Clasificación de cobertura terrestre en la Amazonía usando imágenes satelitales

Jonathan Andrés Granda Orrego*, Cristian Daniel Muñoz Botero†, Ana Isabel Patiño Osorio‡
Departamento de Ingeniería de Sistemas, Universidad de Antioquia
Medellín, Colombia
Email: *jonathan.granda@udea.edu.co, †cristian.munoz3@udea.edu.co, ‡anai.patino@udea.edu.co

***Resumen*—Este trabajo explora la aplicación de técnicas de Deep Learning para etiquetar imágenes satelitales de la región amazónica, utilizando un conjunto de datos proporcionado a través de Kaggle. El objetivo de esta herramienta es clasificar de manera precisa lugares de interés en la región, como minas artificiales, zonas de agricultura u otras áreas que presentan alteraciones en la cobertura vegetal a partir de imágenes satelitales multiespectrales. Para ello, se implementan modelos basados en redes neuronales convolucionales, los cuales son entrenados y evaluados con el fin de identificar patrones visuales característicos presentes en las imágenes satelitales asociados a diferentes climas y usos del suelo. El estudio también aborda desafíos inherentes a la clasificación multietiqueta y al desbalance de etiquetas que presenta el dataset. Los anteriores desafíos resaltan la importancia del preprocesado correcto de los datos y la selección adecuada del modelo.**

***Palabras Clave*——Deep Learning, imagen satelital, cobertura vegetal, redes neuronales convolucionales**

I. Introducción

La Amazonía es una de las regiones ecológicas más diversas y estratégicas del planeta, pero también una de las más vulnerables a actividades humanas como la deforestación, la expansión agrícola, la minería ilegal y los incendios. La magnitud de su territorio y la velocidad con la que ocurren estos fenómenos hacen que el monitoreo tradicional resulte insuficiente; revisar manualmente grandes volúmenes de imágenes satelitales es lento, costoso e imposible de practicar a gran escala.

En este contexto, el Deep Learning resulta ser una herramienta eficaz para identificar patrones visuales complejos y abordar tareas de clasificación multi etiqueta en grandes colecciones de imágenes. Este proyecto aplica modelos basados en Redes Neuronales Convolucionales (CNNs) para la clasificación automática de cobertura terrestre en la Amazonía, utilizando el conjunto de datos público Planet: Understanding the Amazon from Space, disponible en Kaggle [1]. Dicho dataset contiene más de 40.000 imágenes etiquetadas con 17 categorías. La complejidad del problema se intensifica por la presencia simultánea de múltiples etiquetas por imagen, la correlación entre categorías y un marcado desbalance en la distribución de clases.

Este trabajo busca no solo construir modelos predictivos capaces de identificar la cobertura terrestre y detectar alteraciones ambientales, sino también comparar las fortalezas y limitaciones de distintos enfoques de Deep Learning frente a un problema real, complejo y altamente desbalanceado. Se exploran una arquitectura de línea base, la aplicación de Transfer Learning y una implementación usando PyTorch. Los resultados permiten dimensionar el potencial de la inteligencia artificial como apoyo al monitoreo ambiental en la Amazonía y ofrecen una base para futuros desarrollos orientados a la vigilancia automatizada de ecosistemas en riesgo.

II. Estructura de los Notebooks

II-A. Acceso a los datos de Kaggle

Para ejecutar correctamente los notebooks es necesario habilitar la descarga del conjunto de datos desde la plataforma Kaggle. Cada notebook incluye en sus primeras celdas la instalación de la biblioteca `kaggle` y una celda destinada a la carga del archivo de credenciales `kaggle.json`, el cual debe ser generado previamente desde la cuenta personal del usuario en Kaggle.

El archivo `kaggle.json` se carga mediante el mecanismo de subida de archivos del entorno de ejecución (por ejemplo, Google Colab). Una vez cargado, el sistema configura automáticamente las credenciales en el directorio correspondiente, permitiendo la descarga del dataset requerido.

- Subir el archivo `kaggle.json` al inicio de cada notebook.
- Verificar que las credenciales queden almacenadas en la ruta adecuada, usualmente: `/root/.kaggle/`.
- Ejecutar los notebooks de manera secuencial para garantizar la disponibilidad de archivos intermedios y evitar errores de dependencia.

II-B. Notebook de exploración de datos

El notebook nombrado *01 - data-exploration.ipynb* tiene como propósito analizar las características principales del conjunto de datos y comprender la estructura de sus etiquetas antes de proceder al entrenamiento de los modelos (EDA). Este notebook se compone de 4 secciones: carga de datos, Visualización de distribución de etiquetas, visualización de imágenes asociadas a las clases, estadísticas descriptivas y exploración de correlaciones.

En primer lugar, se realiza la carga del conjunto de datos y se extraen las etiquetas asociadas a cada imagen. El dataset contiene un total de 40 479 imágenes de entrenamiento y 40 669 imágenes de prueba(estas no tienen etiquetas asociadas), con 17 clases únicas. Cada imagen puede presentar

múltiples etiquetas, con un promedio de 2.87 etiquetas por imagen, un mínimo de una y un máximo de nueve.

En la Figura 1 se muestra la distribución del número de etiquetas por imagen. La mayoría de las observaciones contienen entre dos y cuatro etiquetas, lo cual confirma que se trata de un problema de clasificación multietiqueta con una importante superposición semántica o relaciones entre estas categorías, es decir, varias categorias que establecen entre estas significados que son similares, pero no idénticos, por ejemplo categorias que estan describiendo el terreno.

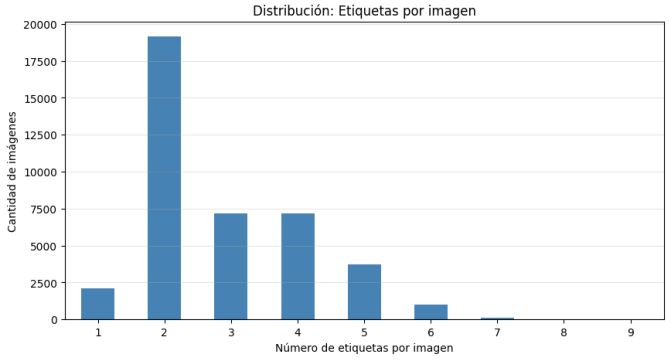

Figura 1. Distribución del número de etiquetas por imagen.

Posteriormente, se analiza la frecuencia de aparición de combinaciones de etiquetas, es decir, que combinaciones de etiquetas son mas frecuentadas en el dataset. En la Figura 2 se presentan las 15 combinaciones más comunes. Se observa que las combinaciones *clear primary* y *partly_cloudy primary* concentran una gran proporción del total, mientras que otras combinaciones más específicas presentan frecuencias significativamente menores, y tambien notamos que tenemos pocas imagenes con muchas etiquetas al mismo tiempo.

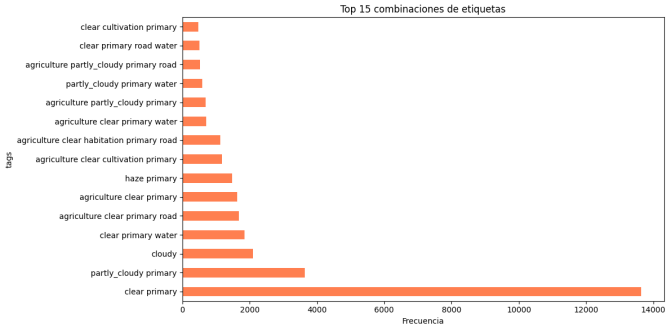

Figura 2. Top 15 combinaciones de etiquetas más frecuentes.

Además de la frecuencia de combinaciones, se examina la correlación entre las etiquetas más comunes con el fin de identificar patrones de coocurrencia relevantes para el proceso de aprendizaje. Cabe aclarar que con las demás etiquetas no se presenta mucha correlación, se cree que es por la poca aparicición de estas etiquetas en las muestras de entrenamiento. La Figura 3 muestra un mapa de calor de correlación entre las diez etiquetas con mayor número de apariciones. Destacan correlaciones positivas moderadas entre etiquetas como *agriculture* y *road* (0.48), así como correlaciones negativas pronunciadas entre *cloudy* y *primary* (-0.83). Estas relaciones sugieren dependencias semánticas que los modelos deberán aprender para mejorar su capacidad de predicción multietiqueta.

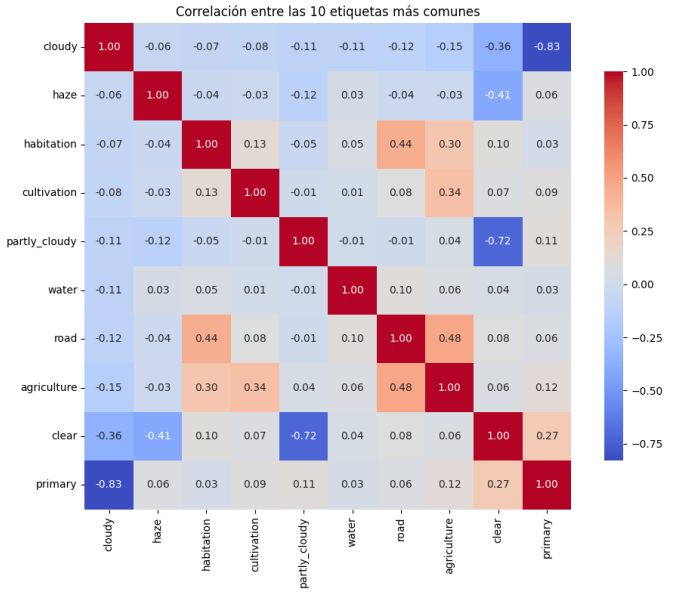

Figura 3. Correlación entre las 10 etiquetas más comunes.

Finalmente, el notebook incluye ejemplos visuales por clase, así como estadísticas descriptivas sobre la distribución de las etiquetas. Se resalta que existen clases con muy pocas instancias, como *blow_down* (98 imágenes) y *conventional_mine* (100 imágenes), lo cual representa un reto adicional debido al desbalance del dataset.

Ademas, es importante destacar el marcado desbalance de clases presente en el conjunto de datos. Aunque el problema es multietiqueta, la frecuencia con la que aparecen las distintas categorías varía considerablemente: algunas clases como *primary* y *clear* cuentan con decenas de miles de ocurrencias, mientras que otras, como *blow_down*, *conventional_mine* o *slash_burn*, apenas están representadas. Esta disparidad implica que el modelo podría sesgarse hacia las clases más comunes si no se emplean estrategias de mitigación como ponderación de pérdidas, técnicas de sobremuestreo o generación de datos sintéticos. En consecuencia, el desbalance introduce un reto adicional, pues obliga al modelo a aprender patrones relevantes incluso para aquellas categorías con escasa presencia en el entrenamiento.

Este desbalance es especialmente relevante en tareas de clasificación multietiqueta, pues la presencia dominante de clases frecuentes puede sesgar el modelo durante el entrenamiento, dificultando la correcta identificación de categorías minoritarias. Además, la gran disparidad entre clases mayoritarias (por ejemplo, *primary* o *clear*) y clases poco representadas incrementa la necesidad de emplear estrategias de mitigación,

como ponderación de pérdidas, técnicas de sobremuestreo o arquitecturas más robustas frente a datos desbalanceados.

II-C. Notebook de preprocesado

En el notebook denominado `02 - data-preprocessing.ipynb` se documenta el flujo de preparación de los datos previo al entrenamiento de los modelos. Su propósito general es transformar las imágenes y etiquetas del conjunto de datos en un formato adecuado, uniforme y eficiente para tareas de clasificación multietiqueta.

El notebook describe las etapas necesarias para organizar el conjunto de datos, reestructurar las etiquetas y preparar las imágenes para los modelos de aprendizaje profundo. Además, implementa un mecanismo de carga por lotes que permite entrenar redes neuronales de manera eficiente incluso con grandes volúmenes de datos.

Las operaciones realizadas en este notebook se pueden resumir en los siguientes puntos:

- Organización y lectura del conjunto de datos original.
- Construcción del esquema de etiquetas para clasificación multietiqueta.
- Preparación de las imágenes para ser procesadas por las arquitecturas propuestas.
- Implementación del generador de datos utilizado durante el entrenamiento.

Estas etapas establecen la base que permitirá, posteriormente, entrenar modelos robustos y eficientes sobre el conjunto de datos preprocesado. Un buen preprocesamiento no solo mejora la calidad de los datos, sino que también facilita que las arquitecturas aprendan patrones relevantes sin verse afectadas por redundancias, variaciones en el tamaño o ruido propio del dataset original. Además, garantiza que el proceso de entrenamiento sea más estable y que los recursos computacionales se aprovechen de manera óptima, evitando cargas innecesarias de memoria o tiempos excesivos de cómputo.

II-D. Notebook de arquitectura de línea base

En el notebook nombrado *03 - baseline-architecture.ipynb* se construyó una primera aproximación al modelo encargado de resolver el problema de clasificación multietiqueta. El objetivo principal de esta etapa fue desarrollar una arquitectura sencilla pero funcional que permitiera validar el flujo completo del pipeline: desde la carga de datos y el preprocesamiento hasta el entrenamiento, la evaluación y la generación de predicciones.

El notebook sigue una estructura organizada en varias fases:

1. **Preparación de los datos**: Se cargan los datos como en los otros notebooks y se definen los generadores de entrenamiento y validación, responsables de cargar las imágenes por lotes y de suministrar sus etiquetas en formato multi-etiqueta.
2. **Definición del modelo base**: se construye una red neuronal convolucional simple compuesta por tres bloques convolucionales con capas de agrupación, seguida de un clasificador denso con activación sigmoidal para producir las probabilidades independientes de cada clase. Esta arquitectura sirve como punto de partida para comprender el comportamiento inicial del problema sin sofisticaciones adicionales.
3. **Proceso de compilación y entrenamiento**: el modelo se entrena durante varias épocas (5 epocas) utilizando métricas clave para problemas multietiqueta, como precisión, recall y F2-Score. Se registran tanto las métricas de entrenamiento como de validación, lo que permite analizar la estabilidad y la capacidad de generalización del modelo.
4. **Análisis visual del aprendizaje**: posteriormente se representan gráficamente la evolución de las métricas (loss, accuracy, precision, recall y F2-Score), lo cual permite identificar tendencias claras como sobreajuste, estabilidad o mejoras progresivas. Este análisis proporciona información esencial para iteraciones posteriores.
5. **Evaluación del modelo**: finalizado el entrenamiento, se calcula el rendimiento sobre el conjunto de validación y se obtiene un F2-Score global que resume la calidad del modelo bajo una métrica más apropiada para datasets desbalanceados.
6. **Predicciones y análisis cualitativo**: se seleccionan ejemplos aleatorios del conjunto de validación para comparar las etiquetas reales con las predicciones generadas por el modelo. Esto permite identificar patrones de error y comportamientos relevantes, como etiquetas sistemáticamente omitidas o predicciones exitosas ante escenas complejas.
7. **Rendimiento por clase**: finalmente, se realiza un análisis detallado por categoría mediante métricas individuales de precisión, recall y F2-Score. Esto permite identificar clases particularmente difíciles, usualmente asociadas a la escasez de muestras, baja variabilidad visual o fuerte superposición semántica.

Este modelo de línea base no tiene como objetivo alcanzar el mejor desempeño posible, sino establecer un punto de comparación sobre el cual desarrollar iteraciones posteriores. Su función principal es garantizar que el pipeline general funciona adecuadamente y que el flujo de datos entre cada etapa es correcto, lo cual permite avanzar hacia arquitecturas más avanzadas y estrategias de mejora fundamentadas.

II-E. Notebook de solución usando Transfer Learning

El notebook denominado `04 - transfer-learning.ipynb` presenta una alternativa más avanzada al enfoque de la arquitectura base, aprovechando modelos preentrenados en grandes corpus de imágenes. Aunque mantiene la misma estructura general del proceso descrito: carga de datos, preprocesamiento, definición del modelo y entrenamiento, su principal diferencia radica en el uso de redes profundas ya entrenadas que actúan como extractores de características.

En este notebook se reemplaza la construcción manual de la arquitectura por un modelo preentrenado (ResNet),

al cual se le añaden capas finales adaptadas a la clasificación multietiqueta del dataset. Este enfoque pretende reducir significativamente el tiempo de entrenamiento y mejorar el desempeño inicial del modelo, al partir de representaciones visuales previamente aprendidas.

Además, se implementan estrategias específicas de Transfer Learning, como el congelamiento inicial de capas, stas modificaciones permiten obtener un modelo más robusto y eficiente que el construido desde cero, sin repetir de forma innecesaria los detalles ya explicados en la sección del modelo base.

II-F. Notebook de solución usando modelos Torch

Finalmente, en el notebook denominado `05 - torch-models.ipynb` se desarrolla una alternativa al enfoque previo, implementando una solución completa empleando la librería *PyTorch*. A diferencia de los notebooks anteriores, este se estructura bajo la lógica propia del ecosistema Torch, lo que implica la creación de un objeto `Dataset` personalizado, el uso de transformaciones específicas para las imágenes y la definición explícita de las funciones de entrenamiento, evaluación y visualización.

Este notebook también incorpora un entorno de ejecución con GPU para aprovechar la aceleración por hardware, lo cual resulta especialmente relevante dado el tamaño del conjunto de datos y la naturaleza multietiqueta de la tarea. Además, se construye un flujo de procesamiento que permite preparar tanto el conjunto de entrenamiento como el de inferencia, facilitando la lectura de imágenes, la aplicación de transformaciones y la generación de lotes de datos compatibles con modelos definidos en PyTorch.

En conclusión, este notebook ofrece una implementación distinta, mas flexible que permite explorar una solución al problema utilizando la infraestructura característica de PyTorch, manteniendo el mismo objetivo general de los ejercicios anteriores pero con una organización y una forma de trabajo propias de esta librería.

III. Descripción de las soluciones

III-A. Preprocesado del dataset

El preprocesamiento del dataset tiene como objetivo garantizar que todas las imágenes y etiquetas se presenten a los modelos en un formato uniforme, normalizado y adecuado para las tareas de clasificación multietiqueta. Para ello, se implementan una serie de pasos que permiten estandarizar tanto la estructura visual de las imágenes como la representación de sus etiquetas.

Un primer paso consiste en cargar el archivo `train.csv`, el cual contiene los nombres de las imágenes junto con sus etiquetas textuales. Estas etiquetas se transforman en una representación multi-hot, es decir, un vector binario de 17 posiciones donde cada entrada señala la presencia (1) o ausencia (0) de una categoría específica. Este formato es fundamental en problemas multietiqueta, ya que cada imagen puede pertenecer simultáneamente a varias clases.

Posteriormente, se verifica que todas las imágenes poseen un tamaño original de 256×256 píxeles. A pesar de esta uniformidad, las imágenes se redimensionan a 128×128 para reducir el costo computacional del entrenamiento y disminuir el número de parámetros requeridos por las redes neuronales.

Para ello se emplea la función `load_and_preprocess_image`, cuyo rol es cargar una imagen desde su nombre de archivo, ajustarla al tamaño objetivo y normalizar sus valores de píxel al rango $[0, 1]$. De manera conceptual, esta función garantiza que cada imagen ingrese a la red bajo condiciones homogéneas de resolución, escala e intensidad, facilitando un aprendizaje más estable.

A continuación, el dataset se divide en dos subconjuntos: 80 % para entrenamiento y 20 % para validación, como se observa en la Figura 5. La carpeta de test de nuestro dataset proveniente de la competición Planet: Understanding the Amazon from Space [2] no se emplea en el entrenamiento debido a que las imágenes no cuentan con etiquetas, dado que originalmente se utilizaban para las evaluaciones de dicha competencia.

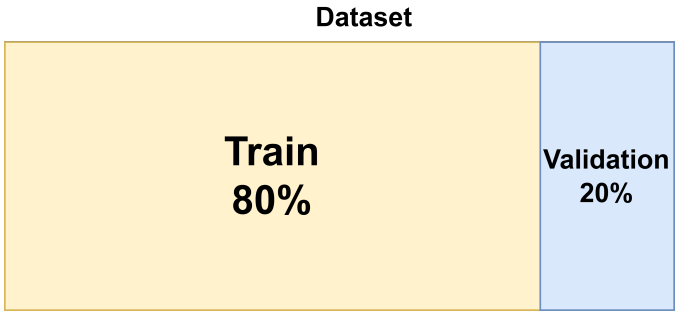

Figura 5. División del dataset

Otro componente clave es el generador de datos (`data_generator`). Este mecanismo permite cargar las imágenes en lotes (*batches*), procesarlas en tiempo real y entregarlas al modelo durante el entrenamiento. El generador evita cargar todo el dataset en memoria simultáneamente, lo cual es crucial dada su magnitud. Además, permite mezclar el orden de las imágenes en cada época para prevenir que la red aprenda patrones derivados del ordenamiento del dataset.

Finalmente, se visualizan ejemplos de batches preprocesados para verificar cualitativamente que tanto las imágenes como sus etiquetas están siendo cargadas correctamente. También se define la configuración de entrenamiento, incluyendo el tamaño del lote, el número de pasos por época y la cantidad total de imágenes empleadas en cada partición.

En conjunto, el proceso de preprocesamiento genera un conjunto de 40 479 imágenes normalizadas, redimensionadas a 128×128 y codificadas mediante un vector binario de 17 clases posibles, constituyendo la base indispensable para el entrenamiento de los modelos posteriores.

III-B. Solución base (Baseline Architecture)

La primera arquitectura desarrollada fue una red neuronal convolucional que hace uso de DropOut para entrada a la última capa densa y de capas de MaxPooling. Las capas están organizadas de la siguiente manera:

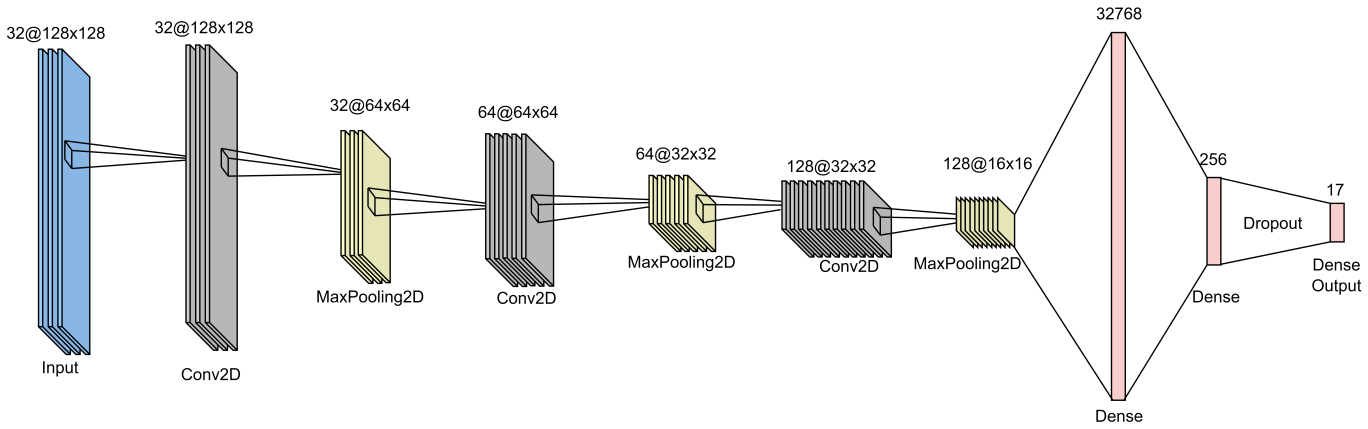

Figura 4. Arquitectura base

Las dimensiones de las entradas se dan de la siguiente manera: nro_imgs_entrada@widthxheight.

1. Capa convolucional 2D 32@128x128
2. Capa MaxPooling 2D 32@64x64
3. Capa convolucional 2D 64@64x64
4. Capa MaxPooling 2D 64@32x32
5. Capa convolucional 2D 128@32x32
6. Capa MaxPooling 2D 128@16x16
7. Capa Densa 32768 neuronas
8. Capa Densa 256 neuronas
9. Capa Densa 17 neuronas (Output)

Tal como se describió anteriormente y como se puede observar en la Figura 4 se formuló una arquitectura simple que aumenta progresivamente la cantidad de filtros para que la red aprenda representaciones más complejas y abstractas. Al final se conecta con capas densas y termina con una capa densa de 17 neuronas que representa cada una de las etiquetas posibles.

III-C. Solución con Transfer Learning

Durante el desarrollo del proyecto se decidió implementar un modelo haciendo uso de Transfer Learning, en este caso con una red neuronal con arquitectura ResNet50, aprovechando el conocimiento de extracción de características adquirido previamente con el dataset masivo ImageNet. Esto con el objetivo de mejorar en las métricas de validación que conseguimos en la anterior implementación.

La arquitectura principal es un modelo híbrido que combina la red ResNet50 preentrenada con una nueva cabeza de clasificación adaptada a la tarea específica.

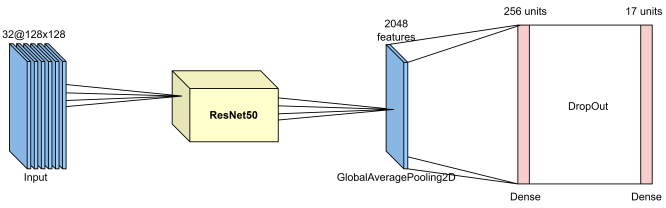

Figura 6. Arquitectura con Transfer Learning

Como se puede observar en la Figura 6 pasamos el input en batches de 32 imagenes preprocesadas y vectorizadas a la arquitectura ResNet50. Las capas pertenecientes a ResNet50, con los pesos preentrenados de ImageNet las 'congelamos'. Esto es, pasamos el parámetro 'trainable=false' a esas capas, con el objetivo de que al hacer el entrenamiento de las capas adicionales que agregamos. Y adicional a esto, excluímos la capa de clasificación de ResNet para agregar una capa de clasificación de 17 etiquetas, acorde a nuestro problema.

Antes de la salida del modelo se agrega capa de GlobalAveragePooling2D con el objetivo de reducir las dimensiones espaciales de la salida convolucional y aplanar la información sin pérdida. Posteriormente agregamos una capa densa de 256 neuronas con activación ReLu para aprender a mapear las características resumidas. A esta capa le aplicamos un DropOut del 0.5 % para prevenir el sobreajuste de la red.

Por último, agregamos una capa densa de 17 neuronas con activación sigmoide, que representan las salidas, cada una de las clases que tenemos en nuestro problema.

El proceso de compilación del modelo se realiza bajo las premisas de una clasificación multietiqueta y con clases desbalanceadas. Agregamos un optimizador Adam con una tasa de aprendizaje baja (0.0001) para que los ajustes de los pesos se hagan de manera meticulosa. Usamos la función de pérdida 'binary_crossentropy' que evalúa cada clase de manera independiente.

Los callbacks los configuramos para guardar la mejor pérdida de validación con el objetivo de escoger el modelo que mejor generalice y añadimos early_stopping con una paciencia

de 3 épocas. El entrenamiento del modelo se realiza durante 5 épocas.

III-D. Solución con modelo Torch

Antes de describir la implementación, es importante mencionar que la estructura general de esta solución está inspirada en el excelente trabajo publicado por Antonio Rueda-Toicen en Kaggle [3], cuyo notebook sirvió como punto de partida conceptual para la organización del *Dataset*, la preparación de transformaciones y el flujo de entrenamiento en PyTorch. A partir de esta base, se extendió y adaptó la propuesta para ajustarse a los objetivos específicos del proyecto, incorporando modificaciones en la arquitectura, métricas, funciones auxiliares y manejo del entrenamiento en GPU.

En el notebook `05 - torch-models.ipynb` se desarrolló una solución alternativa utilizando la librería *PyTorch*, lo que permitió un control más granular sobre el proceso de entrenamiento y una mayor flexibilidad en la definición de la arquitectura. A diferencia de los enfoques previos, este notebook se ejecutó en un entorno acelerado por GPU, optimizando significativamente los tiempos de cómputo, en este caso, se entrenaron con 15 epocas y el tiempo de entrenamiento fue muchisimo menor a las anteriores arquitecturas.

Para la fase de preparación de datos, se implementa un *Dataset* personalizado (`PlanetDataset`), encargado de leer manualmente cada imagen desde el disco, aplicar transformaciones específicas de entrenamiento o validación, y construir las etiquetas multi-etiqueta (*multi-label*) mediante un vector binario de dimensión igual al número total de clases. Esto permite un control granular sobre la representación de las etiquetas, algo particularmente útil en problemas de clasificación con múltiples etiquetas simultáneas.

Asimismo, las transformaciones se definen explícitamente mediante la librería `torchvision.transforms.v2`. Para el conjunto de entrenamiento se emplean operaciones de aumento como *RandomResizedCrop* y *RandomHorizontalFlip*, mientras que para validación se utiliza un preprocesamiento más estructurado basado en *Resize* y *CenterCrop*. Ambas configuraciones incorporan conversión a tensores y normalización utilizando las estadísticas de ImageNet, lo que garantiza compatibilidad con los pesos preentrenados empleados posteriormente.

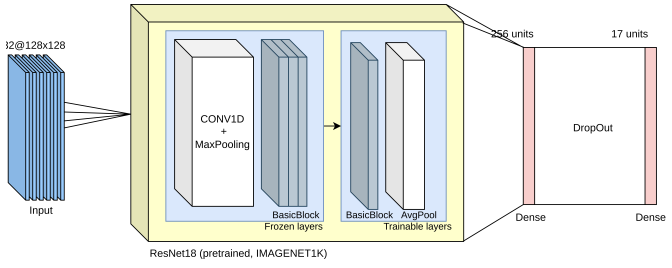

Figura 7. Arquitectura con Torch

En cuanto a la arquitectura (Figura 7), el notebook construye un modelo basado en `ResNet18` con pesos preentrenados en ImageNet. Las capas iniciales de la red se congelan parcialmente para preservar los filtros ya aprendidos, mientras que la capa totalmente conectada final se reemplaza por una nueva capa lineal adaptada al número de clases del conjunto que se trabaja. Este diseño sigue la lógica de *Transfer Learning*, pero implementada de forma manual: la selección de capas congeladas, el reemplazo del clasificador y la transferencia a GPU se controlan directamente desde el código (cuando usamos `model.to(device)`).

En este caso particula la optmización también se programa explícitamente. La función `train()` ejecuta la pasada hacia adelante, el cálculo de la pérdida, la retropropagación y la actualización de parámetros. De manera complementaria, el método `train_model()` gestiona el entrenamiento completo por épocas, incluyendo el cálculo de la métrica F2 durante la validación, el registro de pérdidas y tasas de aprendizaje, y el almacenamiento del mejor modelo mediante un sistema de *checkpoints*. La utilización de un planificador de tasa de aprendizaje (*scheduler*) y la evaluación continua permiten seguir con precisión el comportamiento del modelo a lo largo del entrenamiento.

Este notebook también incorpora utilidades adicionales como la función `denormalize()` para reconstruir imágenes en espacio RGB y la función `plot_batch()` para visualizar muestras junto con sus etiquetas, lo cual facilita inspeccionar el comportamiento del conjunto de datos y verificar la coherencia del preprocesamiento aplicado.

IV. Resultados

IV-A. Resumen

En este caso, se presentan algunos de los datos más relevantes del problema abordado. El conjunto de datos corresponde a un escenario de clasificación multietiqueta. A continuación, podemos ver el resumen de las principales características estadísticas del dataset:

- Total de imágenes de entrenamiento: 40 479
- Total de imágenes de prueba: 40 669
- Número de clases únicas: 17
- Promedio de etiquetas por imagen: 2.87
- Mínimo de etiquetas por imagen: 1
- Máximo de etiquetas por imagen: 9
- Etiqueta más común: *primary* (37 513 apariciones)
- Etiqueta menos común: *blow_down* (98 apariciones)
- Proporción aproximada de imágenes con más de tres etiquetas: 34 %
- Cantidad de combinaciones únicas de etiquetas presentes en el dataset: 4 206
- Distribución de etiquetas altamente desbalanceada, con una relación de aproximadamente 380:1 entre la clase mayoritaria y la minoritaria.
- Resolución típica de las imágenes: 256×256 píxeles
- Tamaño total del dataset (imágenes en disco): ∼1.5 GB (train)

IV-B. Desempeño de la arquitectura base

La arquitectura base presenta un proceso de aprendizaje estable, reflejado en una disminución consistente del *loss* de entrenamiento y una reducción más moderada del *loss* de validación, tal como se aprecia en la Figura 8. Las métricas globales de validación nos dan un *F2-score* moderado, mientras que el *recall* permanece significativamente más bajo. Este comportamiento indica que el modelo tiende a evitar predicciones incorrectas, aunque todavía omite una proporción relevante de etiquetas positivas.

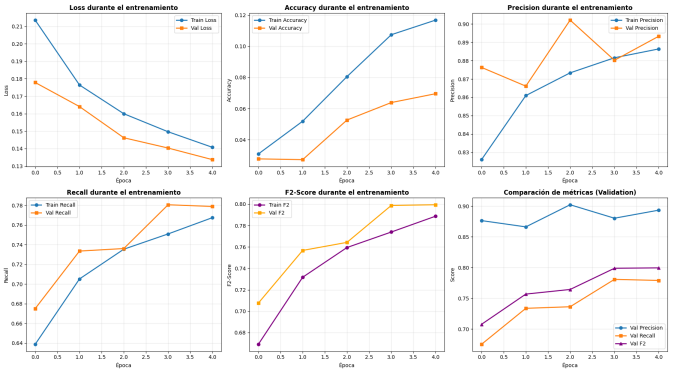

Figura 8. Evolución de las métricas de entrenamiento y validación para el modelo base.

El análisis detallado por clase, resumido en la Tabla I, evidencia un desempeño desigual. En las clases más frecuentes, como *primary*, *clear*, *partly_cloudy* y *cloudy*, el modelo logra valores altos de recall y F2-score (que son las metricas que nos interesan), lo que revela una buena capacidad para capturar patrones dominantes en el conjunto de datos. Sin embargo, en clases de frecuencia media, si bien existe cierto nivel de detección, el valor del *recall* disminuye de manera notable, reflejando dificultades para generalizar características menos comunes.

Las limitaciones más pronunciadas se observan en las clases raras, tales como *blooming*, *conventional_mine*, *blow_down*, *selective_logging* y *slash_burn*, donde el modelo no genera predicciones correctas. La ausencia de ejemplos suficientes, combinada con la alta complejidad visual de estas categorías, impide que la arquitectura base incorpore señales significativas durante el entrenamiento.

En conclusión, de las 17 clases presentes en el problema, el modelo logra un nivel aceptable de generalización en aproximadamente 12 de ellas, mientras que el resto queda fuera de su alcance predictivo. Este resultado indica que la arquitectura base constituye un punto de partida funcional, pero insuficiente para abordar la totalidad de la complejidad del conjunto de datos.

Clase	Precision	Recall	F2
primary	0.97	0.98	0.98
clear	0.96	0.95	0.95
partly_cloudy	0.89	0.86	0.86
cloudy	0.75	0.84	0.82
agriculture	0.80	0.70	0.72
road	0.77	0.58	0.61
haze	0.74	0.52	0.55
artisinal_mine	0.71	0.36	0.39
habitation	0.69	0.34	0.38
water	0.69	0.34	0.38
cultivation	0.52	0.19	0.22
bare_ground	0.61	0.06	0.07
blooming	0.00	0.00	0.00
conventional_mine	0.00	0.00	0.00
blow_down	0.00	0.00	0.00
selective_logging	0.00	0.00	0.00
slash_burn	0.00	0.00	0.00

Cuadro I
RESULTADOS POR CLASE DEL MODELO BASE.

IV-C. Desempeño del modelo con transfer learning

El modelo basado en *transfer learning* muestra un comportamiento de aprendizaje estable pero no mejor que el anterior, continua reflejando en una disminución consistente del *loss* de entrenamiento y un *loss* de validación ligeramente decreciente, sin señales evidentes de sobreajuste, tal como se observa en la Figura 9.

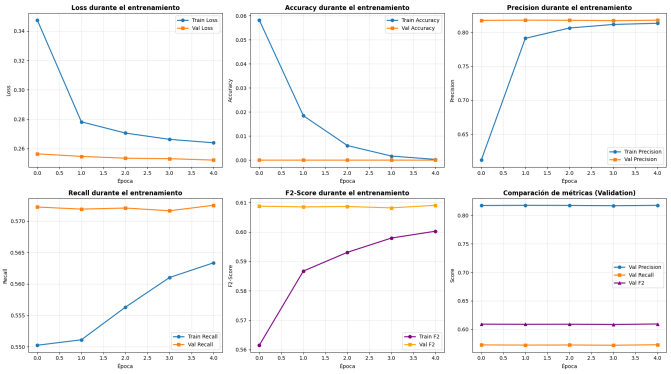

Figura 9. Evolución de las métricas de entrenamiento y validación para el modelo con *transfer learning*.

Las métricas clave de validación muestran comportamientos diferenciados. El *recall* permanece bajo ($\approx$ 0.572–0.573) y prácticamente plano durante todo el entrenamiento, lo que revela que el modelo continúa dejando una proporción considerable de etiquetas verdaderas sin detectar. El *F2-score*, más sensible al *recall*, muestra una ligera tendencia a la baja ($\approx$ 0.6091), lo cual indica que el modelo no mejora su capacidad de detección y, de hecho, presenta un leve deterioro en su balance de generalización.

El análisis detallado por clase, resumido en la Tabla II, muestra un comportamiento extremadamente polarizado. Las clases *primary* y *clear* alcanzan un rendimiento sobresaliente: ambas presentan un *recall* perfecto (1.0) y valores muy altos de *F2-score*, especialmente *primary*, que llega a 0.98. Esto evidencia que las características preentrenadas permiten cap-

turar patrones dominantes de estas clases frecuentes de manera muy efectiva.

Sin embargo, para las quince clases restantes, el desempeño es crítico: la precisión, el *recall* y el *F2-score* son exactamente cero. Esto indica que el modelo no predice ninguna instancia positiva de estas categorías, incluso para clases de presencia media como *agriculture*, *cultivation*, *road* o *water*. En otras palabras, el modelo colapsa hacia una solución que reconoce únicamente dos de las diecisiete clases presentes en el conjunto de datos, ignorando completamente el resto.

Este fenómeno puede atribuirse a varios factores: el fuerte desbalance de clases, un umbral de decisión fijo que favorece clases dominantes, la posible desadaptación entre las características preentrenadas (por el enfoque que tienen las imagenes de ImageNet y las satelitales de nuestro problema) y la estructura multietiqueta del problema, o un entrenamiento insuficiente para ajustar las capas superiores. En conjunto, estos resultados muestran que la incorporación de *transfer learning* en su configuración actual no resuelve las limitaciones del modelo base; de hecho, empeora al reducir drásticamente la diversidad de clases que el modelo es capaz de identificar.

Clase	Precision	Recall	F2
primary	0.927	1.000	0.985
clear	0.708	1.000	0.924
agriculture	0.000	0.000	0.000
blooming	0.000	0.000	0.000
artisinal_mine	0.000	0.000	0.000
blow_down	0.000	0.000	0.000
cloudy	0.000	0.000	0.000
conventional_mine	0.000	0.000	0.000
bare_ground	0.000	0.000	0.000
cultivation	0.000	0.000	0.000
habitation	0.000	0.000	0.000
haze	0.000	0.000	0.000
partly_cloudy	0.000	0.000	0.000
road	0.000	0.000	0.000
selective_logging	0.000	0.000	0.000
slash_burn	0.000	0.000	0.000
water	0.000	0.000	0.000

Cuadro II
RESULTADOS POR CLASE DEL MODELO CON *transfer learning*.

IV-D. Desempeño del modelo con Pytorch

La Figura 10 muestra la evolución de las métricas globales de entrenamiento y validación, mientras que la Tabla III resume el desempeño detallado por clase. En conjunto, estos resultados revelan un comportamiento más equilibrado del modelo respecto a versiones previas, con mejoras visibles en varias categorías de frecuencia media y un mantenimiento sólido en las clases dominantes.

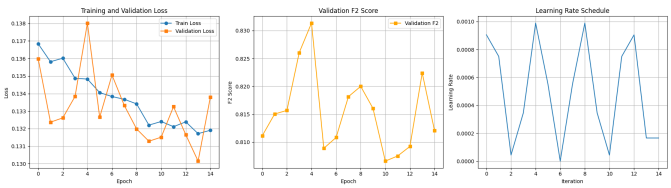

Figura 10. Evolución de las métricas de entrenamiento y validación del modelo actualizado.

Las métricas globales alcanzan un *F2-score micro* promedio de 0.8314, lo que indica un rendimiento mas robusto cuando se ponderan las clases según su frecuencia. Sin embargo, el valor *macro* (0.4783) evidencia una brecha considerable entre clases comunes y raras, patrón habitual en conjuntos de datos con fuerte desbalance.

En las clases más frecuentes *primary*, *clear*, *cloudy* y *partly_cloudy* el modelo mantiene un desempeño sobresaliente. *Primary* y *clear* presentan valores de F2 superiores a 0.95, reflejando una detección estable y confiable. Aunque *partly_cloudy* disminuye ligeramente con respecto a resultados previos, su F2 continúa siendo alto (0.7189), lo que sugiere una buena capacidad de diferenciación en escenarios atmosféricos.

Las clases de frecuencia media muestran mejoras notables. Categorías como *agriculture*, *road*, *haze*, *habitation* y *cultivation* incrementan tanto su *recall* como su F2-score en comparación con la arquitectura base. Un ejemplo destacado es *agriculture*, cuyo F2 asciende a 0.7851, superando significativamente los valores previos. Asimismo, *artisinal_mine* aumenta su F2 desde aproximadamente 0.39 hasta 0.63, indicando una mejor captación de patrones visualmente complejos.

Pese a estos avances, el modelo continúa enfrentando dificultades en clases minoritarias como *blooming*, *blow_down*, *slash_burn* y *selective_logging*, cuyos F2 permanecen cercanos a cero, pero se cree que es por las pocas muestras que tiene originalmente el dataset. Si bien algunas presentan mejoras marginales como *blooming* (0.0194) o *selective_logging* (0.0239) estos valores no representan una capacidad predictiva real. La escasez de ejemplo continúa siendo un factor limitante.

En resumen, el modelo avanzado presenta mejoras sustanciales en clases de frecuencia media, conserva un rendimiento sobresaliente en las categorías principales y continúa enfrentando los desafíos característicos de las etiquetas menos frecuentes.

Clase	Precision	Recall	F2
agriculture	0.7369	0.7981	0.7851
artisinal_mine	0.8393	0.6026	0.6386
bare_ground	0.3810	0.0440	0.0534
blooming	0.5000	0.0156	0.0194
blow_down	0.0000	0.0000	0.0000
clear	0.8944	0.9766	0.9590
cloudy	0.7700	0.7430	0.7482
conventional_mine	0.6923	0.4091	0.4455
cultivation	0.4205	0.4686	0.4581
habitation	0.6981	0.4482	0.4827
haze	0.7282	0.4965	0.5302
partly_cloudy	0.9113	0.6829	0.7189
primary	0.9768	0.9821	0.9810
road	0.7801	0.6758	0.6944
selective_logging	1.0000	0.0192	0.0239
slash_burn	0.0000	0.0000	0.0000
water	0.7133	0.5686	0.5927

Cuadro III
RESULTADOS POR CLASE DEL MODELO PYTORCH.

REFERENCIAS

[1] N. Rom, "Planets dataset," *Kaggle*, 2017.
[2] P. Labs, "Planet: Understanding the amazon from space," *Kaggle*, 2017.
[3] A. Rueda-Toicen, "Planet: Understanding the amazon from space – notebook contribution," *Kaggle*, 2025.