preentrenamiento.

Tal como en el caso de la U-Net, se introdujeron variantes y modificaciones para mejorar el resultado en F1 obtenido. Luego, los modelos que se presentan en el próximo capítulo para esta red son los mismos que para la U-Net: una configuración sin *schedulers*, otra con `OneCycleLR()`, otra con `ReduceLROnPlateau` y una última con `CosineAnnealingLR()`; donde todos los *schedulers* tomaron los mismos parámetros que antes.

Todas las variantes mencionadas tuvieron un tiempo de entrenamiento similar de aproximadamente 6 horas para las 80 épocas consideradas. Este tiempo fue menor al observado en la U-Net, pero debe tenerse en cuenta el *backbone* de la red había sido preentrenado.

4. RESULTADOS

En este capítulo, se presentan los resultados obtenidos y los gráficos y cuadros comparativos para las redes utilizadas en las métricas que resultaron de mayor interés.

En la sección 4.1.1 se replican los resultados del trabajo original, y en las siguientes se introducen modificaciones con el objetivo de mejorar el desempeño sobre el conjunto de datos descripto previamente.

4.1. Resultados obtenidos con la U-Net

4.1.1. Sin *schedulers*

Partiendo desde el modelo inicial utilizado de la U-Net (esto es, sin *dropouts* ni *schedulers*) se graficaron las métricas de *accuracy* y *F1-score*, y la función de pérdida a lo largo de las *epochs* (Fig. 4.1).

En la primera y tercera gráfica puede verse que *F1-score* y *accuracy* aumenta tanto en entrenamiento como en validación de manera sostenida hasta aproximadamente la época 50. Luego, la curva de validación se estabiliza más rápidamente y la curva de entrenamiento crece ligeramente un poco más. Véase que ambas curvas están bastante cerca y no hay una gran diferencia de la *performance* en el entrenamiento respecto de la validación.

En la segunda, puede verse que también las dos curvas de pérdida son muy cercanas y la pendiente es negativa hasta aproximadamente la época 50.

El hecho de que la pérdida baje considerablemente y que la curva de las métricas sea cercana para ambos conjuntos, sugieren que el modelo no está sobreajustando.

Para quedarse con una configuración de pesos específica para evaluar en el conjunto de evaluación, escogimos la de la época donde la *performance* del conjunto de validación en F1 era la más alta. Se obtuvo un resultado de 0.7207 en esta métrica en las imágenes de evaluación, en coincidencia con [9]. En la Tabla 4.1, se mencionan las distintas métricas calculadas. Para obtener un *score* por métrica que englobara todas las clases calculadas, se utilizó el promedio, sin ponderar por clase (comúnmente conocido como *macro average*).

Resulta interesante considerar, además de las métricas anteriores, cómo fueron los errores obtenidos. Es decir, si clasificó un píxel de la clase i incorrectamente, ¿con qué otra clase lo confundió? ¿Cuán seguido clasificó un píxel i en la clase j? ¿Cuáles son las clases que menos distingue entre sí la red? Para responder todos estos interrogantes, se realizó una matriz de confusión, donde en la fila *i-ésima* y columna *j-ésima* se representa con valores entre 0 y 1 la cantidad de veces en las que un píxel de clase i fue clasificado como de clase j por sobre el total de píxeles de la clase i. En el mejor de los casos, la diagonal que comienza en el extremo superior izquierdo y finaliza en el inferior derecho estaría repleta de 1s, y en los demás casilleros habría solo 0s. La Fig. 4.2 corresponde a la matriz de confusión obtenida para esta configuración.

Puede visualizarse en la matriz una columna de valores mayores a 0.0 en el extremo derecho de la matriz de confusión. Nótese que en el casillero *i-ésimo* de esa columna se indica la cantidad de veces que la especie i fue clasificada érroneamente como parte del fondo de la imagen.

Es creíble que esto tenga sentido: recuérdese que muchas especies fueron fusionadas

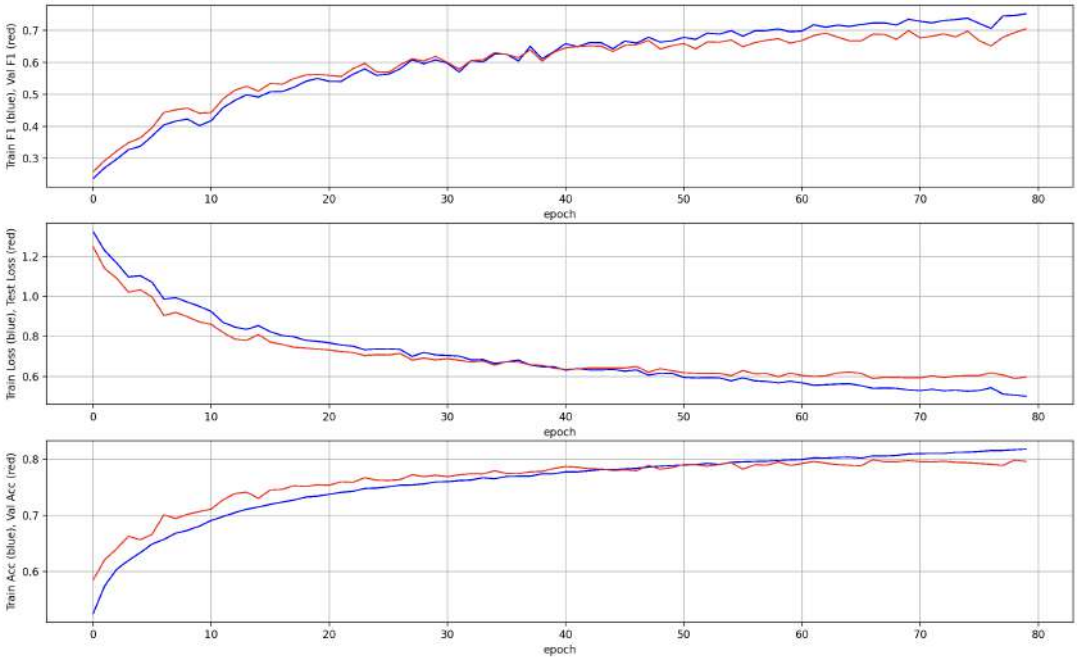


Fig. 4.1: Gráfico de las distintas funciones para el *set* de entrenamiento (azul) y para el de validación (rojo) a través de las épocas.

Clase	***Precision***	***Recall***	**F1**	**IOU**
ABBA	0.8340	0.7420	0.7853	0.6465
ACPE	0.5450	0.5213	0.5329	0.3632
ACRU	0.7340	0.6947	0.7138	0.5550
ACSA	0.7303	0.5405	0.6213	0.4506
BEAL	0.7646	0.6452	0.6998	0.5383
BEPA	0.7996	0.8762	0.8361	0.7184
FAGR	0.5587	0.8262	0.6666	0.4999
LALA	0.8140	0.8115	0.8127	0.6846
Mort	0.6929	0.5958	0.6407	0.4713
PIST	0.8981	0.8350	0.8654	0.7628
Picea	0.8150	0.6697	0.7352	0.5813
Populus	0.8220	0.9012	0.8598	0.7540
THOC	0.6192	0.8116	0.7025	0.5414
TSCA	0.5808	0.7392	0.6505	0.4820
Fondo	0.6903	0.6863	0.6883	0.5248
Promedio	0.7266	0.7264	0.7207	0.5716

Tab. 4.1: Resultados obtenidos al evaluar el conjunto de evaluación en las distintas métricas.

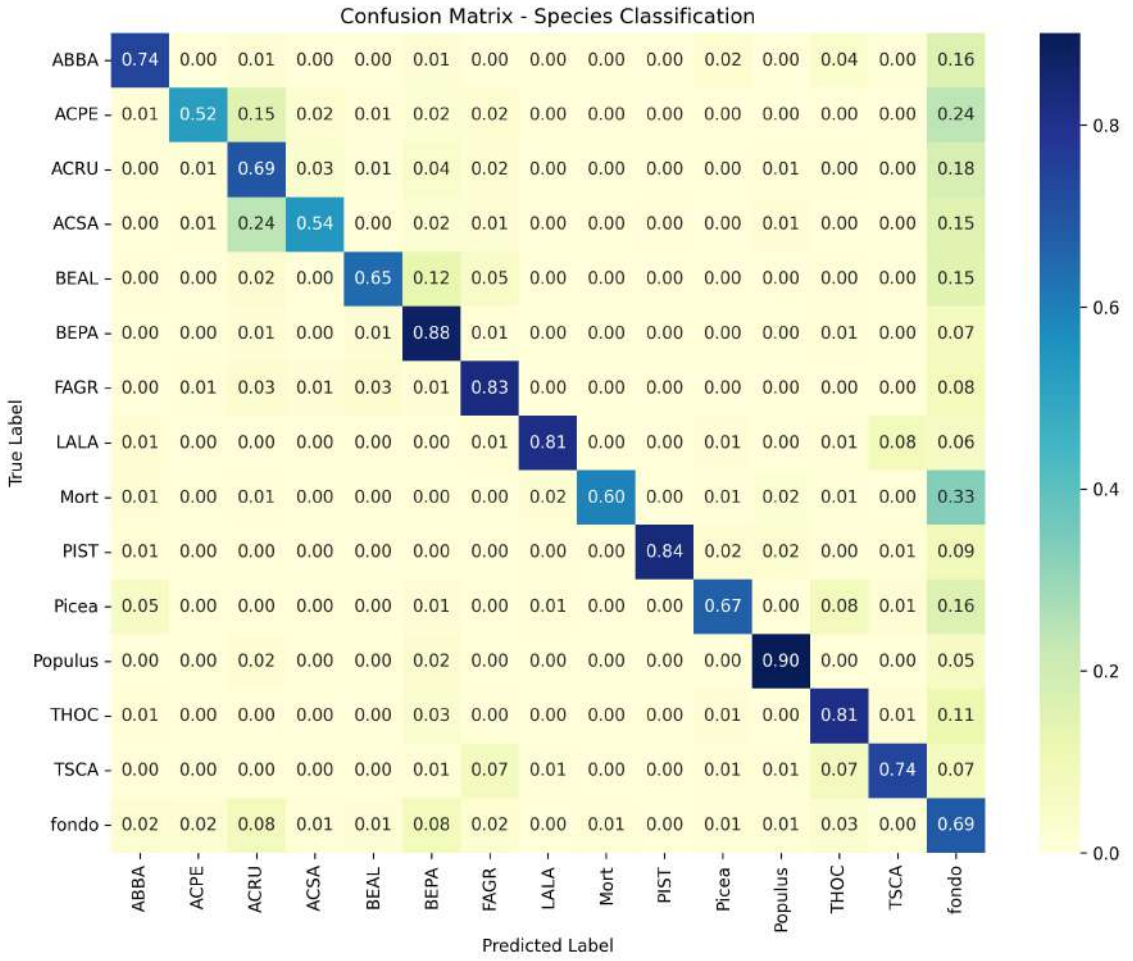


Fig. 4.2: Matriz de confusión obtenida para la primera implementación de la U-Net.

con el fondo. Luego, si son muchas las apariciones de árboles en la clase "fondo", podrían dificultar la diferenciación respecto de las demás clases. Además, hay que considerar que la clase "fondo" es de las mayoritarias en cantidad de píxeles (véanse las Tablas 3.2 y 3.3).

Por otro lado, se puede notar que la clase con valor más alto en la última columna (quitando la clase "fondo", por obviedad) es "Mort". Esta clase pertenece a los árboles muertos (del francés *mort*, que significa "muerto"). Luego, no poseen rasgos que las permitan diferenciarse ampliamente de otras especies como las hojas. Las imágenes fueron tomadas desde una vista superior y los árboles muertos poseen una forma ramificada particular muy distinta a los árboles con hojas. Estos últimos son de una forma algo más globular y de distinto color (posiblemente tonos de verde, aunque no exclusivamente). El color de la clase "Mort" también podría llegar a ser similar al del suelo en algunos casos, como por ejemplo, si el suelo está compuesto de tierra o barro, conduciendo a errores en la clasificación.

Por último, obsérvese que, fuera de la diagonal principal y de la última columna, los casilleros con valores más altos en la matriz de confusión se encuentran cercanos a la diagonal principal. Esto sucede también con otras clases (véase la Tabla 3.1). Además, en la matriz de confusión las especies que son del mismo género aparecen cercanas; y son en los casilleros donde se entrecruzan estas especies que se encuentran los valores más altos de la matriz -sin considerar nuevamente a la diagonal que cruza a una clase con ella misma. Por ejemplo: las clases de la red que comienzan en "AC" son especies de árboles que pertenecen al género *Acer* (en español, arce). Luego de mencionar estas consideraciones, se observa que la matriz de confusión guarda similitud con la obtenida en [9].

Adicionalmente, se realizó una comparación visual entre la máscara original proveniente del *dataset* y la máscara obtenida para cada imagen a partir de la red. En la Fig. 4.3 se grafican 5 imágenes del conjunto de evaluación. Nótese que en la segunda columna se presenta para cada imagen de entrada el *ground truth*, que corresponde al resultado que se espera que la red retorne al tomar como entrada la imagen de la primera columna, y es la máscara obtenida a partir del preprocesado del conjunto de datos en el Capítulo 3. En la primera, segunda y tercera fila de la Fig. 4.3, se ve que la salida de la red tiene algo de ruido, pues clasificó pequeñas cantidades de píxeles cercanos como de una especie distinta a la que pertenecían. En la cuarta fila en cambio, identificó otras áreas grandes como especies distintas a las del *ground truth*, pues detectó parte de la clase "fondo" como una especie distinta. De todas formas, para las primeras cuatro imágenes, la clasificación obtenida con la red es similar a la clasificación real de la imagen. Por último, en la quinta fila la salida de la red dista mucho del *ground truth*. La clase de la esquina superior izquierda no es reconocida y las clases de la mitad inferior de la imagen que no son fondo aparecen mezcladas entre sí sin ser claramente distinguidas.

4.1.2. *Schedulers*

Como previamente se mencionó en el capítulo anterior, se hicieron ensayos de la U-Net usando distintos tipos de *schedulers*.

4.1.2.1. `OneCycleLR()`

Tal como en la configuración de la red anterior, se graficaron algunas métricas a lo largo de las *epochs* (véase Fig. 4.4). En el primer gráfico de la figura, correspondiente al

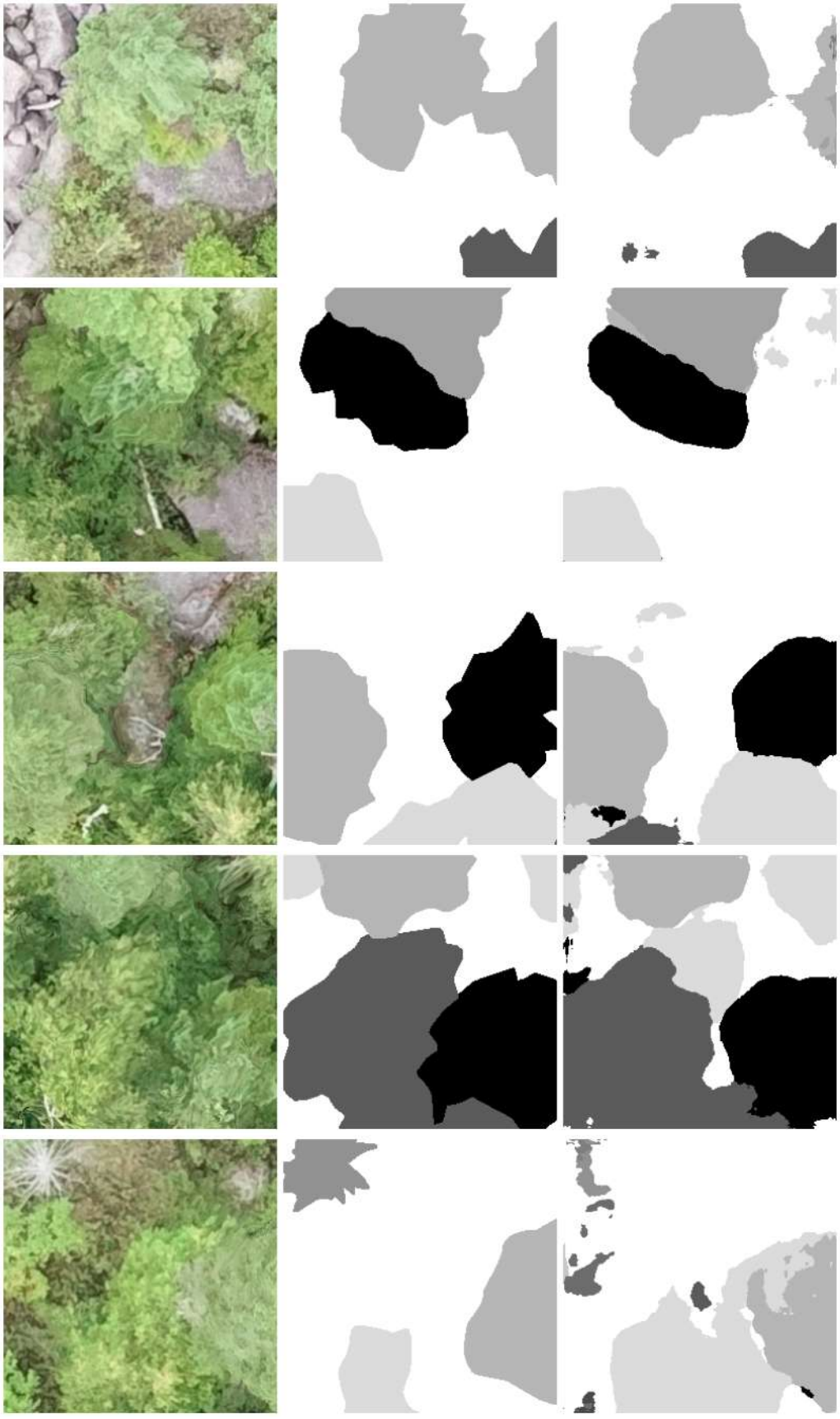


Fig. 4.3: Resultados obtenidos con la red U-Net sin *schedulers*. De izquierda a derecha en cada fila se presentan tres imágenes: ortomosaico, *ground truth* y máscara.

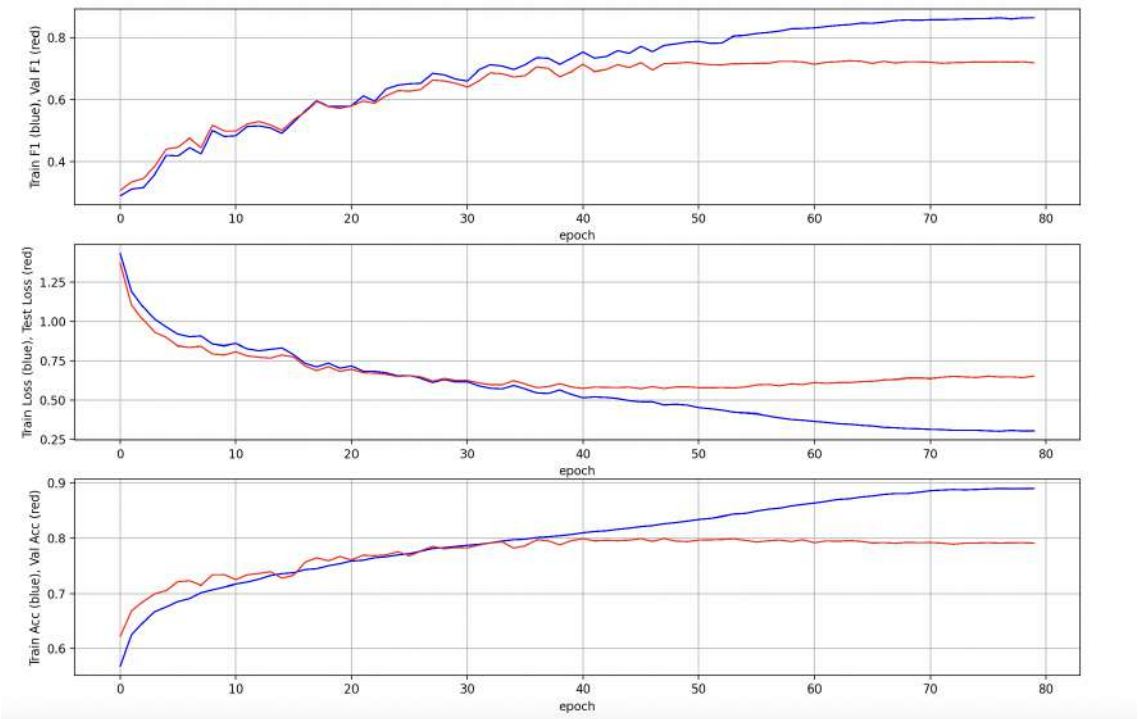


Fig. 4.4: Gráfico de las distintas métricas para el conjunto de entrenamiento (azul) y para el de validación (rojo) a través de las épocas, usando una U-Net con un *scheduler* `OneCycleLR()`.

F1 score, se observa que la pendiente de ambas curvas es creciente hasta aproximadamente la *epoch* 50, y que ambas curvas están relativamente cerca. Luego, la curva de validación se estanca mientras que la de entrenamiento sigue creciendo. Lo mismo sucede pero de manera decreciente en el gráfico de la función de pérdida. El único gráfico donde la curva de entrenamiento y validación se distinguen más entre sí es el último, donde la curva de entrenamiento es casi una función lineal mientras que la de validación parece menos estable pues presenta pequeñas oscilaciones y fluctuaciones que la otra curva no.

Los anteriores resultados indican que, de utilizarse los pesos del modelo en las últimas *epochs*, habría una gran probabilidad de estar utilizando un modelo sobreajustado, pues las métricas en el conjunto de entrenamiento evaluaban mucho mejor que en el de validación. Afortunadamente, y como fue mencionado anteriormente, los pesos utilizados fueron los de la *epoch* donde mejor *performance* obtuvo el conjunto de validación en el F1.

Para este *scheduler*, los resultados obtenidos se visualizan en la Fig. 4.5, y en la Tabla 4.2 se encuentra la *performance* de la red según las distintas métricas.

A comparación entre la Tabla 4.1 y la Tabla 4.2, *precision* aumentó en 0.0348, F1 aumentó en 0.0130, IOU aumentó en 0.0185 y *recall* disminuyó en 0.0070.

En cuanto a las visualizaciones de la Fig. 4.5 respecto de la Fig. 4.3, en las primeras dos imágenes la clasificación de la red es levemente peor, pues para esta configuración hay más ruido. Por ejemplo, en la esquina superior derecha una porción pequeña de la máscara gris clara ahora la detecta con otro tono de gris más oscuro. Además, el tamaño de la máscara gris claro reconocida es más pequeño que en la red anterior, que guardaba un tamaño más fiel respecto del de la *ground truth*. Sin embargo, en la tercera imagen puede verse que, si

Clase	*Precision*	*Recall*	F1	IOU
ABBA	0.8056	0.7989	0.8023	0.6698
ACPE	0.5376	0.4577	0.4944	0.3284
ACRU	0.6913	0.7650	0.7263	0.5702
ACSA	0.7373	0.5152	0.6066	0.4353
BEAL	0.8070	0.6297	0.7074	0.5473
BEPA	0.8069	0.8758	0.8399	0.7240
FAGR	0.7414	0.7172	0.7291	0.5737
LALA	0.7541	0.8478	0.7982	0.6641
Mort	0.8050	0.4279	0.5588	0.3877
PIST	0.8423	0.8914	0.8661	0.7637
Picea	0.7876	0.7640	0.7756	0.6335
Populus	0.8623	0.8940	0.8779	0.7823
THOC	0.7388	0.7563	0.7474	0.5967
TSCA	0.8121	0.7707	0.7908	0.6540
Fondo	0.6910	0.6797	0.6853	0.5213
Promedio	0.7614	0.7194	0.7337	0.5901

Tab. 4.2: Resultados obtenidos al evaluar el conjunto de evaluación en las distintas métricas, utilizando una U-Net con un *scheduler* `OneCycleLR()`.

bien aumentó el ruido en la esquina superior derecha, ya no se reconoce una especie gris claro entre la especie negra y la gris oscuro. Por otra parte, en la cuarta imagen puede notarse que la red ya no detecta como clase distinta del fondo a la región que en la Fig. 4.3 sí lo hacía incorrectamente. En la última imagen puede verse que, si bien sigue sin detectar correctamente la clase de la esquina superior izquierda, logra separar algo más a las clases de la mitad inferior de la imagen que no son fondo.

La matriz de confusión de este modelo se encuentra en la Fig. 4.6. A grandes rasgos, guarda similitud con la obtenida en el modelo sin *schedulers*. La clase peor clasificada fue "Mort", aunque ahora la confusión entre esta clase y el fondo fue mayor. Además, la red confundió frecuentemente también entre especies del mismo género.

4.1.2.2. `ReduceLROnPlateau()`

Las curvas de entrenamiento y validación según las *epochs* para esta configuración se presentan en la Fig. 4.7. En este gráfico, respecto de la Fig. 4.4, la diferencia entre las dos curvas para las tres métricas en las últimas *epochs* es mucho menor. Además, la pendiente de la curva de validación en el F1 se mantiene creciente hasta una mayor cantidad de *epochs*.

En la Tabla 4.3 se representan los *scores* del modelo en cada una de las métricas calculadas. Respecto del *scheduler* anterior (Fig. 4.2), *precision* mejoró en 0.0022, F1 en 0.0055, y IOU en 0.0042. Respecto de la U-Net implementada sin *schedulers* (Fig. 4.1), *recall* empeoró en 0.0026. La última métrica se comparó con el modelo sin *schedulers* ya que ese fue el que mejor resultado obtuvo para esa métrica en particular. La configuración de red de la subsección actual superó a la que utilizaba un *OneCycleLR()* en todas las métricas calculadas.

En la Fig. 4.8 se visualizan la clasificación obtenida por la red para las mismas 5 imágenes utilizadas previamente en comparación con la clasificación correcta para esas imágenes.

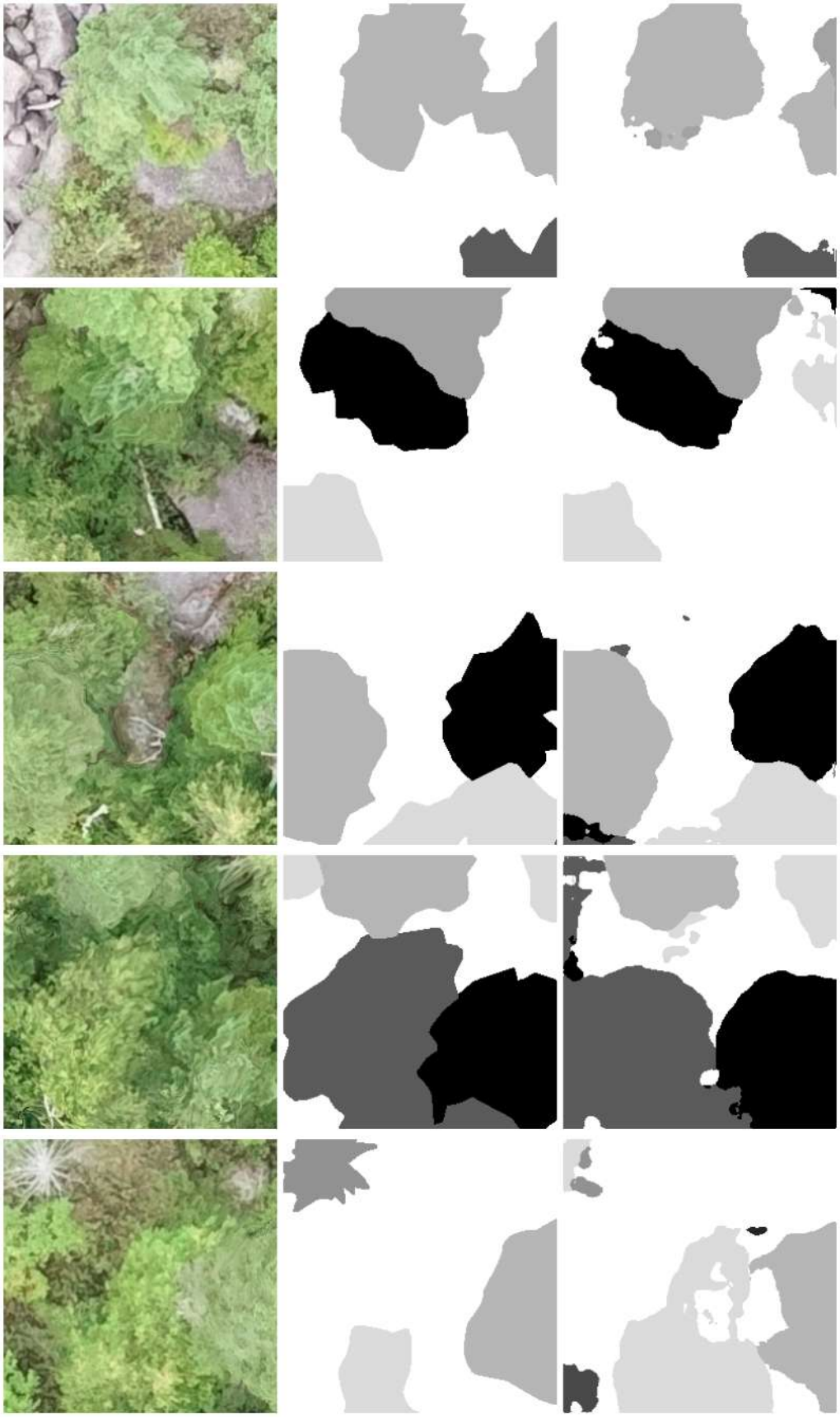


Fig. 4.5: Visualizaciones utilizando una U-Net con el *scheduler* `OneCyleLR()`. De izquierda a derecha: imagen de entrada de la red, *ground truth* y salida de la red.

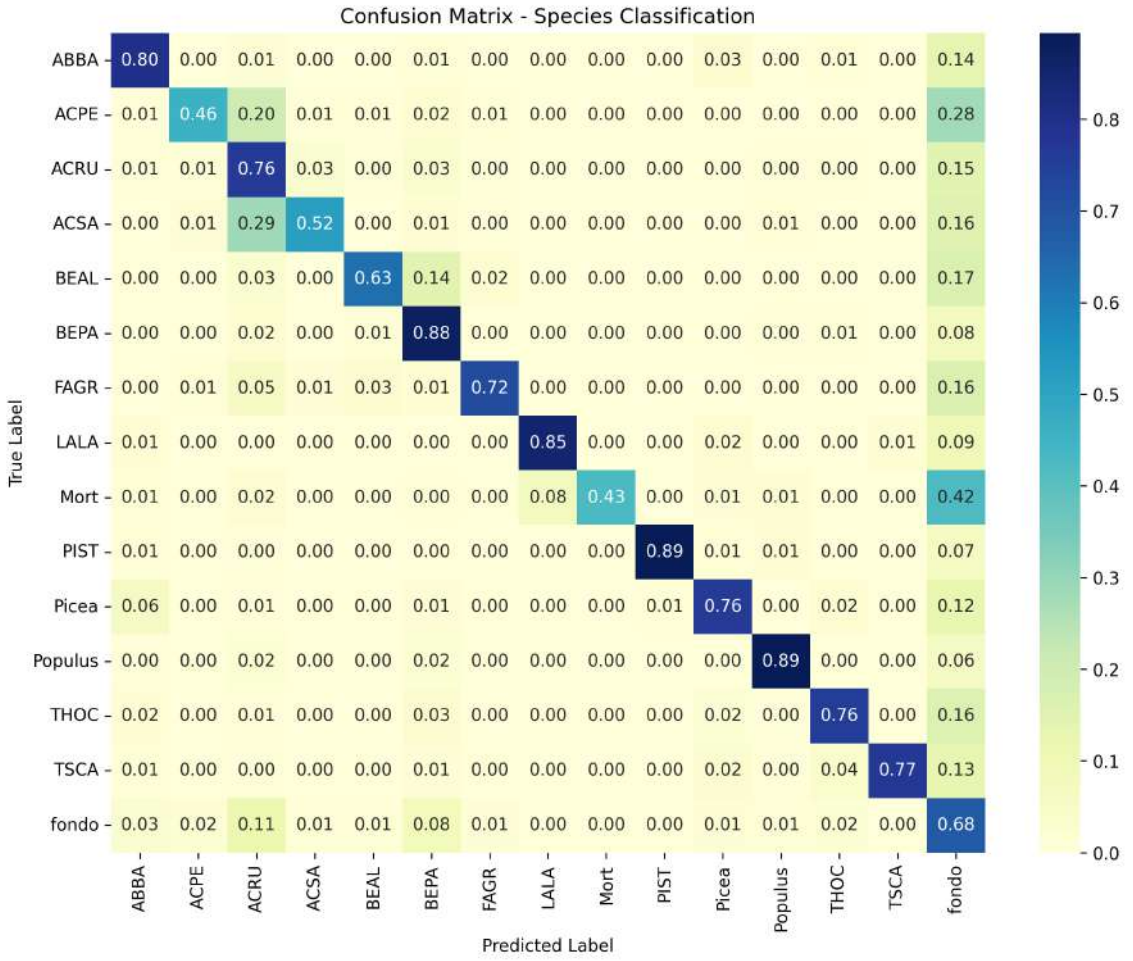


Fig. 4.6: Matriz de confusión obtenida para una U-Net utilizando un *scheduler* `OneCycleLR()`.

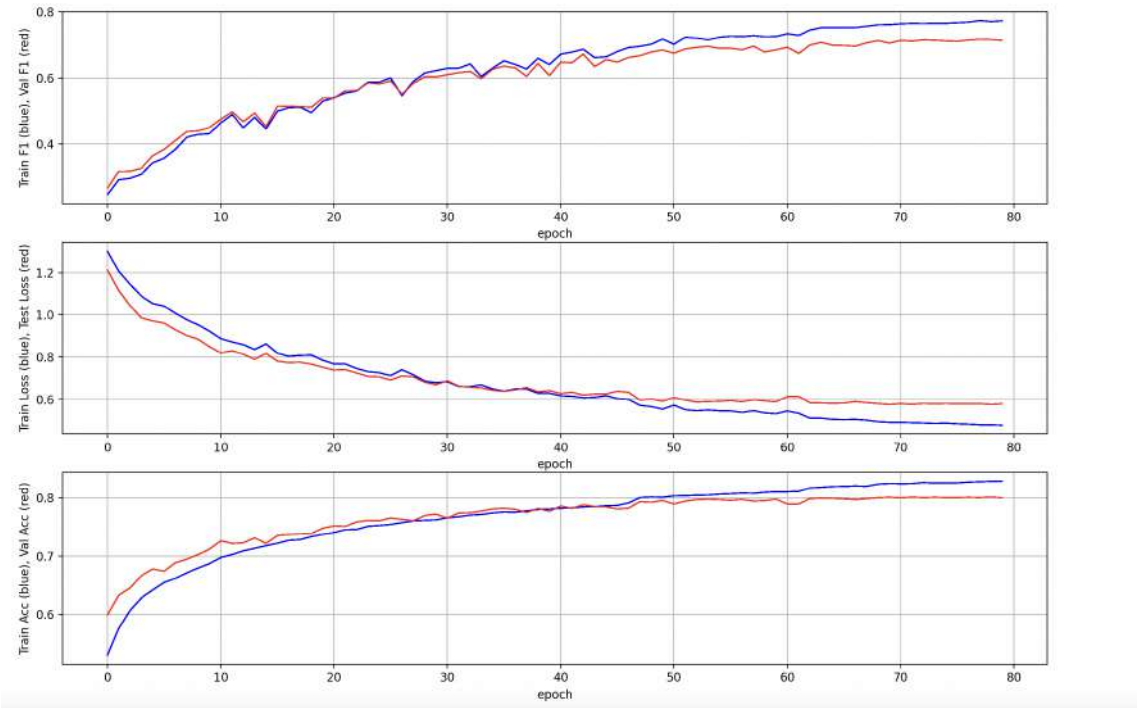


Fig. 4.7: Gráfico de las distintas métricas para el conjunto de entrenamiento (azul) y para el de validación (rojo) a través de las épocas, usando una U-Net con un *scheduler* `ReduceLROnPlateau()`.

Clase	***Precision***	***Recall***	**F1**	**IOU**
ABBA	0.8480	0.7418	0.7914	0.6548
ACPE	0.5542	0.5499	0.5520	0.3813
ACRU	0.7219	0.7457	0.7339	0.5793
ACSA	0.7548	0.5334	0.6251	0.4546
BEAL	0.7806	0.6616	0.7162	0.5579
BEPA	0.8236	0.8629	0.8428	0.7282
FAGR	0.7095	0.7113	0.7104	0.5509
LALA	0.8060	0.8477	0.8264	0.7041
Mort	0.7936	0.5323	0.6372	0.4676
PIST	0.8755	0.8689	0.8722	0.7734
Picea	0.8033	0.7349	0.7676	0.6228
Populus	0.8426	0.9028	0.8717	0.7725
THOC	0.6923	0.8035	0.7438	0.5921
TSCA	0.7605	0.6487	0.7002	0.5387
Fondo	0.6835	0.7122	0.6975	0.5356
Promedio	0.7633	0.7238	0.7392	0.5943

Tab. 4.3: Resultados obtenidos al evaluar el conjunto de evaluación en las distintas métricas, utilizando una U-Net con un *scheduler* `ReduceLROnPlateau()`.

Los resultados de esta configuración se parecen a los de la Fig. 4.3. Las principales diferencia radican en la tercer fila de imágenes, donde la configuración actual clasificó mejor los píxeles de la esquina inferior izquierda; y en la quinta fila de imágenes, donde a diferencia de las configuraciones de red anteriores, las formas de las regiones para las distintas clases no se encuentran mezcladas entre sí. Si bien en esta última imagen las clases que no son fondo ocupan más área del que deberían, no están mezcladas de manera que no exista una clara línea divisoria entre los píxeles de una y otra clase. Además, para esa última imagen, la especie de la esquina superior izquierda se clasifica mucho mejor que en las imágenes anteriores, ya que el color de la máscara es el mismo y el área que ocupa respecto del área que debería ocupar siguiendo el *ground truth* es similar.

La matriz de confusión de esta red (Fig. 4.9) es muy similar a las anteriores, con mayor cantidad de aciertos en algunas clases. Aún siendo la configuración de U-Net que mejor evaluó en las métricas, tuvo menos aciertos sobre la clase *Mort* que la red que no tuvo *schedulers*.

4.1.2.3. `CosineAnnealingLR()`

Por último, se graficaron las curvas de entrenamiento y validación para el *scheduler* restante en la Fig. 4.10. Las curvas de validación y entrenamiento son de una forma peculiar en este modelo: nótese que presentan periódicamente una especie de "montañas sin picos" que ocupan la misma cantidad de *epochs*. Esto se debe probablemente al *scheduler*, que sigue la función coseno.

En el primer y tercer gráfico se ve que la pendiente de ambas curvas es creciente, y que las curvas de validación son muy cercanas a las de entrenamiento. Esto podría estar indicando que aún no convergió el modelo y que todavía puede mejorar. Lo mismo sucedió, pero con pendiente negativa, en el segundo gráfico, correspondiente a la función de pérdida.

En la Tabla 4.4 se disponen los resultados obtenidos según las distintas métricas para esta configuración de la U-Net. Respecto de los modelos anteriores, sólo fue superior que el que no utilizaba *schedulers* en la métrica de *precision*, mientras que obtuvo una peor *performance* frente a todos los demás modelos en todas las métricas. Esto puede deberse a lo mencionado anteriormente: la red podría haber seguido entrenándose y la función de pérdida podría haber seguido bajando conforme subían los *scores* en las distintas métricas, ya que en la Fig. 4.10 no se ve que se estabilice la curva de validación.

Para finalizar, en la Fig. 4.11 se muestra la clasificación que la red dio como resultado para las 5 imágenes que se visualizaron en las demás redes, en contraste con la *ground truth*. En la primera fila de imágenes, puede notarse que la red confunde totalmente una clase con otra y mezcla dos colores para marcar la región central superior. En la segunda fila de imágenes sigue habiendo ruido como lo había sin utilizar *schedulers* en la U-Net. En la tercera imagen, si bien disminuye el ruido respecto de la Fig. 4.3, la clase inferior en la imagen sigue sin reconocerse del todo bien. En la cuarta fila de imágenes se sigue reconociendo una clase distinta al fondo incorrectamente en el centro de la imagen, tal como lo hacía la U-Net sin *scheduler*. En la última fila de imágenes, la clasificación hecha por la red se ve mucho menos ruidosa y más parecida a la de la U-Net que utilizó un `ReduceLROnPlateau()`.

La matriz de confusión asociada a esta configuración de red (Fig. 4.12) también guarda parecido con las anteriores.

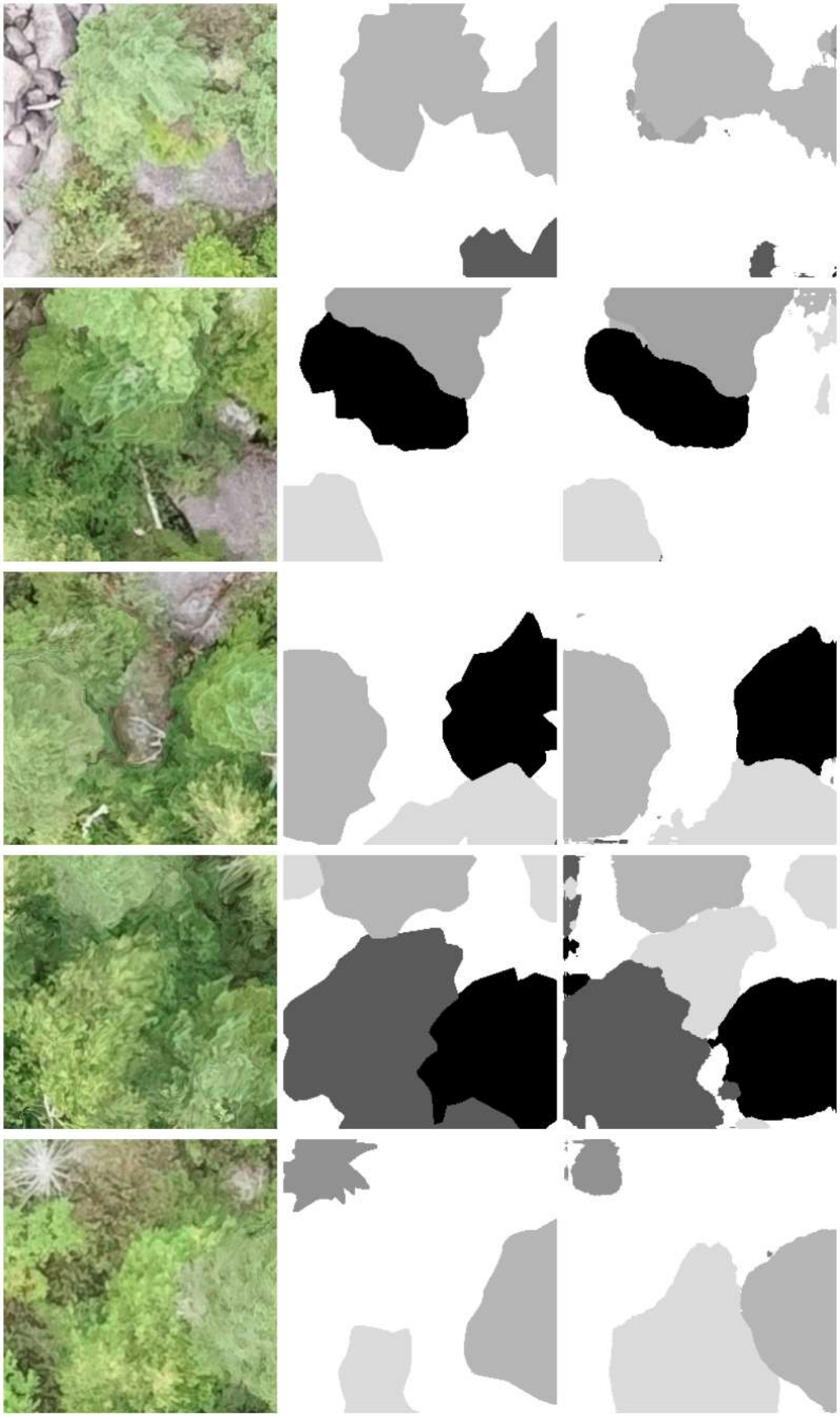


Fig. 4.8: Resultados esperados y obtenidos utilizando una U-Net con el *scheduler* `ReduceLROnPlateau()`.

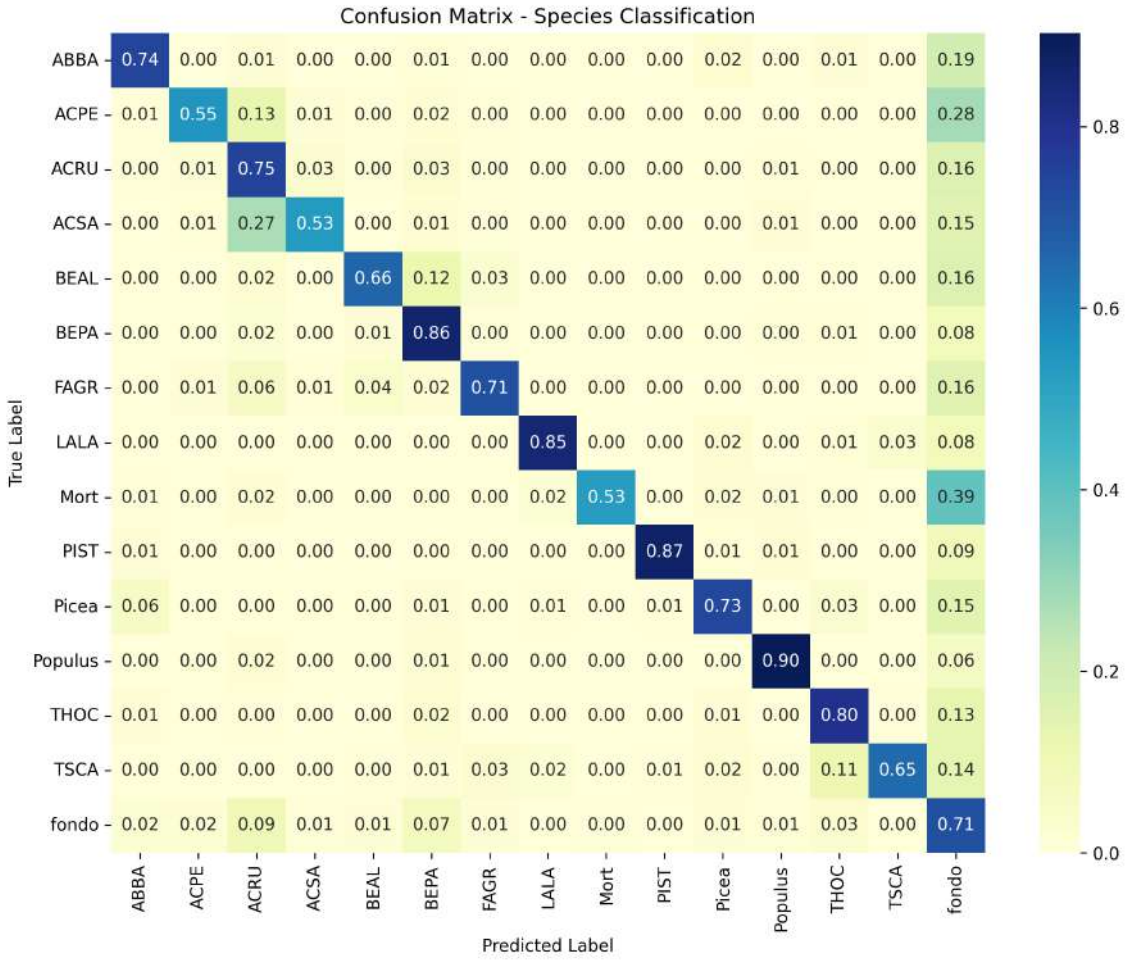


Fig. 4.9: Matriz de confusión obtenida para una U-Net utilizando un *scheduler* `ReduceLROnPlateau()`.

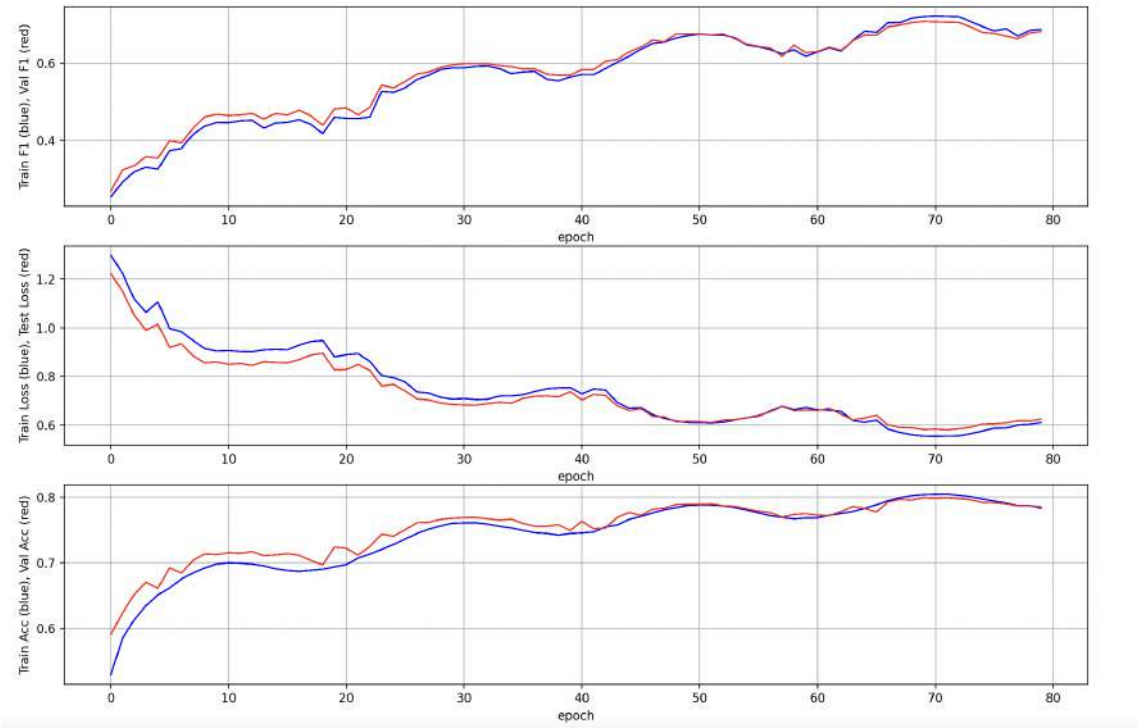


Fig. 4.10: Gráfico de las distintas métricas para el conjunto de entrenamiento (azul) y para el de validación (rojo) a través de las épocas, usando una U-Net con un *scheduler* `CosAnnealingLR()`.

Clase	***Precision***	***Recall***	**F1**	**IOU**
ABBA	0.8219	0.7541	0.7865	0.6482
ACPE	0.5415	0.5710	0.5558	0.3849
ACRU	0.7104	0.7590	0.7339	0.5797
ACSA	0.7261	0.5372	0.6176	0.4467
BEAL	0.6884	0.6682	0.6782	0.5131
BEPA	0.8098	0.8640	0.8360	0.7183
FAGR	0.7361	0.6600	0.6960	0.5338
LALA	0.7621	0.8219	0.7909	0.6541
Mort	0.8118	0.4921	0.6127	0.4417
PIST	0.8111	0.8937	0.8504	0.7397
Picea	0.8020	0.6533	0.7201	0.5626
Populus	0.8385	0.8958	0.8662	0.7640
THOC	0.6368	0.8020	0.7099	0.5503
TSCA	0.7666	0.5079	0.6110	0.4399
Fondo	0.6971	0.6863	0.6917	0.5287
Promedio	0.7440	0.7044	0.7171	0.5670

Tab. 4.4: Resultados obtenidos al evaluar el conjunto de evaluación en las distintas métricas utilizando una U-Net con un *scheduler* `CosineAnnealingLR()`.

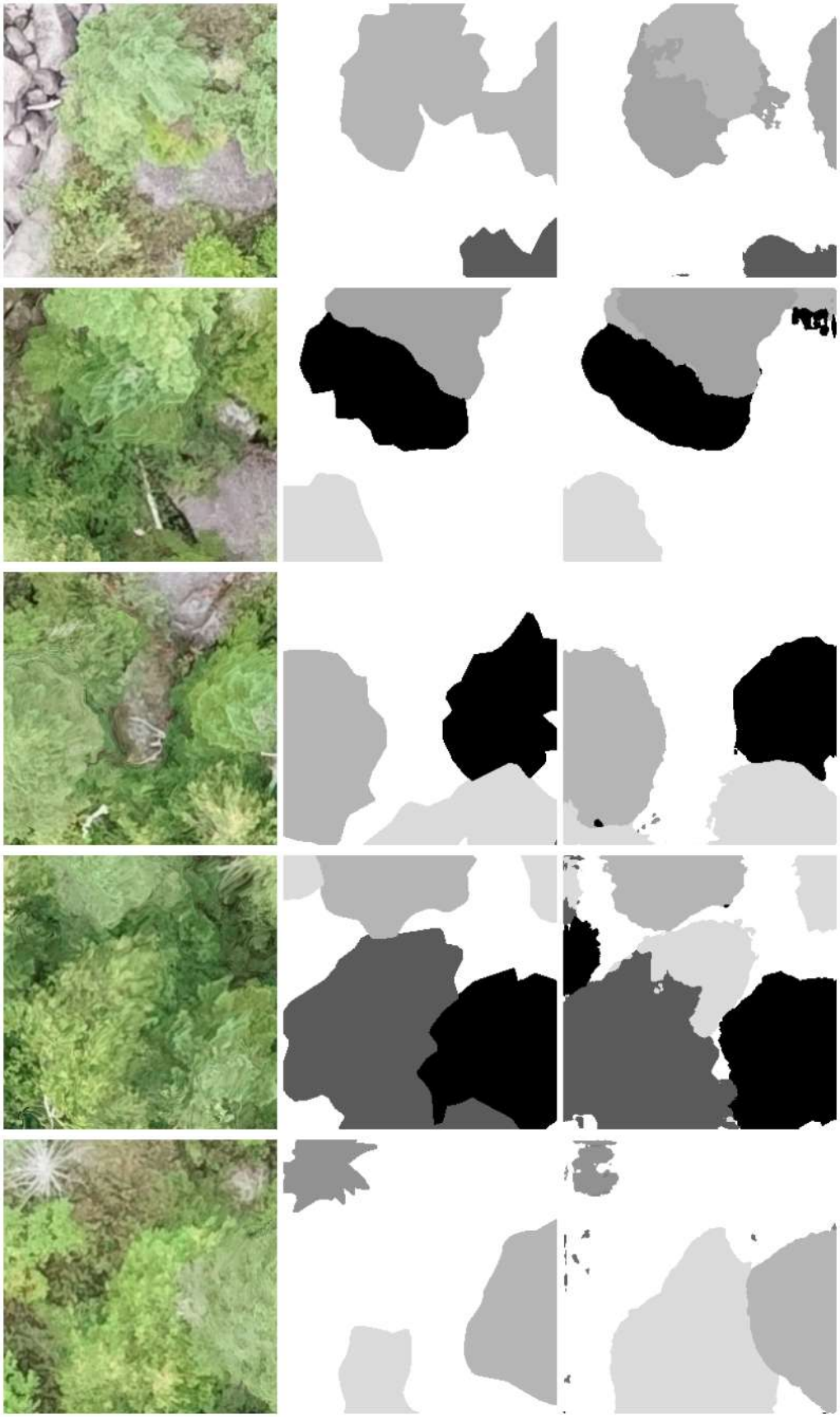


Fig. 4.11: Resultados esperados y obtenidos utilizando una U-Net con el *scheduler* `CosineAnnealingLR()`.

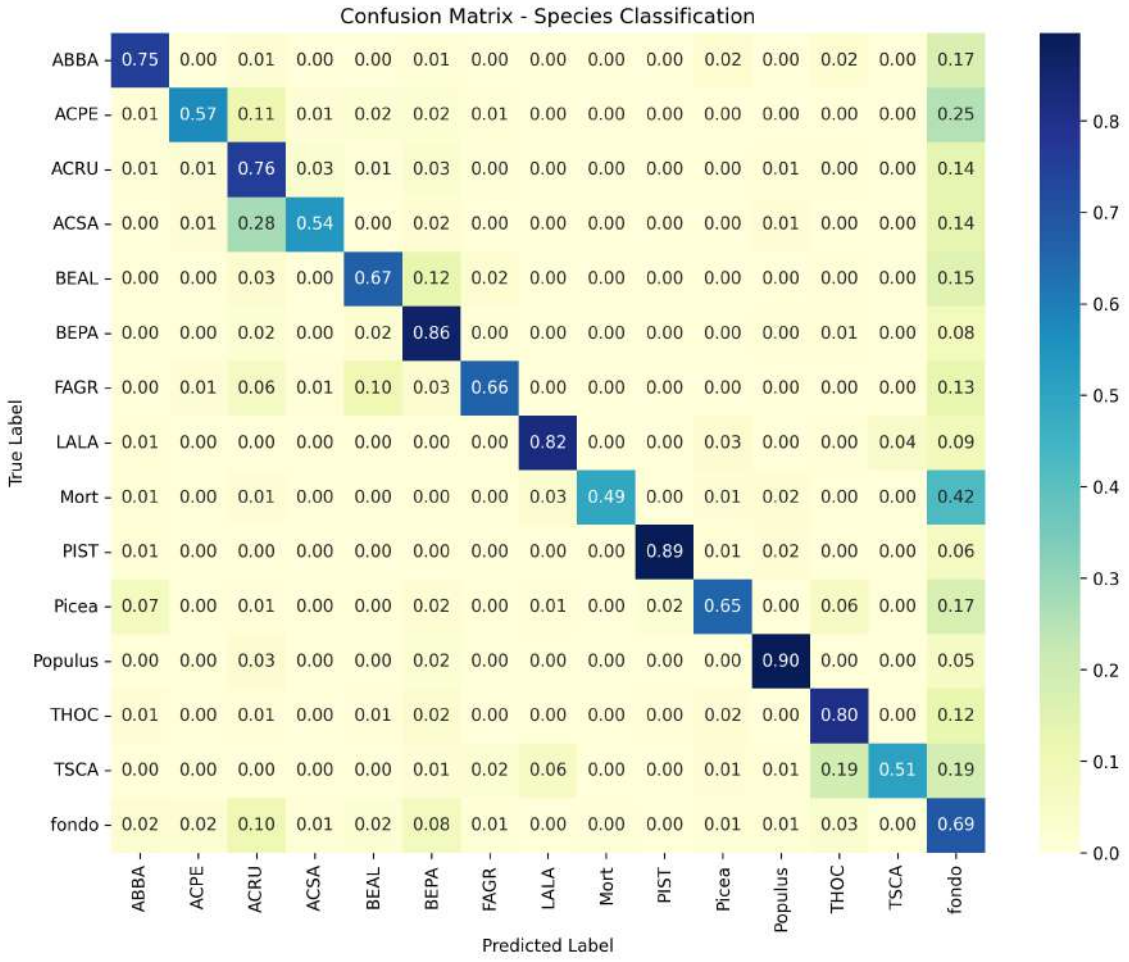


Fig. 4.12: Matriz de confusión obtenida para una U-Net utilizando un *scheduler* `CosineAnnealingLR()`.

Configuración	F1	IOU
U-Net sin *scheduler*	0.7207	0.5716
U-Net con `OneCycleLR()`	0.7337	0.5910
U-Net con `ReduceLROnPlateau()`	**0.7392**	**0.5943**
U-Net con `CosAnnealingLR()`	0.7171	0.5670

Tab. 4.5: Métricas F1 e IOU obtenidas para cada configuración de la U-Net. En negrita se resalta aquel modelo que obtuvo mejores resultados en ambas.

4.1.2.4. Resumen de resultados obtenidos con la U-Net

Luego, la configuración que mejor evaluó en las métricas utilizadas fue aquella que utilizó el *scheduler* `ReduceLROnPlateau()`. En particular obtuvo un *score* de 0.7392 en el F1, mayor al de una U-Net sin *scheduler*, que fue de 0.7207. Con esto además, se lograron mejorar los resultados el trabajo que se replicó inicialmente. La Tabla 4.5 indica, a modo de resumen, los *scores* de los anteriores modelos para algunas métricas relevantes.

4.2. Resultados obtenidos con la DeepLabV3, con ResNet50 de *backbone*

Como se mencionó en el capítulo anterior, se hicieron varios ensayos con esta red. Una sin utilizar un *scheduler*, y otras usando los tres previamente explicados.

4.2.1. Sin *scheduler*

En la Fig. 4.13 se graficaron las curvas de entrenamiento y validación para *F1-score*, *accuracy* y la función de pérdida de entropía cruzada. En contraste con los gráficos obtenidos para una U-Net tanto con como sin *schedulers*, las curvas de entrenamiento se separan de las de validación más rápidamente. A la vez, la pendiente en las primeras épocas de las curvas de los tres gráficos es muy gránde en módulo. Asimismo, desde la *epoch* 40 aproximadamente, la curva de validación en la función de pérdida tiene pendiente positiva. Estos argumentos junto al hecho de que las métricas obtenidas fueron ligeramente superiores a las de la U-Net (véase la Tabla 4.6), sostienen la hipótesis de que esta red entrenó mucho más rápidamente.

En comparación con los resultados que se obtuvieron para una U-Net sin *scheduler* (Tabla 4.1), el *F1-score* para este modelo (Tabla 4.6) fue mayor por un 0.0072. En todas las métricas consideradas, este modelo fue superior al de una U-Net sin *schedulers*.

Se graficaron también los resultados obtenidos para la red para las imágenes que habían sido utilizadas en los demás modelos (Fig. 4.14). En la primera fila de imágenes, confunde la clase gris claro con una más oscura. En la tercera fila de imágenes, se ve que ya no se genera tanto ruido en la esquina inferior izquierda como sí lo generaban los otros modelos. De todas formas, sigue habiendo ruido en las máscaras y una región del fondo coloreada incorrectamente en la cuarta fila de imágenes. La Fig. 4.15 muestra la matriz de confusión para este modelo, que también fue similar a las anteriores.

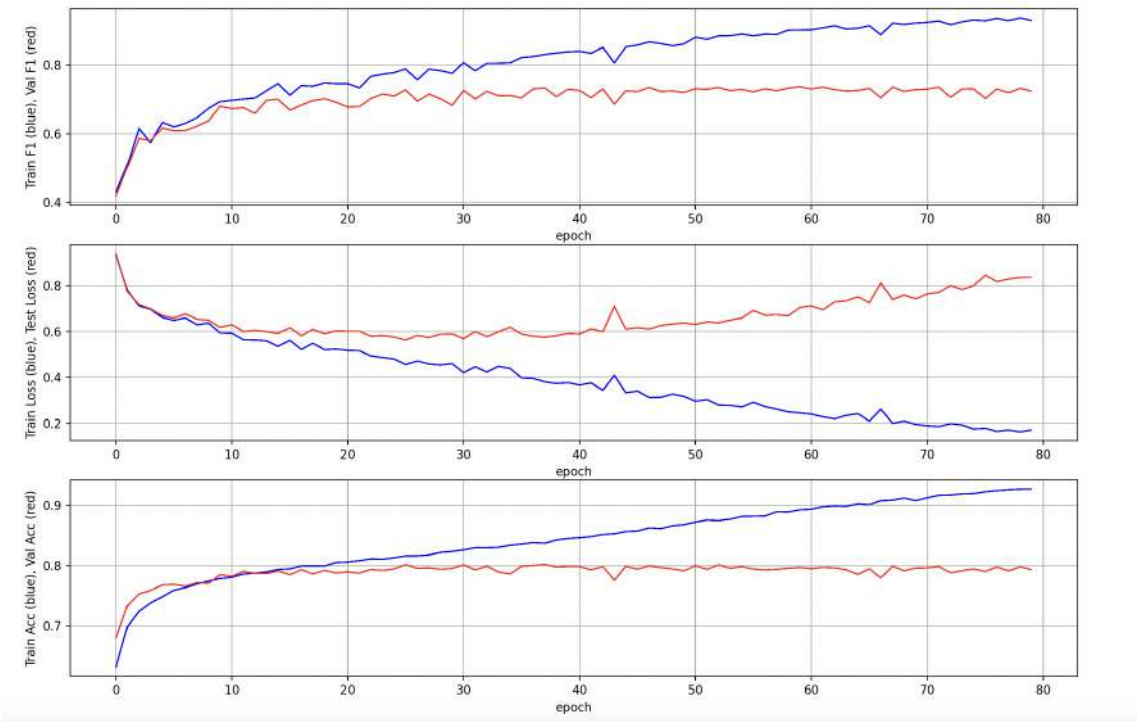


Fig. 4.13: Gráfico de las distintas métricas para el conjunto de entrenamiento (azul) y para el de validación (rojo) a través de las épocas, usando una DeepLabV3 sin preentrenar, con una ResNet50 de *backbone*.

Clase	***Precision***	***Recall***	**F1**	**IOU**
ABBA	0.8207	0.7650	0.7919	0.6555
ACPE	0.5533	0.4548	0.4992	0.3326
ACRU	0.6900	0.7591	0.7229	0.5660
ACSA	0.6361	0.6563	0.6460	0.4771
BEAL	0.7768	0.6703	0.7196	0.5621
BEPA	0.8021	0.8747	0.8368	0.7195
FAGR	0.6238	0.7517	0.6818	0.5173
LALA	0.7319	0.8755	0.7973	0.6629
Mort	0.7471	0.5342	0.6229	0.4524
PIST	0.9059	0.8613	0.8830	0.7906
Picea	0.8051	0.6847	0.7402	0.5876
Populus	0.8126	0.8987	0.8535	0.7444
THOC	0.7313	0.7485	0.7398	0.5871
TSCA	0.6354	0.7935	0.7057	0.5452
Fondo	0.7085	0.6482	0.6770	0.5118
Promedio	0.7321	0.7318	0.7279	0.5808

Tab. 4.6: Resultados obtenidos al evaluar el conjunto de evaluación en las distintas métricas para una DeepLabV3 con ResNet50 de *backbone*.

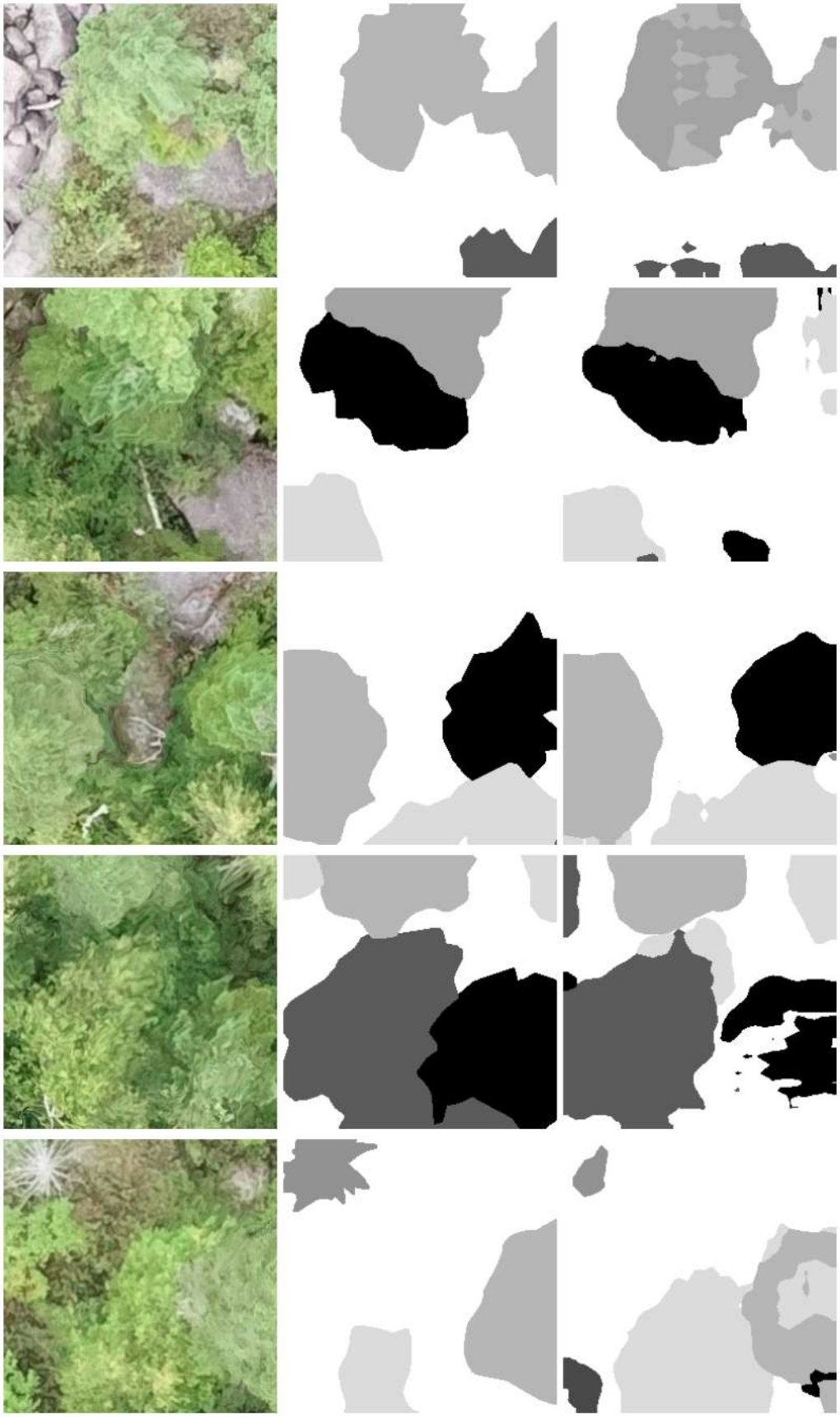


Fig. 4.14: Resultados esperados y obtenidos utilizando una DeepLabV3 con un *backbone* de ResNet50.

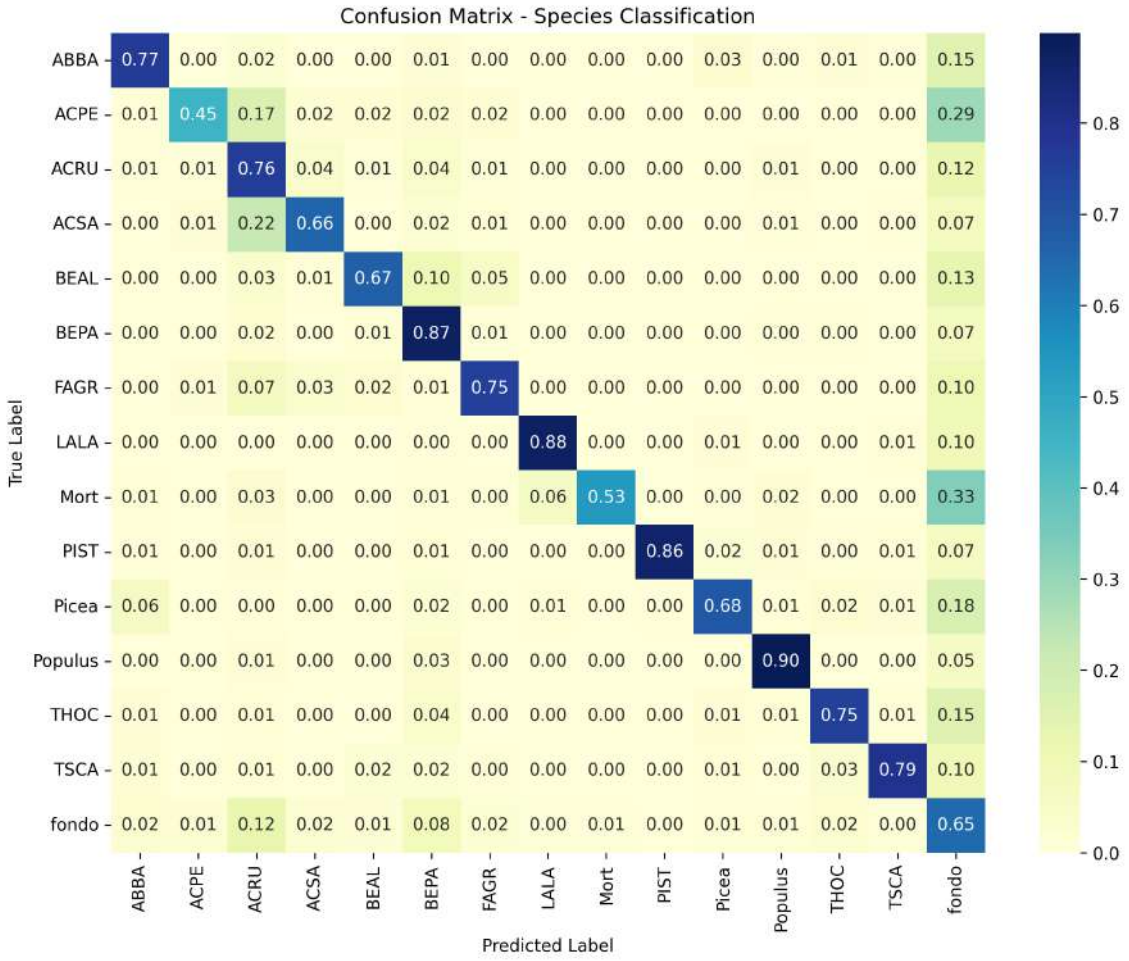


Fig. 4.15: Matriz de confusión obtenida para una DeepLabV3 con una ResNet50 de *backbone*.

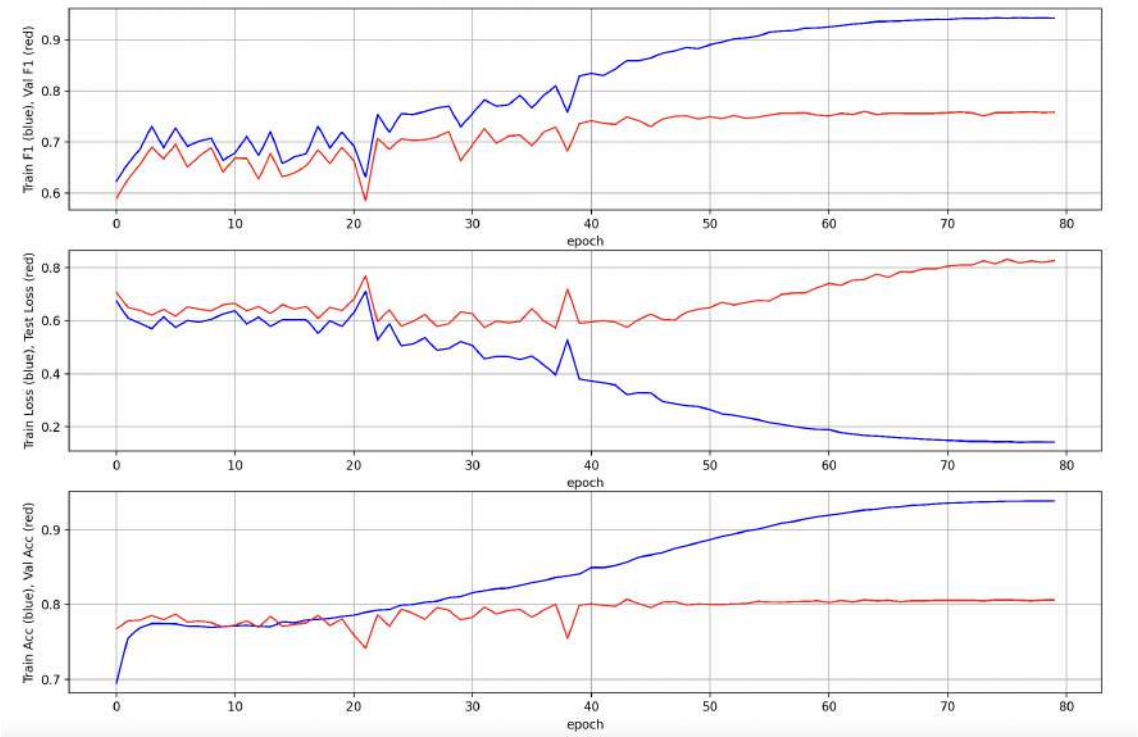


Fig. 4.16: Gráfico de las distintas métricas para el conjunto de entrenamiento (azul) y para el de validación (rojo) a través de las épocas, usando una DeepLabV3 sin preentrenar con una ResNet 50 de *backbone* y un *scheduler* `OneCycleLR()`.

4.2.2. Con *schedulers*

4.2.2.1. `OneCycleLR()`

En la Fig. 4.16 se grafican las curvas de entrenamiento y validación a lo largo de las *epochs* para distintas métricas. En la primera gráfica, se ve que el F1 en el conjunto de validación aumenta mucho inicialmente, y luego cada vez menos hasta estabilizarse alrededor de la *epoch* 40. Presenta grandes picos a lo largo de las *epochs*, tal como en el conjunto de entrenamiento. Las curvas inicialmente son bastante cercanas, y comienzan a separarse alrededor de la *epoch* 25. En el caso de la función de pérdida, la curva de validación acaba teniendo pendiente positiva, lo que indicaría, junto con el hecho de que las curvas en las últimas *epochs* están muy separadas, que hay un posible sobreajuste hacia el final del entrenamiento. De todas formas, esto no afectaría los resultados en la evaluación ya que se eligieron los pesos que mejor ajustaban a los datos de validación.

En la Tabla 4.7 se presentan los resultados obtenidos con esta configuración. En todas las métricas se superó al modelo de la DeepLabV3 anterior que no utilizaba *schedulers*. En la Fig. 4.17 se presentan las visualizaciones obtenidas según las imágenes de entrada de la red junto al *ground truth*. En contraste con Fig. 4.14, se ve la aparición de más ruido entre las distintas máscaras de las clases. La matriz de confusión se presenta en la Fig. 4.18, y guarda mucha similitud con Fig. 4.15.

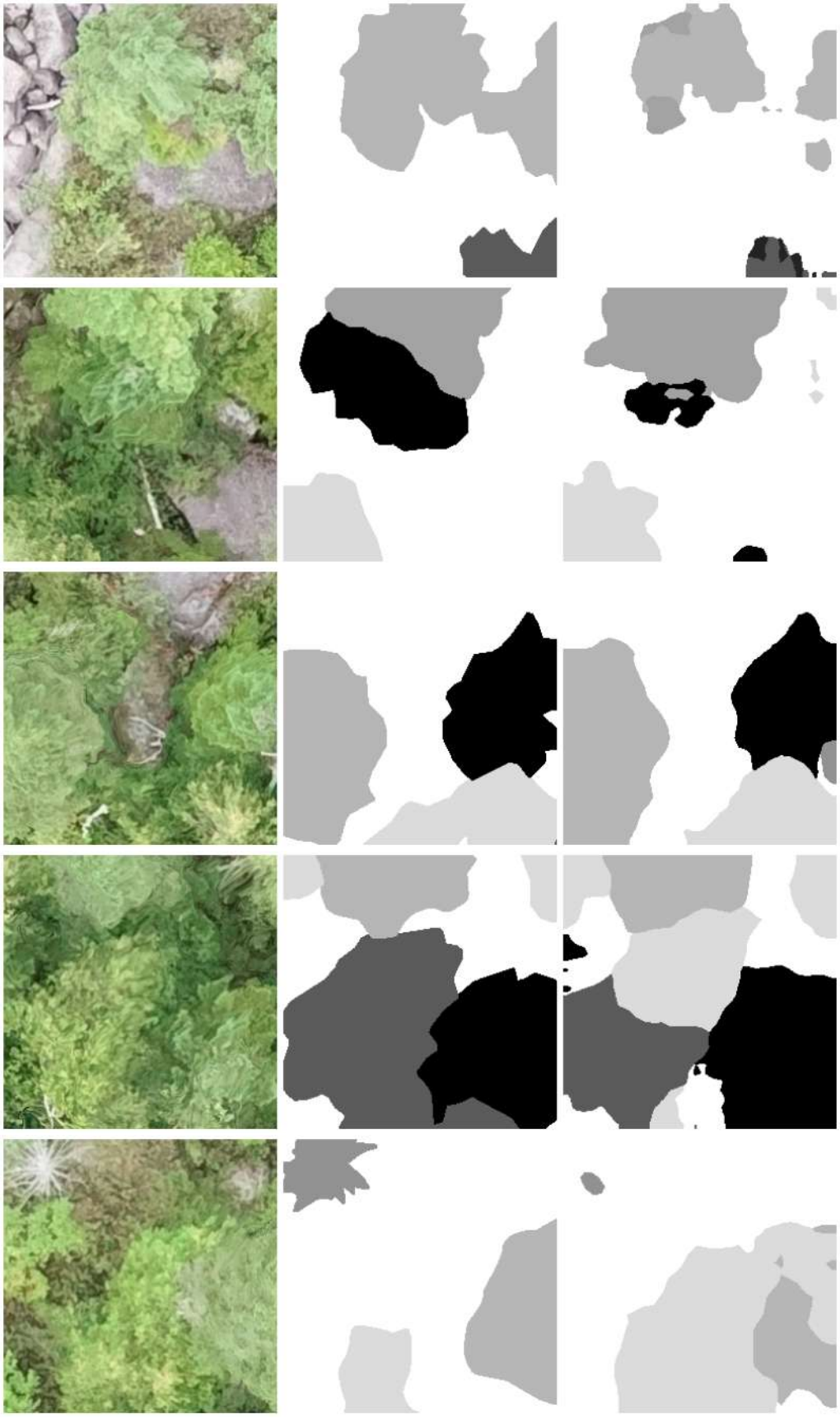


Fig. 4.17: Resultados esperados y obtenidos utilizando una DeepLabV3 con un *backbone* de ResNet50 y un *scheduler* `OneCycleLR()`.

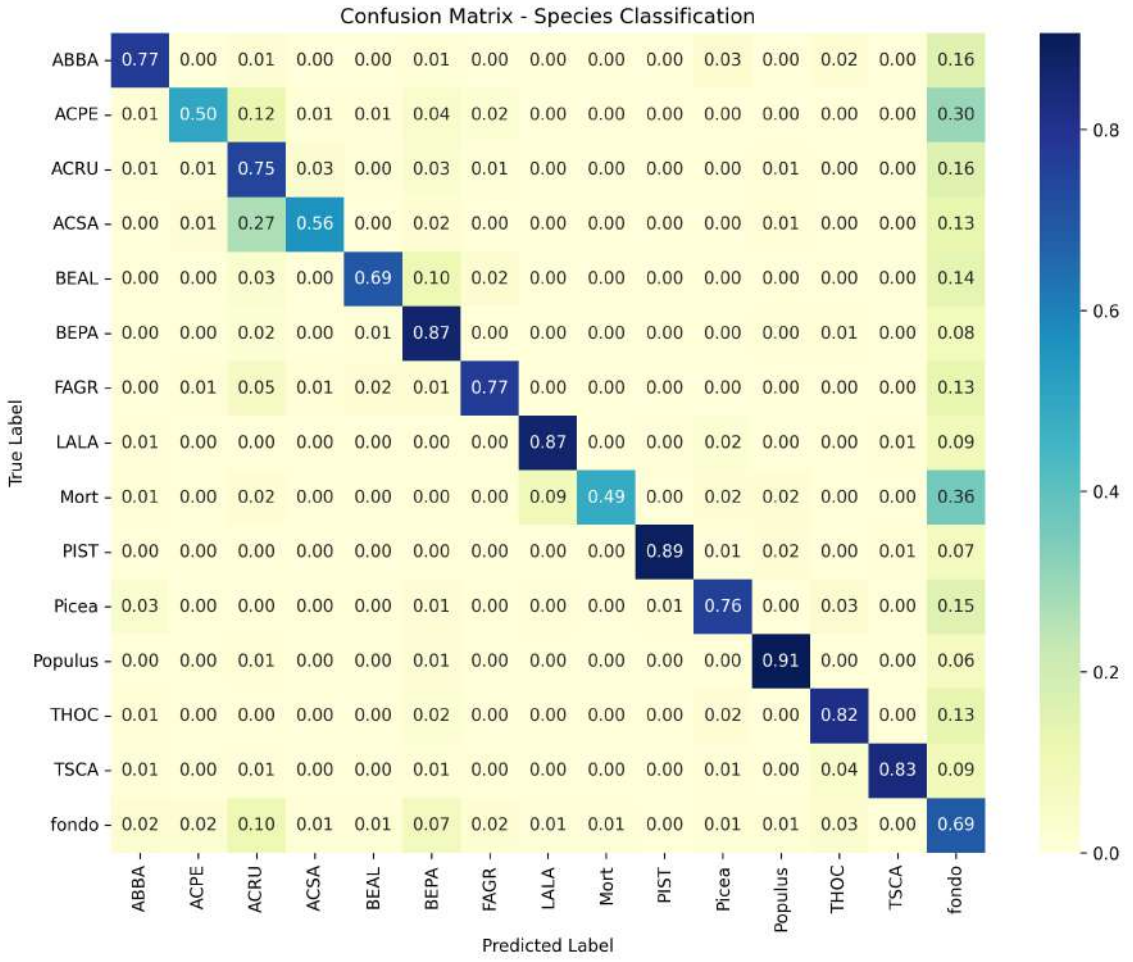


Fig. 4.18: Matriz de confusión obtenida para una DeepLabV3 con una ResNet50 de *backbone* y utilizando un *scheduler* `OneCycleLR()`.

Clase	*Precision*	*Recall*	F1	IOU
ABBA	0.8367	0.7722	0.8031	0.6711
ACPE	0.5268	0.5006	0.5134	0.3453
ACRU	0.7128	0.7469	0.7294	0.5741
ACSA	0.7497	0.5566	0.6389	0.4694
BEAL	0.8461	0.6947	0.7630	0.6168
BEPA	0.8213	0.8671	0.8436	0.7295
FAGR	0.7087	0.7693	0.7378	0.5845
LALA	0.6880	0.8665	0.7670	0.6220
Mort	0.7217	0.4866	0.5813	0.4097
PIST	0.8536	0.8908	0.8718	0.7727
Picea	0.8163	0.7578	0.7859	0.6474
Populus	0.8439	0.9068	0.8742	0.7766
THOC	0.6996	0.8170	0.7537	0.6048
TSCA	0.7896	0.8349	0.8116	0.6830
Fondo	0.6915	0.6897	0.6906	0.5274
Promedio	0.7537	0.7438	0.7444	0.6023

Tab. 4.7: Resultados obtenidos al evaluar el conjunto de evaluación en las distintas métricas para una DeepLabV3 con ResNet50 de *backbone* y con un *scheduler* `OneCycleLR()`.

4.2.2.2. `ReduceLROnPlateau()`

En la Fig. 4.19 se grafican las curvas de validación y entrenamiento según las *epochs*. Si bien este gráfico es similar a la Fig. 4.13, se diferencian principalmente en las últimas 40 *epochs*. En la Fig. 4.19, las métricas para el conjunto de validación se estabilizan y las de entrenamiento crecen en una medida mayor. Sin embargo, en la Fig. 4.13 la pendiente negativa en la función de pérdida del conjunto de entrenamiento sigue siendo aún así muy grande, y la de validación se vuelve positiva.

En la Tabla 4.8 se presenta la *performance* del modelo en las distintas métricas. Fueron mejores que las obtenidas para la DeepLabV3 sin *scheduler* (Tabla 4.6). Si bien no superó en *precision* al modelo de la U-Net con el mismo *scheduler*, sí lo hizo para las restantes tres métricas de la tabla. Además existe un *trade off* entre *precision* y *recall*, con lo cual la pérdida de *precision* también podría ser explicada por la mejora en la otra métrica.

Por último, se visualizaron las clasificaciones hechas por la red en la Fig. 4.20. Los cambios en la red respecto a la Fig. 4.14 lograron que, por ejemplo, la clase de color más oscuro detectada parcialmente antes se detectara en mayor medida en la parte inferior derecha de la imagen de la fila 4. De igual manera, en la primera fila de imágenes la clase de color gris claro sigue siendo confundida por otra más oscura. En la última fila de imágenes, se mitiga el ruido respecto de la clasificación con la DeepLabV3 sin *schedulers*. En la Fig. 4.21 se grafica la matriz de confusión para esta configuración de red.

4.2.2.3. `CosineAnnealingLR()`

Para obtener información acerca del entrenamiento a lo largo de las *epochs*, en la Fig. 4.22 se graficaron las curvas de entrenamiento y validación. Luego de la *epoch* 30 aproximadamente, se observa que las métricas no varían demasiado en la validación, como sí lo hacen en el entrenamiento. Nuevamente, en esta configuración, tal como en las anteriores

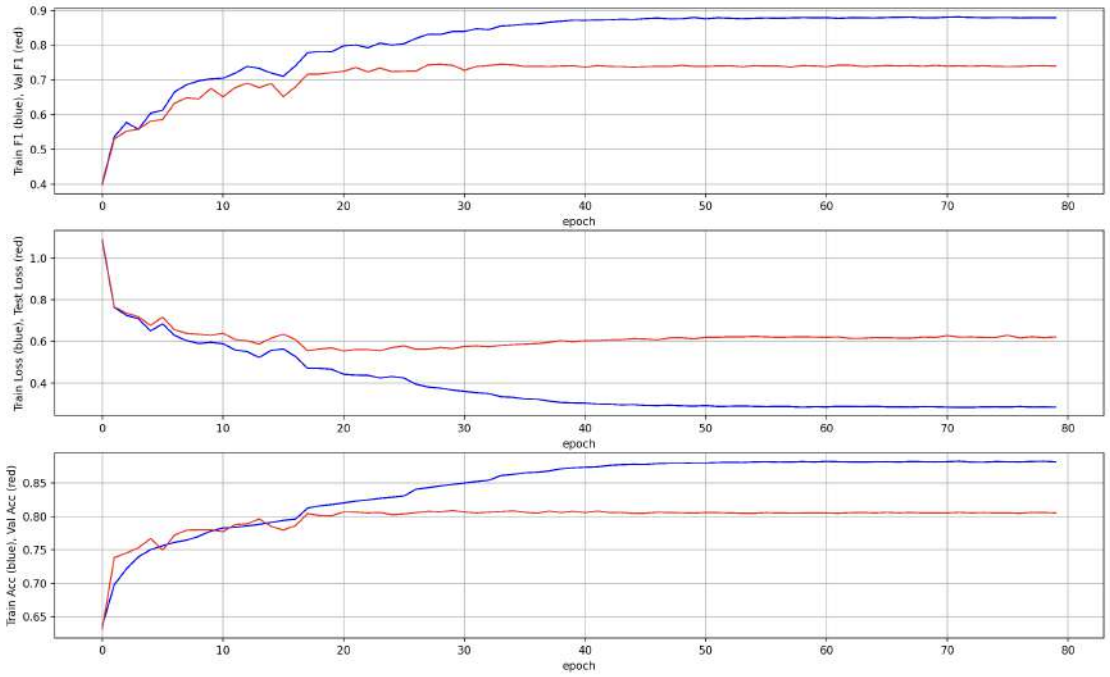


Fig. 4.19: Gráfico de las distintas métricas para el conjunto de entrenamiento (azul) y para el de validación (rojo) a través de las épocas, usando una DeepLabV3 sin preentrenar con una ResNet 50 de *backbone* y un *scheduler* `ReduceLROnPlateau()`.

Clase	*Precision*	*Recall*	F1	IOU
ABBA	0.8272	0.7824	0.8042	0.6725
ACPE	0.5201	0.5142	0.5171	0.3487
ACRU	0.7158	0.7449	0.7300	0.5749
ACSA	0.7250	0.5881	0.6494	0.4809
BEAL	0.8291	0.7012	0.7598	0.6126
BEPA	0.8232	0.8671	0.8446	0.7310
FAGR	0.6186	0.7820	0.6908	0.5276
LALA	0.7251	0.8781	0.7943	0.6588
Mort	0.7933	0.5292	0.6349	0.4651
PIST	0.8433	0.8987	0.8701	0.7701
Picea	0.7739	0.7705	0.7722	0.6289
Populus	0.8504	0.9103	0.8793	0.7846
THOC	0.7072	0.8076	0.7541	0.6052
TSCA	0.6958	0.8445	0.7630	0.6168
Fondo	0.7115	0.6834	0.6972	0.5351
Promedio	0.7440	0.7535	0.7441	0.6009

Tab. 4.8: Resultados obtenidos al evaluar el conjunto de evaluación en las distintas métricas para una DeepLabV3 con ResNet50 de *backbone* y con un *scheduler* `ReduceLROnPlateau()`.

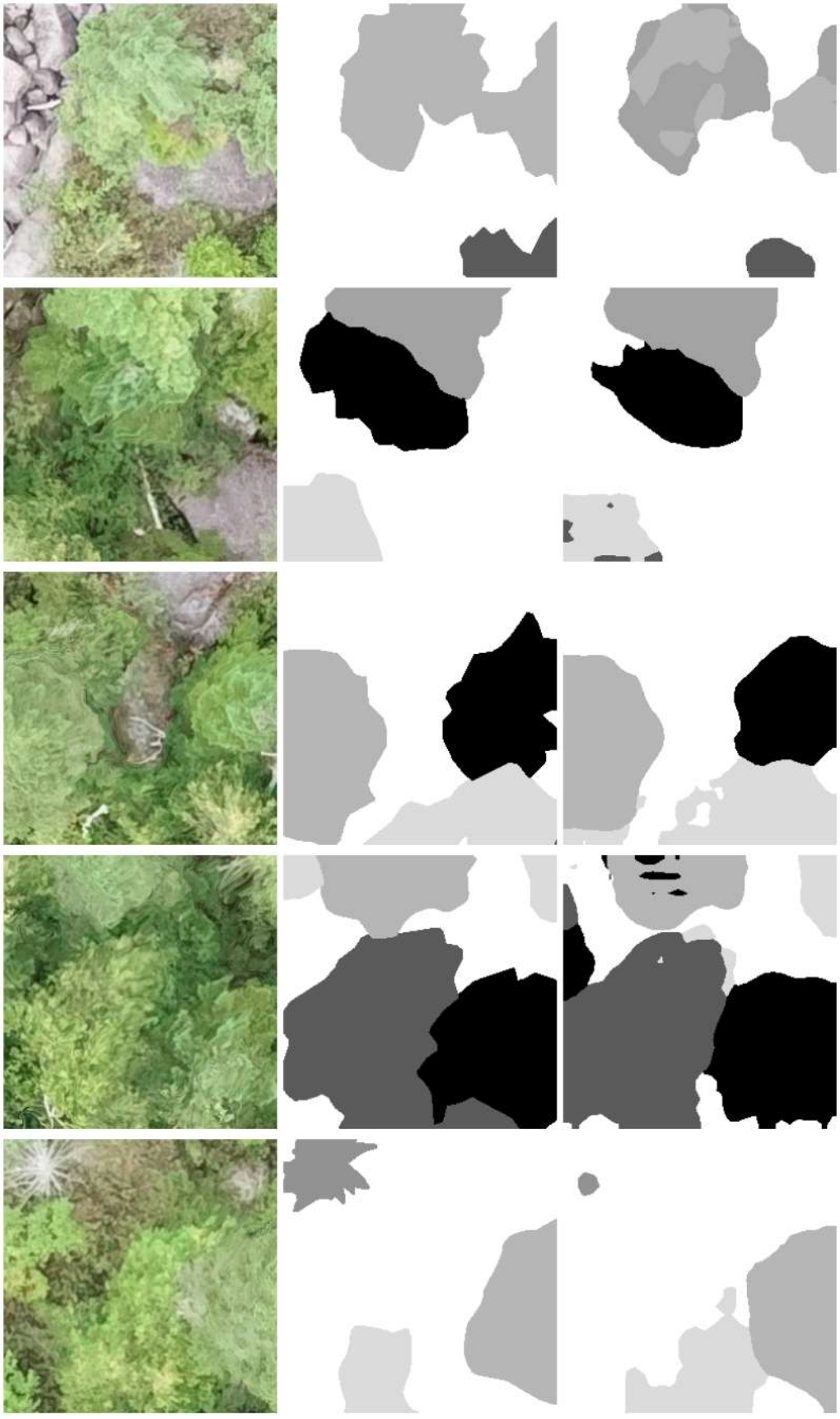


Fig. 4.20: Resultados esperados y obtenidos utilizando una DeepLabV3 con un *backbone* de ResNet50 y un *scheduler* `ReduceLROnPlateau()`.

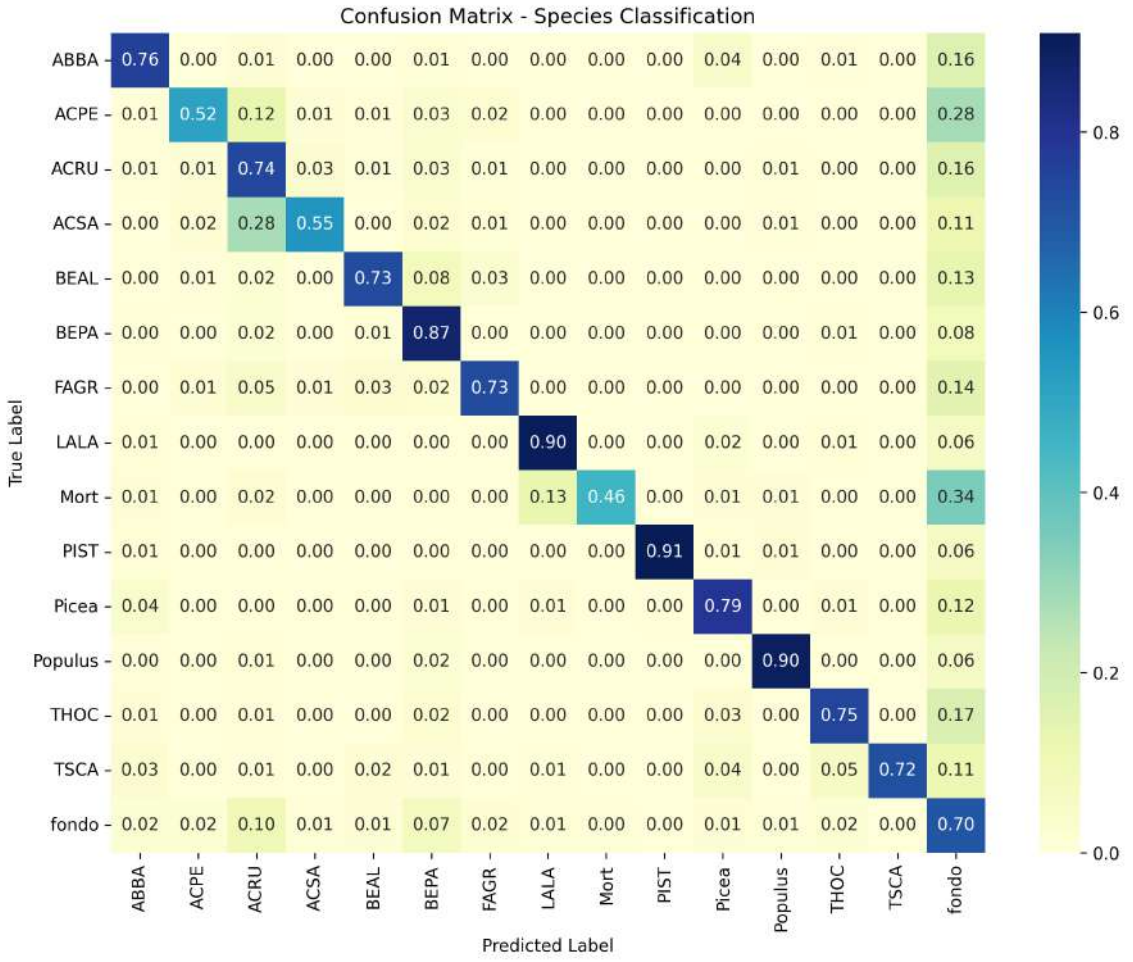


Fig. 4.21: Matriz de confusión obtenida para una DeepLabV3 con una ResNet50 de *backbone* y utilizando un *scheduler* `ReduceLROnPlateau()`.

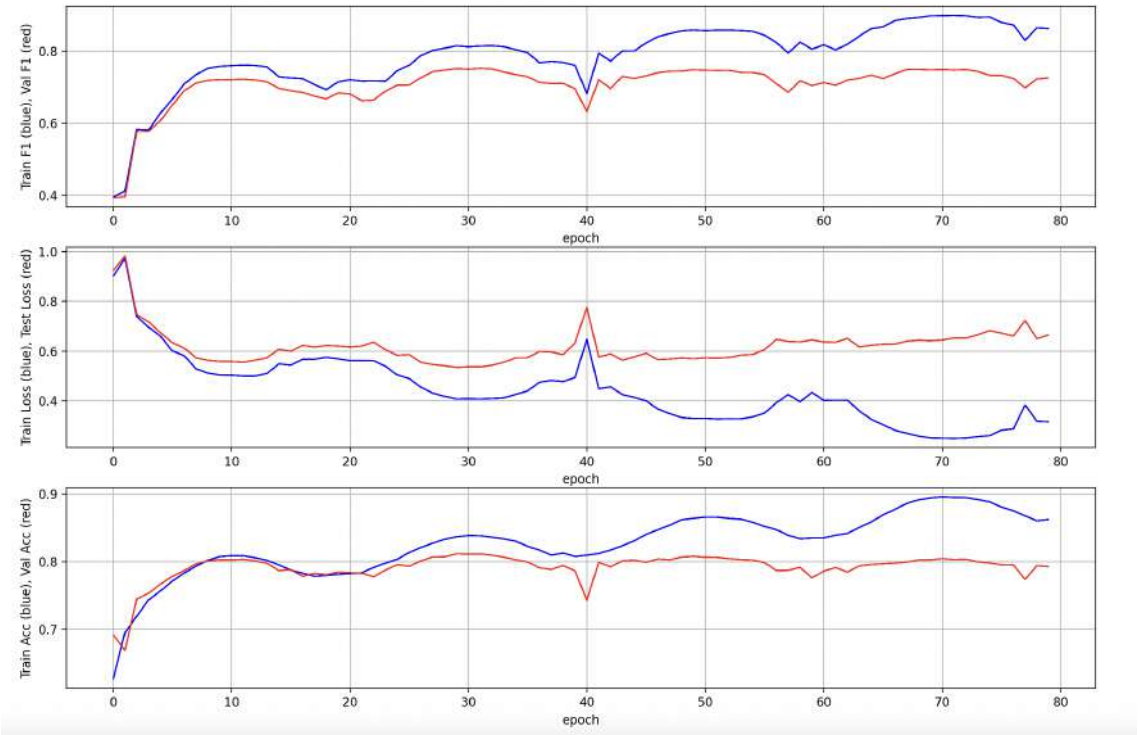


Fig. 4.22: Gráfico de las distintas métricas para el conjunto de entrenamiento (azul) y para el de validación (rojo) a través de las épocas, usando una DeepLabV3 sin preentrenar con una ResNet 50 de *backbone* y un *scheduler* `CosineAnnealingLR()`.

para esta red, el entrenamiento se estabilizó más rápidamente que para la U-Net. Además, aquí también se visualiza que ambas curvas van cambiando su concavidad a lo largo de las *epochs*, con un trazo similar al de Fig. 4.11, el modelo que utilizó el mismo *scheduler* para una U-Net.

En la Tabla 4.9 se presentan los resultados obtenidos para esta configuración, y en la Fig. 4.23 algunas visualizaciones. Además, esta configuración fue la que mejor evaluó en todas las demás métricas de las configuraciones de la DeepLab, y sobrepasó por un 0.0308 al resultado de F1 de la U-Net sin *schedulers*. Entonces, se obtuvo una mejor *performance* que en [9]. La matriz de confusión se presenta en la Fig. 4.24.

4.3. Resumen de resultados utilizando U-Net y DeepLabV3

Luego, la Tabla 4.10 presenta un resumen de los modelos de DeepLabV3 e incluyendo también los de la U-Net de la Tabla 4.5. Cabe remarcar que, incluyendo todos los resultado analizados, la DeepLabV3 con el *scheduler* `CosineAnnealingLR()` fue la configuración de mejor *performance* sobre el conjunto de evaluación.

Clase	*Precision*	*Recall*	F1	IOU
ABBA	0.8451	0.7670	0.8042	0.6725
ACPE	0.4707	0.5676	0.5146	0.3465
ACRU	0.7218	0.7585	0.7397	0.5869
ACSA	0.7543	0.5782	0.6546	0.4866
BEAL	0.8072	0.7383	0.7712	0.6276
BEPA	0.8374	0.8638	0.8504	0.7397
FAGR	0.6086	0.7745	0.6816	0.5170
LALA	0.7422	0.8895	0.8092	0.6796
Mort	0.8018	0.5542	0.6554	0.4875
PIST	0.9121	0.8854	0.8986	0.8158
Picea	0.7788	0.7824	0.7806	0.6402
Populus	0.8559	0.9147	0.8843	0.7926
THOC	0.7074	0.8212	0.7601	0.6130
TSCA	0.7693	0.7583	0.7637	0.6178
Fondo	0.7168	0.6911	0.7037	0.5429
Promedio	0.7553	0.7563	0.7515	0.6111

Tab. 4.9: Resultados obtenidos al evaluar el conjunto de evaluación en las distintas métricas para una DeepLabV3 con ResNet50 de *backbone* y con un *scheduler* `CosineAnnealingLR()`.

Configuración	F1	IOU
U-Net sin *scheduler*	0.7207	0.5716
U-Net con `OneCycleLR()`	0.7337	0.5910
U-Net con `ReduceLROnPlateau()`	0.7392	0.5943
U-Net con `CosAnnealingLR()`	0.7171	0.5670
DeepLabV3 sin *scheduler*	0.7279	0.5808
DeepLabV3 con `OneCycleLR()`	0.7444	0.6023
DeepLabV3 con `ReduceLROnPlateau()`	0.7441	0.6009
DeepLabV3 con `CosineAnnealingLR()`	**0.7515**	**0.6111**

Tab. 4.10: Métricas F1 e IOU obtenidas para cada configuración de redes. En negrita se resalta aquel modelo que obtuvo mejores resultados en ambas.

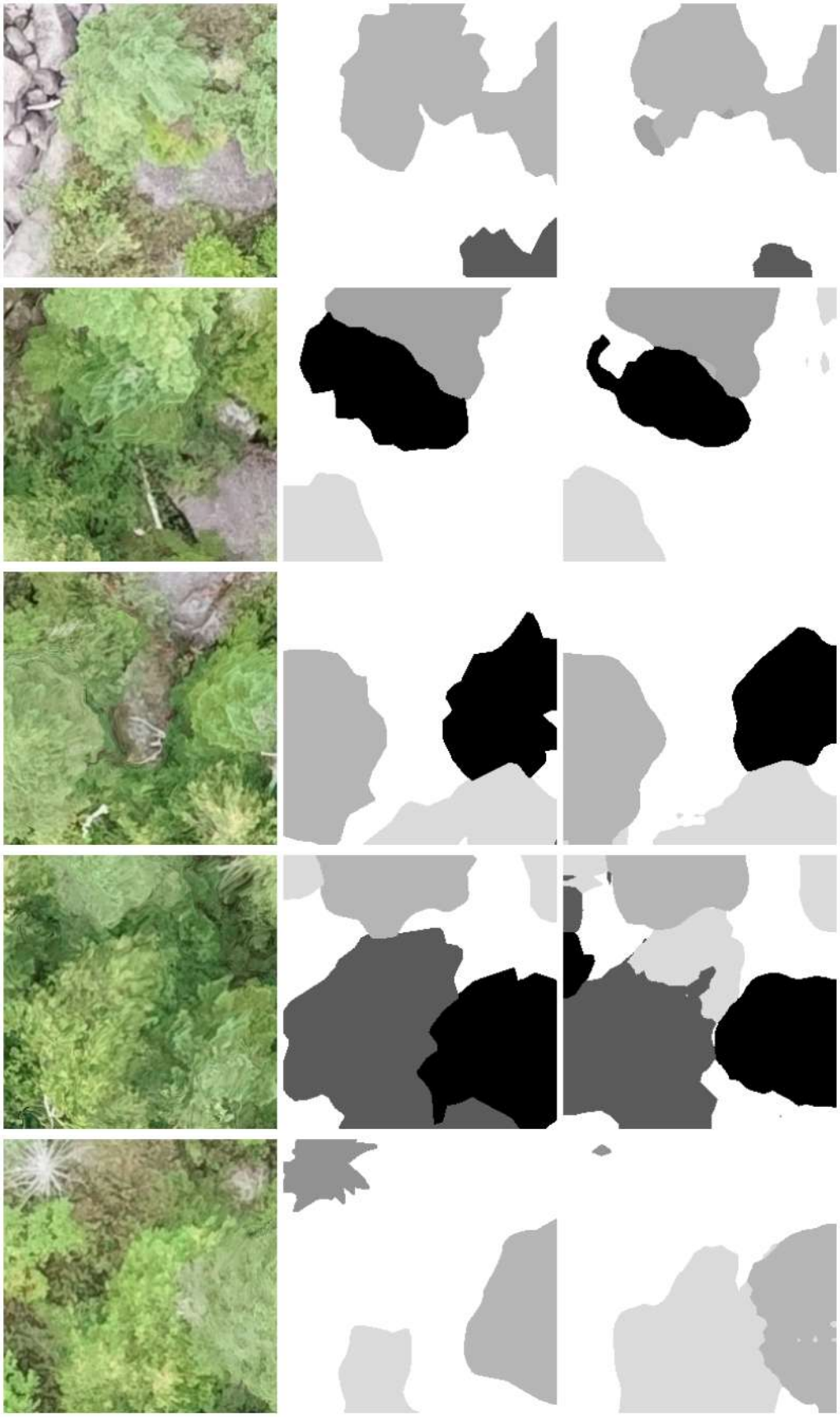


Fig. 4.23: Resultados esperados y obtenidos utilizando una DeepLabV3 con un *backbone* de ResNet50 y un *scheduler* `CosineAnnealingLR()`.

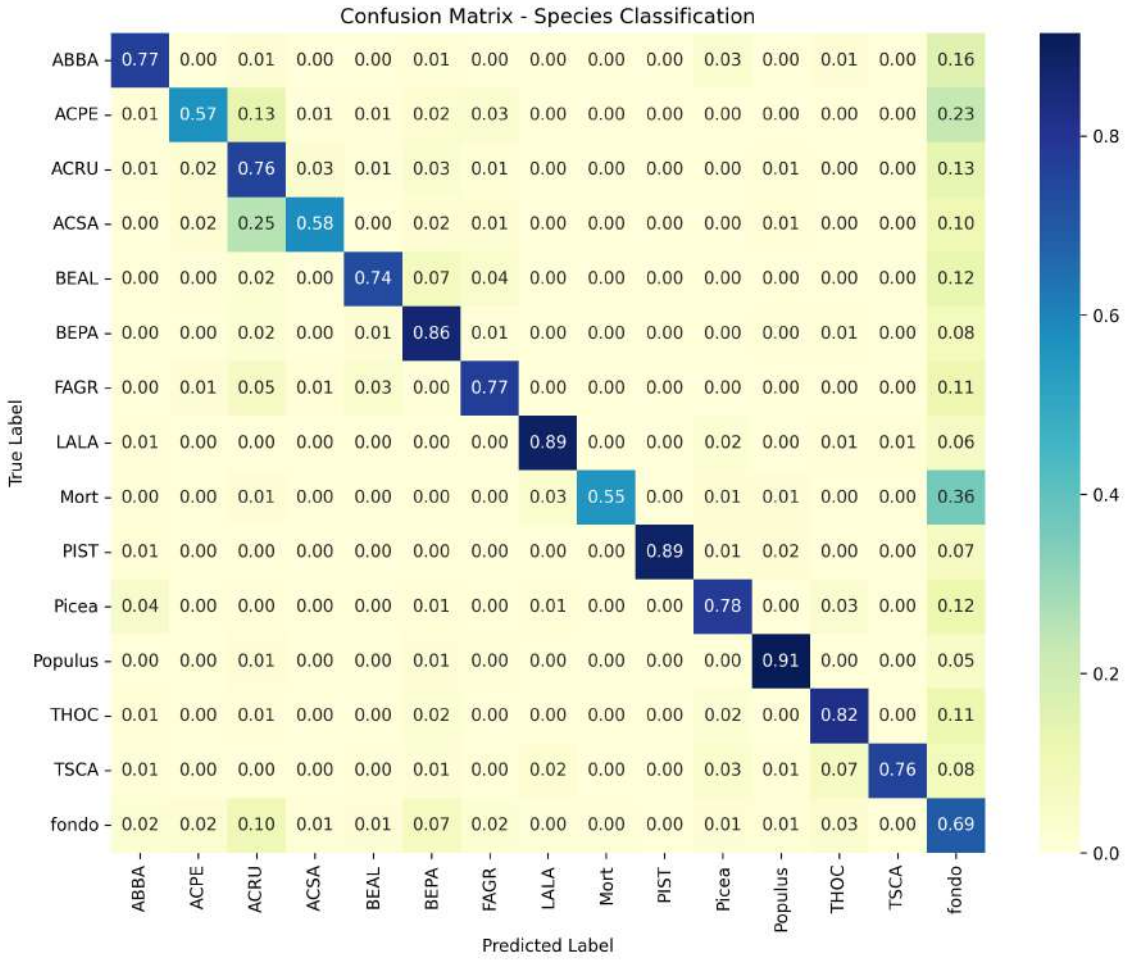


Fig. 4.24: Matriz de confusión obtenida para una DeepLabV3 con una ResNet50 de *backbone* y utilizando un *scheduler* `CosineAnnealingLR()`.

5. CONCLUSIONES

En este trabajo se abordó el desafío de clasificar especies arbóreas a partir de imágenes RGB aéreas utilizando técnicas de aprendizaje profundo. Se logró realizar una segmentación semántica de las especies para la clasificación de las imágenes tomadas.

Se tuvieron que enfrentar complejidades propias de este tipo de datos, ya que no se contaba con información adicional como si lo tienen las cámaras multiespectrales, ni tecnología LiDAR. Ejemplos de esto son la variabilidad en la iluminación, la oclusión - pues algunos árboles tapan partes de otros al tratarse de imágenes aéreas- y la similitud fenotípica entre especies. Cabe destacar además que los drones con cámaras RGB son hoy en día tecnologías de muy bajo costo y accesibles, en comparación con otras herramientas que se utilizan para el estudio de áreas forestales y su clasificación.

Como parte del desarrollo, se llevó a cabo un análisis exhaustivo del estado del arte y la exploración de datasets adecuados para el problema. Asimismo, se replicó el trabajo presentado en el trabajo de Cloutier *et al.* [9]. Se implementaron además modificaciones a la arquitectura original, logrando igualar e incluso mejorar el resultado en el *F1 score* reportado en dicho trabajo. El mejor resultado reportado con esta red fue de 0.7392, utilizando un *scheduler*.

Además, se implementó una arquitectura alternativa, la DeepLabV3, que permitió superar la *performance* obtenida en [9]. Se observó también que los tiempos de entrenamiento para esta red fueron considerablemente menores respecto de los de la U-Net. Con esto se mejoró la clasificación de especies arbóreas en las imágenes utilizadas. Un resumen de las métricas obtenidas para cada modelo se encuentra en la Tabla 4.10.

El mejor resultado obtenido en el *F1-score* en promedio fue de 0.7515 con una DeepLabV3 con ResNet50 de *backbone* y un *scheduler* `CosineAnnealingLR()`. Por clase, los resultados de esta métrica para el mismo modelo se describen en la Tabla 4.9. El mejor resultado obtenido en F1 entre las configuraciones analizadas de una U-Net fue aquella en la que se utilizó un *scheduler* `ReduceLROnPlateau()`, cuyos resultados se muestran en la Tabla 4.3.

El trabajo realizado supuso la utilización del servidor del área de Imágenes y Robótica de la Facultad de Ciencias Exactas y Naturales, al contener el mismo dos GPUs útiles para reducir los tiempos de cómputo de entrenamiento de redes. Asimismo, se utilizó Docker [12] para el encapsulamiento del entorno de ejecución en un contenedor, y PyTorch [25] para la implementación de las redes neuronales.

5.1. Trabajos futuros

En esta sección, se brindarán líneas de investigación para continuar este trabajo dentro del *Laboratorio de Robótica y Sistemas Embebidos (LRSE)*, del ICC-DC.

Un posible camino consiste en extender el enfoque actual incorporando arquitecturas más recientes basadas en *transformers*, que han demostrado un rendimiento sobresaliente en tareas de visión por computadora, especialmente en escenarios donde es necesario capturar relaciones espaciales de largo alcance. Estas redes podrían aportar mejoras sustanciales en la segmentación semántica de especies arbóreas. Otra línea de investigación que se espera explorar es proponer una red propia que mejore los resultados para este

problema en particular.

Además, se plantea como objetivo analizar imágenes aéreas provenientes de regiones locales, particularmente de bosques y entornos naturales en Argentina. Dada la escasez de estudios aplicados a la naturaleza autóctona, cuya composición y características difieren notablemente de las especies canadienses presentes en el *dataset* utilizado en [9], el desarrollo y la evaluación de modelos sobre un *dataset* argentino permitiría no solo avanzar en la investigación científica nacional, sino también generar herramientas de utilidad para la gestión y el monitoreo ambiental y específicamente forestal de nuestro país.

Bibliografía

[1] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *CoRR*, abs/1511.00561, 2015.

[2] Mirela Beloiu, Lucca Heinzmann, Nataliia Rehush, Arthur Gessler, and Verena C. Griess. Individual tree-crown detection and species identification in heterogeneous forests using aerial rgb imagery and deep learning. *Remote Sensing*, 15(5), 2023.

[3] Ishaan Bhargav. Guide to pytorch learning rate scheduling. `https://www.kaggle.com/code/isbhargav/guide-to-pytorch-learning-rate-scheduling`, 2021. Accessed: 2025-06-01.

[4] Ali Bhatti, Muhammad Umer, Syed Adil, Mansoor Ebrahim, Daniyal Nawaz, and Faizan Ahmed. Explicit content detection system: An approach towards a safe and ethical environment. *Applied Computational Intelligence and Soft Computing*, 2018, 07 2018.

[5] Jason Brownlee. Understand the dynamics of learning rate on deep learning neural networks. `https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/`, 2020. Accessed: 2025-06-01.

[6] Kaili Cao and Xiaoli Zhang. An improved res-unet model for tree species classification using airborne high-resolution images. *Remote Sensing*, 12(7), 2020.

[7] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs, 2017.

[8] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *CoRR*, abs/1706.05587, 2017.

[9] Myriam Cloutier, Mickaël Germain, and Etienne Laliberté. Influence of temperate forest autumn leaf phenology on segmentation of tree species from uav imagery using deep learning. *Remote Sensing of Environment*, 311:114283, 2024.

[10] Sally Deborah Pereira da Silva, Fernando Coelho Eugenio, Roberta Aparecida Fantinel, Lucio de Paula Amaral, Alexandre Rosa dos Santos, Caroline Lorenci Mallmann, Fernanda Dias dos Santos, Rudiney Soares Pereira, and Régis Ruoso. Modeling and detection of invasive trees using uav image and machine learning in a subtropical forest in brazil. *Ecological Informatics*, 74:101989, 2023.

[11] Songqiu Deng, Masato Katoh, Xiaowei Yu, Juha Hyyppä, and Tian Gao. Comparison of tree species classifications at the individual tree level by combining als data and rgb images using different algorithms. *Remote Sensing*, 8(12), 2016.

[12] Docker Inc. Docker: Lightweight linux containers for consistent development and deployment. `https://www.docker.com`, 2013.

[13] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning, 2018.

[14] Sebastian Egli and Martin Höpke. Cnn-based tree species classification using high resolution rgb image data from automated uav observations. *Remote Sensing*, 12(23), 2020.

[15] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[17] Yunmei Huang, Botong Ou, Kexin Meng, Baijian Yang, Joshua Carpenter, Jinha Jung, and Songlin Fei. Tree species classification from uav canopy images with deep learning models. *Remote Sensing*, 16(20), 2024.

[18] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: with Applications in R*. Springer Texts in Statistics. Springer, 2nd edition, 2021.

[19] R. Keys. Cubic convolution interpolation for digital image processing. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 29(6):1153–1160, 1981.

[20] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[21] Sizhuo Li, Martin Brandt, Rasmus Fensholt, Ankit Kariryaa, Christian Igel, Fabian Gieseke, Thomas Nord-Larsen, Stefan Oehmcke, Ask Carlsen, Samuli Junttila, Xiaoye Tong, Alexandre d'Aspremont, and Philippe Ciais. Deep learning enables image-based tree counting, crown segmentation and height prediction at national scale. *PNAS Nexus*, 2, 03 2023.

[22] Giorgio Morales, Guillermo Kemper, Grace Sevillano, Daniel Arteaga, Ivan Ortega, and Joel Telles. Automatic segmentation of mauritia flexuosa in unmanned aerial vehicle (uav) imagery using deep learning. *Forests*, 9(12), 2018.

[23] Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. *Distill*, 2016. `https://distill.pub/2016/deconv-checkerboard/`.

[24] Facundo Pessacg, Francisco Gómez-Fernández, Matías Nitsche, Nicolás Chamo, Sebastián Torrella, Rubén Ginzburg, and Pablo De Cristóforis. Simplifying uav-based photogrammetry in forestry: How to generate accurate digital terrain model and assess flight mission settings. *Forests*, 13(2), 2022.

[25] PyTorch Team. PyTorch: An Open Source Machine Learning Framework, 2024. Accessed: 2025-06-02.

[26] QGIS Development Team. *QGIS Geographic Information System*. Open Source Geospatial Foundation Project, 2025. `https://qgis.org`.

[27] Francisco Raverta Capua, Juan Schandin, and Pablo De Cristóforis. Training point-based deep learning networks for forest segmentation with synthetic data. In Apostolos Antonacopoulos, Subhasis Chaudhuri, Rama Chellappa, Cheng-Lin Liu, Saumik Bhattacharya, and Umapada Pal, editors, *Pattern Recognition*, pages 64–80, Cham, 2025. Springer Nature Switzerland.

[28] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[29] Evan Shelhamer, Jonathan Long, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *CoRR*, abs/1605.06211, 2016.

[30] Gurpreet Singh and Manoj Sachan. Multi-layer perceptron (mlp) neural network technique for offline handwritten gurmukhi character recognition. In *2014 IEEE International Conference on Computational Intelligence and Computing Research*, pages 1–5, 2014.

[31] Leslie N. Smith and Nicholay Topin. Super-convergence: Very fast training of residual networks using large learning rates. *CoRR*, abs/1708.07120, 2017.

[32] Tomasz Szandala. Review and comparison of commonly used activation functions for deep neural networks. *CoRR*, abs/2010.09458, 2020.

[33] Jingru Wu, Qixia Man, Xinming Yang, Pinliang Dong, Xiaotong Ma, Chunhui Liu, and Changyin Han. Fine classification of urban tree species based on uav-based rgb imagery and lidar data. *Forests*, 15(2), 2024.

[34] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, José M. Álvarez, and Ping Luo. Segformer: Simple and efficient design for semantic segmentation with transformers. *CoRR*, abs/2105.15203, 2021.

[35] Chong Zhang, Jiawei Zhou, Huiwen Wang, Tianyi Tan, Mengchen Cui, Zilu Huang, Pei Wang, and Li Zhang. Multi-species individual tree segmentation and identification based on improved mask r-cnn and uav imagery in mixed forests. *Remote Sensing*, 14(4), 2022.

[36] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. *CoRR*, abs/1612.01105, 2016.