UNIVERSIDAD DE SANTIAGO DE COMPOSTELA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA

Segmentación de imágenes de sensado remoto usando algoritmos de aprendizaje profundo en Pytorch

Autor:
Jesús Campos Manjón

Tutores:
Francisco Argüello Pedreira
Dora Blanco Heras

Grao en Enxeñaría Informática

Julio 2024

Trabajo de Fin de Grado presentado en la Escuela Técnica Superior de Ingeniería de la Universidad de Santiago de Compostela para la obtención del Grado en Ingeniería Informática

D. Francisco Santiago Argüello Pedreira, Profesor del Departamento de Electrónica y Computación de la Universidad de Santiago de Compostela, y **D.ª Dora Blanco Heras**, Profesora del Departamento de Electrónica e Computación de la Universidad de Santiago de Compostela,

INFORMAN:

Que la presente memoria, titulada Segmentación de imágenes de sensado remoto usando algoritmos de aprendizaje profundo en Pytorch, presentada por **D. Jesús Campos Manjón** para superar los créditos correspondientes al Trabajo de Fin de Grado de la titulación de Grado en Ingeniería Informática, se realizó bajo nuestra tutoría en el Departamento de Electrónica y Computación de la Universidad de Santiago de Compostela.

Y para que así conste a los efectos oportunos, expiden el presente informe en Santiago de Compostela, a 2 de Julio de 2024:

Titor, Cotitora, Alumno,

Francisco Santiago Argüello Pedreira Dora Blanco Heras Jesús Campos Manjón

Agradecimientos

- A Francisco y Dora, por su ayuda y paciencia a lo largo de todo el trabajo.
- A mis padres, mi hermano y mi pareja, por hacer de mi casa el mejor lugar de estudio y apoyarme en todo momento.

Resumen

La segmentación semántica de imágenes multiespectrales de sensado remoto ha visto un aumento significativo en el uso de algoritmos de aprendizaje profundo en la última década, debido a las altas precisiones que estos métodos pueden alcanzar. Las imágenes multiespectrales son aquellas capturadas a través de múltiples bandas del espectro electromagnético, más allá de las bandas visibles del rojo, verde y azul, permitiendo captar información detallada sobre las propiedades y características de los objetos observados. Este trabajo se centra en adaptar modelos de redes neuronales profundas, originalmente diseñados para imágenes RGB, al dominio de las imágenes multiespectrales, utilizando el entorno de PyTorch. El objetivo principal es desarrollar y evaluar modelos eficientes y robustos para la segmentación semántica, validando su rendimiento con conjuntos de datos de imágenes multiespectrales.

Una de las principales contribuciones es la demostración experimental de que los modelos de aprendizaje profundo diseñados para la segmentación semántica de imágenes RGB pueden adaptarse eficazmente a imágenes multiespectrales, utilizando toda la información espectral y espacial disponible en este tipo de imágenes y logrando resultados de alta calidad. Esto valida la hipótesis planteada inicialmente de que estos algoritmos pueden ser aplicados con éxito en el contexto de imágenes multiespectrales, superando los modelos del estado del arte en términos de precisión y efectividad.

En el desarrollo de este trabajo, se estudian arquitecturas típicas de redes neuronales profundas y se prueban distintos modelos para determinar cuál ofrece un mejor rendimiento en la segmentación semántica de imágenes multiespectrales. Se identifica que, para el conjunto de datos utilizado, después de aproximadamente 60 épocas de entrenamiento, los incrementos en precisión e IoU media se vuelven mínimos, indicando un punto de saturación. Este hallazgo sugiere que entrenar los modelos más allá de este punto no proporciona mejoras significativas en el rendimiento, pero sí incrementa considerablemente el tiempo de computación. Además, se observan signos de sobreentrenamiento en algunos modelos que continúan entrenando más allá de este punto, lo que no mejora el rendimiento y, en algunos casos, incluso lo perjudica.

Otra contribución importante es la comprobación de que las arquitecturas más profundas no necesariamente ofrecen los mejores resultados. Modelos más ligeros, como la ResNet18dilated + C1_deepsup, alcanzan una alta precisión y buen rendimiento con menor tiempo de entrenamiento en comparación con modelos más complejos y profundos como la ResNet101dilated + PPM_deepsup.

Las pruebas finales, realizadas con el modelo que presentó mejores resultados en las pruebas preliminares (ResNet18dilated + C1_deepsup), revelan una notable variabilidad en el rendimiento del modelo entre diferentes imágenes de prueba, lo que sugiere que el tamaño del conjunto de datos de entrenamiento podría ser insuficiente para capturar toda la variabilidad necesaria para un rendimiento consistente y generalizado. Esto abre la posibilidad de futuras mejoras, como aumentar el tamaño y la diversidad del conjunto de datos de entrenamiento para mejorar la robustez y consistencia de los modelos, optimizar los hiperparámetros para descubrir configuraciones más eficientes y efectivas para la segmentación semántica de imágenes multiespectrales.

Palabras clave: segmentación semántica, imágenes multiespectrales, aprendizaje profundo, redes neuronales, PyTorch, sensado remoto.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

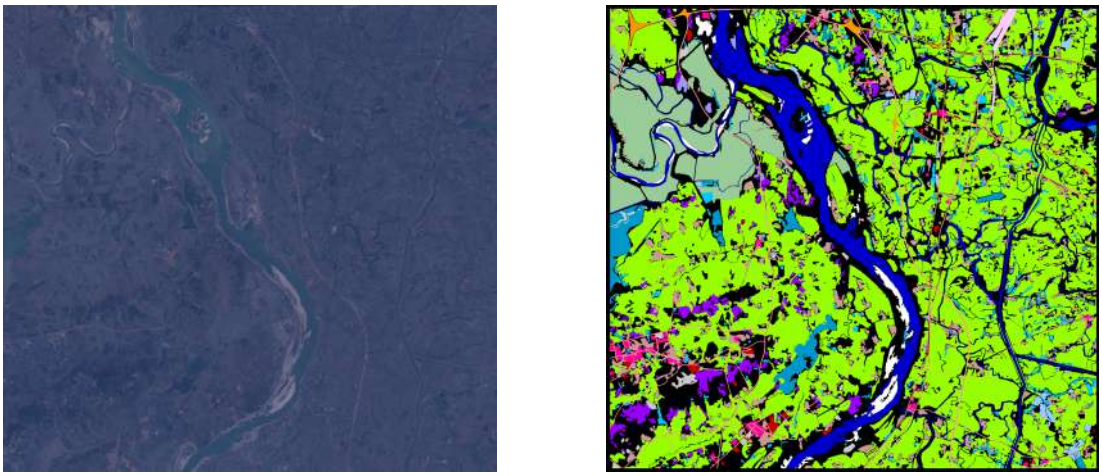
Introdución

El sensado remoto ha revolucionado la forma en que monitoreamos y gestionamos el medio ambiente, permitiendo la adquisición de datos detallados de la superficie terrestre desde plataformas aéreas y espaciales, como satélites y drones. Estos dispositivos utilizan sensores avanzados que capturan la radiación electromagnética reflejada o emitida por los objetos en la superficie terrestre, generando imágenes multiespectrales e hiperespectrales [1]. Este tipo de imágenes captura información a través de múltiples bandas del espectro electromagnético, más allá del rango visible (que incluye las bandas roja, verde y azul). A diferencia de las imágenes en color tradicionales, que se basan en tres bandas (RGB), las imágenes multiespectrales incluyen varias bandas adicionales que pueden abarcar desde el ultravioleta hasta el infrarrojo cercano y medio, lo que las convierte en una potente fuente de información para una amplia gama de aplicaciones. Sin embargo, el análisis efectivo de estas imágenes presenta desafíos significativos debido a la complejidad y el gran volumen de datos involucrados.

La segmentación de imágenes es una técnica que permite identificar y aislar los píxeles que componen objetos específicos dentro de una imagen. Esta técnica es aplicable en imágenes multiespectrales e hiperespectrales, como se puede ver en [2], donde se realiza un estudio sobre la aplicación de este método en imágenes multiespectrales. Entre las técnicas de segmentación, la segmentación semántica [3] destaca por su capacidad de clasificar cada píxel de una imagen en función de su pertenencia a una clase específica. A diferencia de otros métodos de segmentación que simplemente dividen la imagen en regiones homogéneas sin interpretación del contenido, la segmentación semántica asigna etiquetas significativas a cada píxel, permitiendo así identificar y distinguir diferentes objetos y materiales en la imagen.

Por ejemplo, en la imagen a la izquierda (véase Figura 1.1a) se muestra una imagen tomada por el satélite Gaofen-2 (GF-2) en territorio chino. A la derecha

(véase Figura 1.1b) se muestra la misma imagen anotada a nivel de píxel, donde los distintos colores de los píxeles muestran distintos tipos de materiales, como pueden ser agua, edificios, bosque... Con estos píxeles identificados, se podrían fácilmente eliminar todos los píxeles pertenecientes a una clase o modificar el fondo de la imagen. Esta capacidad de asignar significado a cada píxel es crucial para aplicaciones como la detección de cambios en el uso del suelo, la identificación de cultivos en la agricultura y el monitoreo de desastres naturales, entre muchas otras posibilidades.



(a) Imagen tomada por satélite (b) Máscara de la imagen

Figura 1.1: Imágenes del dataset GID [4]. A la izquierda se encuentra una imagen tomada por el satélite Gaofen-2, modificada para representarse en formato RGB. A la derecha se encuentra la máscara de esta imagen, la cual clasifica cada píxel en diferentes categorías según su contenido.

En el ámbito del procesamiento de imágenes, los algoritmos de aprendizaje profundo han demostrado ser particularmente potentes para la segmentación semántica [5]. El aprendizaje profundo, una subárea del aprendizaje automático, utiliza redes neuronales artificiales con múltiples capas para aprender representaciones jerárquicas de datos. Estas redes, conocidas como redes neuronales profundas, son capaces de modelar relaciones complejas en los datos a través de procesos de entrenamiento supervisado y no supervisado. En el contexto de la segmentación semántica de imágenes multiespectrales [2], los algoritmos de aprendizaje profundo pueden aprender características relevantes directamente a partir de los datos de entrada, lo que los hace altamente efectivos para esta tarea.

El entorno de programación Pytorch ha emergido como una de las herramientas más populares para implementar modelos de aprendizaje profundo debido a su flexibilidad y capacidad para manejar grandes volúmenes de datos. Proporciona una plataforma robusta para el desarrollo y la implementación de modelos de

redes neuronales, permitiendo a los investigadores construir y entrenar modelos complejos con relativa facilidad. Sin embargo, muchos modelos existentes están diseñados para trabajar con imágenes RGB estándar y no se adaptan bien a las características específicas de las imágenes capturadas por drones o satélites, que suelen incluir múltiples bandas espectrales.

La hipótesis de este TFG es que los algoritmos de aprendizaje profundo diseñados para imágenes RGB cuando son adaptados para imágenes multiespectrales, pueden producir una segmentación semántica de alta calidad. Para poner a prueba esta hipótesis, se explorarán repositorios de código abierto que implementan redes neuronales profundas en PyTorch, identificando y seleccionando aquellos modelos que puedan ser adaptados a las necesidades específicas de este Trabajo de Fin de Grado. Se propone así desarrollar un modelo de segmentación semántica que identifique y clasifique con precisión los diferentes materiales en las imágenes multiespectrales de sensado remoto, utilizando toda la información espectral y espacial disponible en este tipo de imágenes. Se busca superar los modelos del estado del arte y contribuir a la investigación científica en el campo de la visión por computadora y el procesamiento de imágenes multiespectrales.

Para alcanzar este objetivo, se plantean varios objetivos concretos:

1. Realizar una revisión exhaustiva de la literatura existente sobre segmentación semántica y algoritmos de aprendizaje profundo.
2. Adaptar modelos de segmentación semántica basados en redes neuronales profundas utilizando PyTorch, ajustándolos para trabajar con imágenes multiespectrales.
3. Entrenar los modelos desarrollados utilizando conjuntos de datos de imágenes multiespectrales de cobertura terrestre de calidad contrastada.
4. Validar el rendimiento de los modelos mediante técnicas de evaluación estándar.
5. Seleccionar el mejor modelo y realizar una evaluación final de su rendimiento.

Capítulo 2

Estado del arte

La segmentación semántica de imágenes multiespectrales de sensado remoto ha sido objeto de numerosos estudios debido a su importancia en numerosas aplicaciones. En este capítulo se proporciona una visión cohesionada de las metodologías utilizadas en estos procesos, analizando los distintos métodos de aprendizaje profundo desarrollados para realizar la segmentación semántica de estas imágenes. Para ello nos basaremos en estudios recientes y desarrollos tecnológicos donde se utilizan imágenes del satélite Gaofen-2, en particular, el dataset conocido como *GID* [4], el cual será también utilizado en nuestros experimentos.

A continuación, se presentan y discuten las técnicas de aprendizaje profundo empleadas en la segmentación semántica y la clasificación de imágenes multiespectrales capturadas por satélite en diversos estudios relevantes.

2.1. Redes neuronales convolucionales (CNN)

Las redes neuronales convolucionales (CNN)[6] han demostrado ser extremadamente eficaces en diversas tareas de reconocimiento visual, incluyendo la segmentación y clasificación de imágenes multiespectrales e hiperespectrales. Estas redes han revolucionado el análisis de imágenes al permitir la extracción automática de características, eliminando así la necesidad de diseñar manualmente características específicas para cada tarea. En el contexto de las imágenes multiespectrales, las CNN se encuentra con diversos desafíos debido a la alta correlación entre bandas y la limitada disponibilidad de muestras de entrenamiento etiquetadas.

Una red neuronal convolucional típica está compuesta por capas de convolución y capas de *pooling* espacial. Las capas de convolución son responsables de extraer mapas de características utilizando filtros lineales seguidos de funciones de activación no lineales (por ejemplo, ReLU, sigmoide, tanh). Por otro lado, las capas

de *pooling* agrupan características locales de píxeles adyacentes, mejorando la robustez de la red frente a pequeñas variaciones en la forma de los objetos.

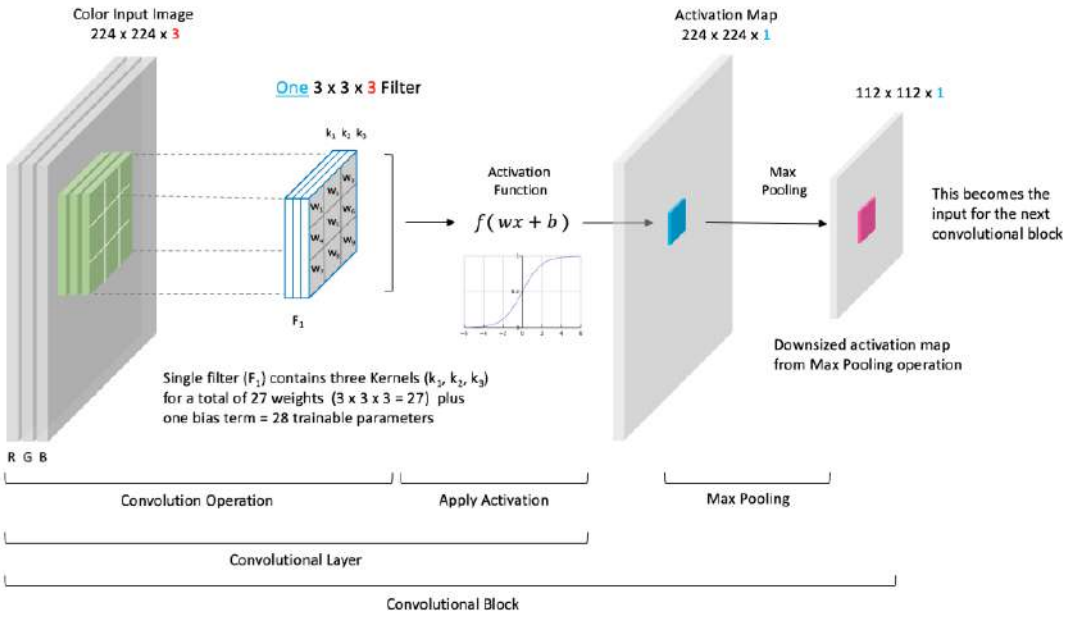


Figura 2.1: Representación visual de un bloque convolucional [7]

Las imágenes multi/hiperespectrales están compuestas por distintas capas o bandas, donde cada una de ellas contiene información relacionada con la radiación electromagnética asociada a una diferente longitud de onda. Dado que cada elemento o material refleja la luz incidente de diferente manera, capturar una imagen multiespectral permite identificar materiales que a simple vista serían indistinguibles. La identificación de elementos en imágenes multiespectrales o hiperespectrales mediante redes neuronales convolucionales es un desafío considerable debido a la alta correlación entre las numerosas bandas espectrales. Estas imágenes contienen una gran cantidad de datos detallados en múltiples bandas, lo que complica el procesamiento y análisis por parte de las CNN. Además, en general, no suele contarse con imágenes con un elevado número de muestras etiquetadas que permitan realizar un buen entrenamiento. Por este motivo se suelen usar diferentes técnicas que permiten realizar un preprocesado de las imágenes multiespectrales que da lugar a un procesado mediante CNNs más eficiente:

- **Reducción de dimensionalidad**: Un enfoque común para manejar la alta dimensionalidad de los datos multiespectrales es la reducción de dimensionalidad. Métodos como el análisis de componentes principales (PCA) [8] y el análisis de componentes independientes (ICA) [9], o directamente elegir las bandas más representativas de las imágenes, transforman los datos origina-

les en un espacio de menor dimensión, reduciendo el impacto del fenómeno de *Hughes* [10], que requiere una gran cantidad de muestras etiquetadas para obtener resultados confiables. Por otro lado, esto puede provocar pérdida de información valiosa, por lo que se podrían perder características útiles para diferenciar distintos materiales en la imagen.

- **Extracción de características espaciales**: Además de la información espectral, la información espacial es crucial para la clasificación de imágenes multiespectrales. Esta información refleja el hecho de que píxeles adjuntos tienen una alta probabilidad de pertenecer a la misma clase. Métodos de morfología matemática y filtros espaciales [11] ayudan a explotar la regularidad espacial de los materiales, mejorando la precisión de la clasificación. Para sacar características espaciales de los objetos correctamente, es necesario utilizar conjuntos de datos de entrenamiento de gran tamaño.
- **Fusión de características espacial-espectral**: La combinación de información espectral y espacial es esencial para mejorar el rendimiento en la clasificación de imágenes multiespectrales [12]. Métodos basados en *wavelets* tridimensionales integran ambas fuentes de información, resultando en un rendimiento competitivo en escenarios con muestras de entrenamiento limitadas.

Aunque estas técnicas son ampliamente utilizadas y han demostrado ser efectivas en la clasificación de imágenes multiespectrales, este Trabajo de Fin de Grado se enfocará en la aplicación de redes neuronales convolucionales sin implementar estas técnicas específicas de preprocesamiento, tratando así de utilizar toda la información espectral y espacial disponible en las imágenes.

2.1.1. Redes residuales (ResNet)

Un tipo de CNN muy utilizado en la segmentación semántica y clasificación de imágenes multiespectrales son las Redes Residuales (ResNet) [13]. Este tipo de redes son una extensión de las CNN que permiten la construcción de modelos mucho más profundos. Se introduce así el concepto de aprendizaje residual, esta técnica simplifica el proceso de aprendizaje y optimización de redes muy profundas. Esto se logra utilizando conexiones de atajo que saltan una o más capas dentro de la red, mejorando la eficiencia del flujo de gradientes y reduciendo la degradación del rendimiento en redes, resolviendo así problemas como la desaparición y explosión de gradientes.

Debido a su alta capacidad para extraer automáticamente características complejas y su robustez frente a la grandes dimensiones y correlación de bandas, estas redes son herramientas poderosas en el análisis de imágenes multiespectrales e hiperespectrales. Las técnicas avanzadas de aprendizaje residual han permitido construir redes más profundas y efectivas, mejorando significativamente el rendi-

miento en diversas tareas de reconocimiento visual. Por ejemplo, en [14] se utiliza una ResNet-50 entrenada por muestras del dataset GID. Esta red está compuesta por 49 capas convolucionales y 16 estructuras de cuello de botella. Argumentan el uso de esta red basándose en la facilidad de su entrenamiento cuando la arquitectura de la red es muy profunda y en la simplificación del problema de la desaparición del gradiente.

2.1.2. Redes totalmente convolucionales (FCN)

Las *Fully Convolutional Networks* (FCN) [15], propuestas por Long, Shelhamer y Darrell en 2015, surgieron como una evolución natural de las CNN para abordar tareas de predicción densa, como la segmentación semántica. A diferencia de las CNN tradicionales, que se utilizan principalmente para tareas de clasificación de imágenes, las FCN están diseñadas para generar predicciones píxel a píxel, lo cual es esencial en tareas donde se requiere una interpretación detallada y precisa de cada parte de la imagen.

Las CNN tradicionales han demostrado ser extremadamente eficaces en tareas de clasificación de imágenes gracias a su capacidad para aprender representaciones jerárquicas de las características visuales. Sin embargo, su estructura incluye capas completamente conectadas que eliminan la información espacial de las imágenes, lo que limita su aplicabilidad en tareas que requieren predicciones espaciales precisas, como la segmentación semántica. Para superar esta limitación, las FCN reestructuran las redes de clasificación eliminando las capas completamente conectadas y reemplazándolas por capas convolucionales que pueden operar sobre entradas de tamaño arbitrario. Este enfoque permite que la red mantenga la resolución espacial de las características a lo largo de toda la arquitectura de la red. En lugar de producir una única predicción para toda la imagen, las FCN generan un mapa de características a partir del cual se pueden obtener predicciones para cada píxel de la imagen original.

Una de las innovaciones clave de las FCN es el uso de capas de *upsampling* o *deconvolution* para aumentar la resolución de las predicciones. Estas capas permiten que la red recupere la información espacial perdida durante las operaciones de *downsampling*, como las capas de *pooling*. Esto se logra mediante el uso de convoluciones transpuestas que amplían el tamaño de las características y permiten realizar predicciones a nivel de píxel con alta resolución. Además, las FCN utilizan arquitecturas que implementan conexiones de salto que combinan la información semántica de las capas profundas con la información espacial detallada de las capas superficiales. Esto mejora la precisión de las predicciones al permitir que la red incorpore tanto la información global como los detalles locales en el proceso de segmentación.

Las FCN son esenciales para la segmentación semántica porque permiten una pre-

dicción precisa a nivel de píxel, manteniendo la estructura espacial de las imágenes y combinando información de diferentes niveles de abstracción. Este enfoque ha demostrado ser altamente efectivo en tareas de segmentación. En [16] se experimenta con este tipo de arquitectura para segmentar semánticamente imágenes de los conjuntos de datos GID y ESAR. Una particularidad que se lleva a cabo en este artículo es la integración de información de borde. Mediante el método *Edge-FCN*, utiliza la detección de bordes como conocimiento previo para corregir los resultados de segmentación generados por una FCN. Este método combina la red FCN y la red HED (*Holistically Nested Edge Detection*). Esta última detecta los bordes para incorporarlos en el proceso de segmentación, mejorando así la precisión en los límites de los objetos.

2.1.3. Arquitecturas codificador-decodificador

En las FCN originales, varias etapas de convoluciones estratificadas y/o agrupación espacial reducen la predicción final de la imagen, perdiendo así información delicada de la estructura de la imagen y causando predicciones inexactas, especialmente en los bordes de los objetos. Para abordar las limitaciones de las FCN, se desarrollaron arquitecturas de codificador-decodificador, como SegNet [17] y U-Net [18], que mejoran la precisión de la segmentación al recuperar detalles finos durante el proceso de *upsampling*. SegNet utiliza índices de *pooling* para realizar un *upsampling* no lineal, mientras que U-Net emplea conexiones de salto que combinan características de diferentes resoluciones para mejorar la segmentación de objetos pequeños y detalles finos.

La arquitectura de codificador-decodificador [19] es considerado el método de aprendizaje profundo más eficiente para la segmentación semántica en distintos campos. Esta arquitectura considera la CNN troncal como un codificador, encargado de codificar una imagen de entrada sin procesar en mapas de características de menor resolución. A continuación, se utiliza un decodificador para recuperar la predicción densa píxel por píxel de los mapas de características de menor resolución. En [19] se utiliza dicha arquitectura con una SegNet, realizando experimentos con el dataset GID y comparando los resultados con otras arquitecturas similares.

Este Trabajo de Fin de Grado hace una contribución significativa al campo de la segmentación semántica de imágenes multiespectrales de sensado remoto. Se aprovecha toda la información espacial y espectral disponible en estas imágenes, y se enfoca en el estudio y la aplicación de técnicas avanzadas y eficientes de aprendizaje profundo desarrolladas recientemente para esta tarea. Se utiliza la arquitectura codificador-decodificador debido a su capacidad demostrada para recuperar detalles finos y mejorar la precisión de la segmentación, tal como se ha evidenciado en estudios previos con modelos como SegNet y U-Net. Esta elección

se justifica por la eficacia de esta arquitectura en combinar características de múltiples resoluciones y en realizar un *upsampling* eficiente, lo que es crucial para mantener la integridad de la información espacial y estructural en las imágenes de alta complejidad propias del sensado remoto.

Además, este estudio ofrece una evaluación comparativa con otros enfoques y herramientas, destacando la importancia de seleccionar métodos adecuados para mejorar la precisión y eficiencia en la segmentación de estas imágenes complejas. Los resultados y metodologías desarrolladas pueden servir como referencia para futuros estudios y aplicaciones en el análisis de imágenes de sensado remoto de cobertura terrestre.

Capítulo 3

Entorno de trabajo

En este capítulo se explica en detalle el entorno de trabajo utilizado durante el desarrollo de este TFG. Se documenta y explica el material utilizado, incluyendo conjuntos de datos, herramientas software para la visualización de imágenes multiespectrales e hiperespectrales y el código base del que se parte la implementación de la segmentación semántica de imágenes multiespectrales de sensado remoto. Por otro lado, al tratarse de un trabajo que conlleva el entrenamiento de múltiples redes neuronales, se necesitan unos requisitos computacionales de procesador, GPU y memoria mínimos, por lo que es de especial importancia detallar el equipo en el que se realizó el entrenamiento de dichas redes, así como servidores u otras alternativas utilizadas.

3.1. Conjuntos de datos

En este apartado se introducen y detallan los distintos conjuntos de datos utilizados durante la elaboración del trabajo.

3.1.1. Conjunto de datos del ADE20K

Para el desarrollo de este TFG se parte de un código diseñado para utilizarse con el dataset ADE20K [20]. Por esta razón, las primeras pruebas con el código se realizarán utilizando este conjunto de datos, con el fin de asegurar que el código funciona correctamente. ADE20K es un dataset ampliamente anotado utilizado para la segmentación semántica de escenas. Forma parte del MIT Scene Parsing Benchmark y contiene más de 20,000 imágenes centradas en diversas escenas, cada una anotada con objetos y partes de objetos. Comparado con otros conjuntos de datos, ADE20K destaca por la diversidad de escenas y la riqueza de sus anotaciones, incluyendo 150 clases y una segmentación a nivel de partes de objetos.

Esta estructura detallada y extensiva permite validar el correcto funcionamiento del código y ajustar las metodologías antes de aplicarlas al dataset específico utilizado en este TFG.

En la Figura 3.1 se puede observar un ejemplo de una escena que pertenece a este conjunto de datos, así como su correspondiente anotación y la imagen resultante de realizar la inferencia con uno de los modelos.

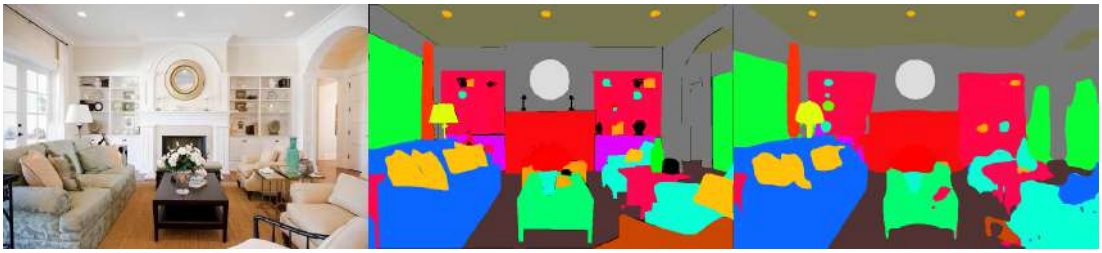


Figura 3.1: Imagen del ADE20K ofrecida por [21]. De izquierda a derecha se presenta: la imagen de la escena original, la imagen anotada y la imagen resultante de la inferencia con uno de los modelos implementados

3.1.2. Conjunto de datos GID

GID (Gaofen Image Dataset) [4] es un conjunto de datos de imágenes satelitales multiespectrales de alta resolución (4 metros por píxel) capturadas por el satélite Gaofen-2 (GF-2). Este dataset es altamente valorado por su amplia cobertura geográfica, distribución diversificada y alta resolución espacial. Contiene 150 imágenes GF-2 anotadas a nivel de píxel (máscaras), donde cada píxel pertenece a una de 5 posibles clases.

El conjunto de datos Five Billion Pixels [22] es una extensión del GID, que no solo incluye más imágenes, sino que también proporciona diferentes formatos de imagen y máscaras anotadas con más clases. Este dataset extiende el GID al incluir imágenes satelitales originales sin anotar y anotaciones a nivel de píxel de 24 clases para las mismas imágenes. De aquí en adelante, nos referiremos a ambos conjuntos de datos como GID.

Las imágenes satelitales están compuestas por 4 bandas (R, G, B y NIR) y están disponibles en formatos de 8 y 16 bits. El uso de imágenes multiespectrales de 16 bits ofrece ventajas significativas, como un mayor rango dinámico que permite una captura más precisa de las variaciones de intensidad, beneficiando el entrenamiento de modelos de predicción.

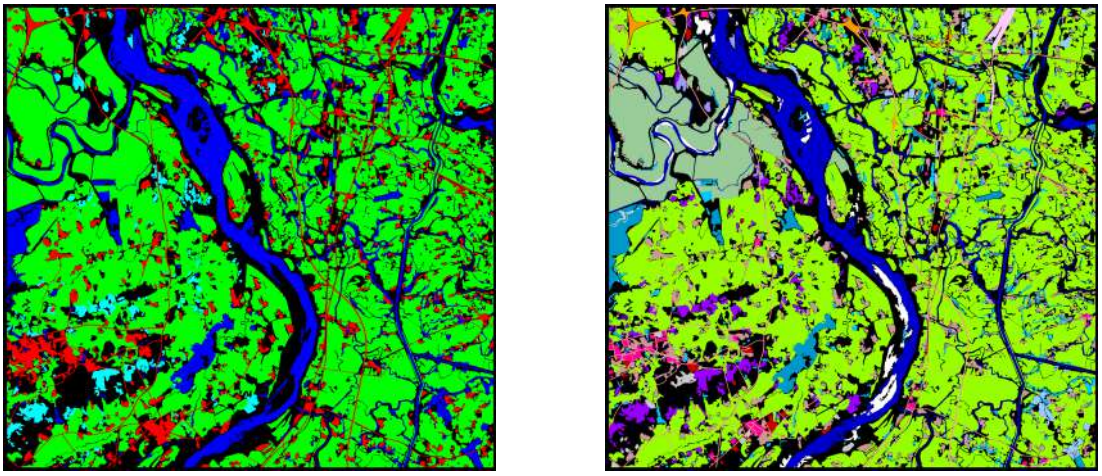
Las imágenes anotadas están disponibles en dos formatos:

- Anotaciones en color: Máscaras de etiquetas coloreadas en formato RGB, donde diferentes colores representan distintas clases. Este tipo de anotación

es muy útil para el análisis visual, permitiendo distinguir fácilmente los objetos y las clases a las que pertenecen.

- Anotaciones por índices: Máscaras de etiquetas de un solo canal, con valores de píxel que representan índices de clase. Estas imágenes son menos informativas visualmente, pero más eficientes para el almacenamiento y procesamiento, lo que las hace preferibles para el entrenamiento de modelos.

GID proporciona una rica variedad de clases, llegando a cubrir hasta 24 categorías diferentes. En este TFG se entrenará y probarán los modelos utilizando las imágenes de 16 bits y las máscaras de 24 clases para explorar el potencial de los modelos de segmentación semántica en la identificación y clasificación de materiales. Las Figuras 3.2a y 3.2b muestran claramente la diferencia en el número de clases entre el GID y su versión extendida. Este aumento de clases puede influir en la complejidad de la tarea de clasificación, haciendo esta tarea mucho más difícil. La correspondencia de colores (RGB) e índices con sus respectivas clases se definen en el Cuadro 3.1.



(a) Máscara de 5 clases (b) Máscara de 24 clases

Figura 3.2: Máscaras extraídas del dataset GID. A la izquierda se encuentra una imagen anotada (máscara) capturada por el Gaofen-2 en la cual únicamente se utilizan 5 clases, perteneciente al GID. A la derecha se encuentra la misma imagen pero anotada con 24 clases distintas, sacada de la versión extendida del GID.

Cuadro 3.1: Correspondencia de colores e índices con sus respectivas clases en el GID

Color	Índice	Clase
	1	industrial area
	2	paddy field
	3	irrigated field
	4	dry cropland
	5	garden land
	6	arbor forest
	7	shrub forest
	8	park
	9	natural meadow
	10	artificial meadow
	11	river
	12	urban residential
	13	lake
	14	pond
	15	fish pond
	16	snow
	17	bareland
	18	rural residential
	19	stadium
	20	square
	21	road
	22	overpass
	23	railway station
	24	airport
	0	unlabeled

3.2. Herramientas de visualización

Htool es una herramienta desarrollada por el grupo de investigación de los tutores del trabajo. Esta herramienta permite la visualización y manipulación de imágenes, aunque no está disponible para uso público. A diferencia de la mayoría de aplicaciones de visualización y edición de imágenes, este software está enfocado en la visualización de imágenes multiespectrales e hiperespectrales, permitiéndonos visualizar imágenes de más de 3 bandas, manipular el orden de estas o incluso aplicar modelos de segmentación y clasificación. Aunque no se explota todo el potencial de esta herramienta, se hace uso de ella durante el desarrollo de

gran parte del trabajo. Como las imágenes multiespectrales de los datasets GID y Five Billion Pixels no se pueden visualizar con cualquier programa, esta herramienta nos permitirá comprobar visualmente el estado de las imágenes (véase Figura 3.3), asegurándonos así de que no se está cometiendo ningún error en la lectura, transformaciones ni guardado de estas.

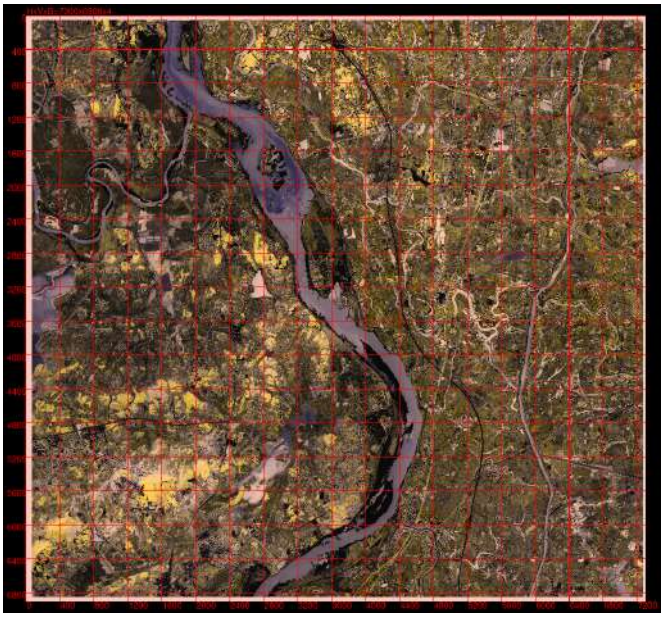


Figura 3.3: Imagen de 4 bandas extraída del GID, visualizada con la herramienta htool

3.3. Repositorio base para el proyecto

El método de segmentación basado en aprendizaje profundo del que se parte en este TFG para su adaptación para imágenes multiespectrales es el que se encuentra disponible en el repositorio del *MIT CSAIL Computer Vision*, ***semantic-segmentation-pytorch*** [21]. Este repositorio es conocido por su flexibilidad y completa documentación, lo que lo hace ideal para adaptarlo a nuestras necesidades específicas. A continuación, se detalla la estructura básica del código:

- /config: Contiene los archivos de configuración de los modelos, permitiendo personalizar el dataset, elegir el modelo y modificar parámetros de entrenamiento como el optimizador o el ratio de aprendizaje.
- /data: Incluye los datos de entrenamiento y validación, además de un archivo de coloración que mapea cada clase a su color correspondiente.
- /mit_semseg: Este directorio contiene varios scripts de Python que son fun-

damentales para definir los datasets a utilizar, las transformaciones de los datos y la inicialización de los codificadores y decodificadores. En cuanto a las técnicas de aprendizaje profundo empleadas, se utilizan Redes Neuronales Convolucionales (CNNs) con arquitecturas de codificador-decodificador. Los codificadores implementados incluyen modelos como ResNet y MobileNet, mientras que los decodificadores implementados comprenden PSPNet y UPerNet entre otros. La implementación de estos modelos se detalla en la sección 4.3.

- eval.py y eval_multipro.py: Estos scripts evalúan los modelos, generando estadísticas como la precisión o la intersección sobre unión para cada clase. Eval_multipro.py permite evaluaciones más rápidas usando múltiples GPUs. Además, permite generar visualizaciones de las predicciones del modelo evaluado.

- test.py: Este script prueba un modelo con las imágenes que deseemos, descargando las inferencias producidas por el modelo sin proporcionar ningún tipo de métricas de evaluación.

- train.py: Automatiza el proceso de entrenamiento de la CNN, desde la carga de datos hasta la gestión del entrenamiento y el guardado de puntos de control.

El código del repositorio se desarrolló bajo las siguientes configuraciones:

- Hardware: $\geq$4 GPUs para entrenamiento, $\geq$1 GPU para pruebas.

- Software: Ubuntu 16.04.3 LTS [23], CUDA [24] $\geq$8.0, Python [25] $\geq$3.5, PyTorch [26] $\geq$0.4.0.

- Dependencias: numpy [27], scipy [28], opencv [29], yacs [30], tqdm [31].

Un aspecto clave de este código es la implementación de la Normalización por Lotes Sincronizada en PyTorch, que mejora la estabilidad y precisión durante el entrenamiento en múltiples GPUs, aunque con un pequeño costo en velocidad. Para mantener la proporción de las imágenes durante el entrenamiento, el módulo *DataParallel* [33] ha sido reimplementado, por los autores del repositorio, para poder distribuir datos a múltiples GPUs, permitiendo que cada una procese imágenes de diferentes tamaños. Esto ajusta el tamaño del batch al número de GPUs y distribuye los datos de forma eficiente, asegurando un uso óptimo de las capacidades de PyTorch.

3.4. Equipamiento informático

Entrenar cualquier tipo de red neuronal suele ser una tarea con altos requerimientos computacionales, por lo que no cualquier computadora puede llevar a cabo esta tarea. En esta sección se explican los equipos utilizados para el entrenamiento de los modelos durante el desarrollo de este TFG, explicando los motivos de su uso, y detallando su hardware, software y capacidad computacional.

3.4.1. Computadora personal

El primer equipo en el que se empezará a trabajar y realizar pruebas con el código anterior descrito es una computadora de uso personal, en concreto un HP Victus Laptop. Este dispositivo tiene una única tarjeta gráfica NVIDIA GeForce RTX 3060 [34] de 12GB de memoria y un procesador Intel(R) Core(TM) i7-12700H de doceava generación [35]. En cuánto software, el sistema operativo utilizado es Ubuntu 22.04.3 LTS [23], y la versión de CUDA es la 12.2 [24]. Se usa Python 3.10.12 [25], torch 2.2.0+cu118, torchaudio 2.2.0+cu118, torchvision 0.17.0+cu118 [26]. Por otro lado, las dependencias son las siguientes: numpy 1.26.3 [27], scipy 1.12.0 [28], opencv-python 4.9.0.80 [29], yacs 0.1.8 [30], tqdm 4.66.1 [31], GDAL 3.4.1 [36]

Al ser una computadora de uso personal, es la opción más simple y cómoda. Sin embargo, las limitadas prestaciones del hardware restringen su uso a probar la funcionalidad del código o a ejecutar procesos que no requirieran un alto poder computacional, como la evaluación del modelo o la inferencia de algunas imágenes.

Un obstáculo que se presenta al realizar el entrenamiento de los modelos en los equipos que tienen una única GPU, es que el código base esta diseñado para utilizar el módulo *DataParallel*, el cuál necesita más de un GPU para su correcto funcionamiento. Sin embargo, con las debidas modificaciones (explicadas en la sección 4.1) se puede sortear este obstáculo sin mayor dificultad.

Otro problema muy frecuente que surge al utilizar este equipo en concreto, es que durante el entrenamiento la GPU se quede sin memoria, dando lugar al error *torch.cuda.OutOfMemoryError: CUDA out of memory*, debido al elevado espacio que ocupan los modelos en la GPU. Ademaś, la velocidad de entrenamiento con este equipo es especialmente baja. Surge así la necesidad de explorar otras alternativas.

3.4.2. HPC cluster ctcomp3

Debido a las limitaciones de la computadora personal, se opta por utilizar el High Performance Computing (HPC) cluster ctcomp3 [37] del CiTIUS. Este clúster tiene un poder computacional mucho mayor que el del equipo anterior.

Con esta alternativa, no se da ya el problema de falta de memoria en la GPU. Sin embargo, sigue sin ser posible realizar el entrenamiento usando más de una GPU. Las políticas de Quality of Service (QoS) imponen límites en varios aspectos del uso de recursos, tales como el número de GPUs, CPUs, memoria, o el tiempo de ejecución permitido para un trabajo, no permitiendo utilizar más de una GPU. En este equipo se hace uso de una GPU Tesla V100S-PCIE-32GB [38] y un procesador Intel(R) Xeon(R) Gold 5220R CPU @ 2.20GHz [39]. En cuánto software, el sistema operativo utilizado es AlmaLinux release 8.4 (Electric Cheetah) [32], y la versrión de CUDA es la 12.5 [24]. Se usa Python 3.8.18 [25], torch 2.0.1, torchaudio 2.0.2 y torchvision 0.15.2 [26]. Por otro lado, las dependencias son las siguientes: numpy 1.24.3 [27], scipy 1.10.1 [28], opencv-python 4.7.0 [29], yacs 0.1.8 [30], tqdm 4.66.2 [31], GDAL 3.6.2 [36]

A diferencia con el equipo de uso personal, la configuración de un entorno virtual donde instalar todos los requerimientos para hacer funcionar nuestro software es tediosa. Además, para ejecutar cualquier proceso hay tiempos de espera que pueden superar los días, haciendo que el uso de este equipo sea mucho más incómodo y complejo.

3.4.3. Finisterrae III

Por último, en un intento de usar el código original con los requerimientos hardware indicados por los autores del repositorio, se exploró la alternativa de usar un nodo del supercomputador Finisterrae III [40] del Cesga.

En este caso, sí se permite el uso de más múltiples GPUs para llevar a cabo nuestros entrenamientos, pero la presencia de colas de trabajo aún más largas y limitaciones del espacio disponible para el usuario acaban haciendo inviable el uso de esta alternativa para el entrenamiento con el dataset GID. Es por ello que no se especifica el entorno software ni hardware utilizado.

3.5. Métodos matemáticos/estadísticos utilizados

En esta sección se describen las métricas y funciones utilizadas para evaluar el modelo. Específicamente, se abordarán las métricas de precisión e Intersección sobre Unión (IoU), así como la función de pérdida empleada durante el entrenamiento del modelo. Además, se describirá el optimizador utilizado para ajustar los parámetros del modelo durante el proceso de entrenamiento.

3.5.1. Métricas de evaluación

- **Precisión**: En el contexto de la segmentación semántica, la precisión es una medida de la exactitud de las predicciones del modelo. Se define como la proporción de píxeles correctamente clasificados en una clase específica (verdaderos positivos) sobre el total de píxeles que el modelo ha predicho como pertenecientes a esa clase (la suma de verdaderos positivos y falsos positivos). Es una métrica crucial ya que indica qué tan bien el modelo está identificando correctamente las clases positivas y evita clasificaciones incorrectas.

$$\text{Precisión} = \frac{\text{Píxeles Verdaderos Positivos}}{\text{Píxeles Verdaderos Positivos} + \text{Píxeles Falsos Positivos}} \quad (3.1)$$

- **(IoU)**: En el contexto de la segmentación semántica, la métrica IoU, o Intersección sobre Unión, evalúa la precisión de las predicciones del modelo comparando la superposición entre las áreas predichas y las áreas reales para cada clase. Es una métrica crucial que proporciona una medida de la precisión espacial del modelo al cuantificar el grado de coincidencia entre las predicciones y las etiquetas verdaderas.

  En términos matemáticos, IoU se define como:

$$\text{IoU} = \frac{\text{Área de Superposición}}{\text{Área de Unión}} \quad (3.2)$$

  Donde:

  - Área de Superposición: son los píxeles correctamente predichos como pertenecientes a la clase en cuestión (verdaderos positivos).
  - Área de Unión: es la suma de los píxeles que el modelo predijo como pertenecientes a la clase (verdaderos positivos y falsos positivos) y los píxeles que realmente pertenecen a la clase (verdaderos positivos y falsos negativos).

3.5.2. Función de pérdida

La función de pérdida cuantifica la discrepancia entre las predicciones del modelo y los valores verdaderos, asignando un valor numérico a cada par de predicciones y valores reales.

En este TFG se utiliza la función de pérdida de probabilidad logarítmica negativa (Negative Log-Likelihood Loss, NLLLoss) de PyTorch, ideal para tareas de

clasificación y especialmente útil en conjuntos de datos desequilibrados, como es el caso de nuestro dataset. Esta función penaliza fuertemente las predicciones incorrectas, ajustando el modelo para asignar mayores probabilidades a las clases correctas, mejorando su capacidad de clasificación y segmentación semántica en imágenes multiespectrales.

La NLLLoss se define formalmente como:

$$\ell(x, y) = L = \{l_1, \ldots, l_N\}^\top, \quad l_n = -w_{y_n} x_{n,y_n}, \quad w_c = \text{weight}[c] \cdot 1\{c \neq \text{ignore\_index}\}, \tag{3.3}$$

donde x es el input, y es el *target*, w es el peso, y N es el tamaño del *batch*. El input debe contener log-probabilidades de cada clase y ser un tensor de tamaño $(minibatch, C)$ o $(minibatch, C, d_1, d_2, \ldots, d_K)$ con $K \geq 1$ para dimensiones superiores. El *target* debe ser un índice de clase en el rango $[0, C-1]$, donde C es el número de clases. Con la opción de *ignore_index* podemos excluir ciertas clases, en nuestro caso excluimos la clase sin etiquetar 0.

3.5.3. Optimizador

Un optimizador ajusta iterativamente los pesos del modelo para minimizar una función de pérdida, mejorando así su rendimiento. Los optimizadores son esenciales en el entrenamiento de modelos de aprendizaje automático, ya que determinan cómo se modifican los pesos en respuesta al error calculado en cada iteración.

El SGD (*Stochastic Gradient Descent*) es uno de los optimizadores más comunes en el aprendizaje automático. En este TFG se entrenan los modelos usando la implementación de PyTorch del este método [41], proporcionando un marco flexible y eficiente para el entrenamiento de modelos de aprendizaje profundo. La fórmula de actualización de los parámetros se define como:

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

donde θ son los parámetros del modelo, η es la tasa de aprendizaje, y $\nabla_\theta J(\theta)$ es el gradiente de la función de pérdida J con respecto a θ.

El optimizador SGD también puede incluir *momentum*, que acelera la convergencia al considerar los gradientes de iteraciones previas, y *weight decay*, que introduce regularización $L2$ para evitar el sobreajuste. El *momentum* suaviza las actualizaciones y ayuda a evitar oscilaciones. La regularización $L2$ penaliza los grandes valores de los parámetros, fomentando modelos más simples y reduciendo el riesgo de sobreajuste.

Capítulo 4

Metodología

En este capítulo se describen los procedimientos y técnicas diseñados e implementados, así como su funcionamiento y la forma en que contribuyen a resolver el problema inicial planteado. Se justifican las decisiones de diseño tomadas, explicando las alternativas consideradas y basándose en la bibliografía existente o en las características específicas del problema.

Primero, se abordan todos los pasos previos a la fase de entrenamiento, incluyendo las modificaciones realizadas tanto en el código base como en el dataset. Estas modificaciones son esenciales para adaptar el proyecto a los requerimientos específicos y optimizar el rendimiento de los modelos.

El desarrollo de scripts específicos también es detallado, destacando cómo estos scripts automatizan procesos clave como la preparación de los datos y la configuración del entorno de entrenamiento. Además, se explican las técnicas de aprendizaje profundo implementadas, detallando el funcionamiento de las arquitecturas y modelos utilizados para la segmentación semántica de imágenes de sensado remoto.

Estas etapas preliminares son cruciales para asegurar que la red convolucional esté bien adaptada a las particularidades de las imágenes del GID y los objetivos del proyecto. La experimentación inicial con el código y las modificaciones realizadas permiten una mejor comprensión de las necesidades específicas del modelo y cómo ajustarlo para obtener resultados óptimos. Los scripts desarrollados facilitan la repetibilidad y la automatización de tareas tediosas, permitiendo un enfoque más eficiente y efectivo en el entrenamiento del modelo.

Cada decisión de diseño será justificada a través de una comparación de alternativas, apoyándose en estudios previos y en la bibliografía relevante. Esto permitirá entender por qué se eligieron ciertas estrategias sobre otras y cómo estas decisiones están alineadas con las mejores prácticas y los avances actuales en el campo.

4.1. Entrenamiento con una GPU

El entrenamiento de CNNs para la segmentación semántica de imágenes de sensado remoto, es una tarea con altos requerimientos computacionales debido a la complejidad y el gran tamaño de los datos involucrados. Las imágenes de sensado remoto suelen ser de alta resolución, lo que incrementa la demanda de poder de procesamiento y memoria para manejar tanto los datos como los cálculos necesarios durante el entrenamiento de la red.

Las GPUs están diseñadas para manejar miles de operaciones en paralelo, lo que permite procesar grandes volúmenes de datos simultáneamente, siendo ideales para tareas de entrenamiento de redes neuronales que requieren cálculos intensivos y paralelizables. Este paralelismo masivo resulta en un rendimiento superior, ya que las GPUs pueden realizar operaciones matemáticas complejas mucho más rápido que las CPUs, reduciendo significativamente los tiempos de entrenamiento para modelos de aprendizaje profundo.

Además, las GPUs están optimizadas para operaciones de álgebra lineal y matrices, fundamentales en la mayoría de los algoritmos de aprendizaje profundo y procesamiento de imágenes. Aunque las GPUs consumen más energía que las CPUs, tienen un tiempo de procesamiento reducido, lo que puede mejorar la eficiencia energética en tareas prolongadas y complejas. Por estos motivos, las GPUs suelen ser la opción preferida para tareas de aprendizaje automático y procesamiento de imágenes.

Como se puede ver en la sección 3.3, el código base está diseñado con la premisa de realizar el entrenamiento de los modelos usando 4 GPUs, un requerimiento hardware que un principio no está disponible. En esta sección se explica la adaptación del código ofrecido por el *MIT CSAIL Computer Vision*, en el repositorio *semantic-segmentation-pytorch*, para poder realizar el entrenamiento de los modelos con una única GPU.

Para realizar esta adaptación es necesario modificar la forma en la que el código gestiona y maneja la carga de lotes de datos. El código original utiliza una reimplementación de *DataParallel* de PyTorch para distribuir los datos y operaciones del modelo entre múltiples GPUs. Para comprender correctamente el funcionamiento de este módulo, es fundamental definir algunos conceptos:

- **Lote**: Un lote es un conjunto de muestras que se procesan juntas en una única iteración del entrenamiento de un modelo. Su tamaño afecta la eficiencia computacional, la estabilidad del entrenamiento y la capacidad del modelo para generalizar datos no vistos.
- ***Dataloader***: Un *DataLoader* en PyTorch es una herramienta crucial para la gestión de datos durante el entrenamiento y la evaluación de modelos.

Facilita la carga de datos, dividiéndolos en lotes que pueden ser procesados eficientemente por el modelo. Además, permite mezclar los datos antes de crear los lotes, lo cual es crucial para evitar el sobreajuste.

El módulo *DataParallel* ha sido reimplementado por los autores del repositorio para mejorar la eficiencia del entrenamiento con múltiples GPUs, especialmente en tareas de segmentación semántica donde es crucial mantener la relación de aspecto de las imágenes. Esta implementación incluye una normalización sincronizada [21], calculando la media y la desviación estándar en todos los dispositivos durante el entrenamiento, asegurando coherencia en redes distribuidas y mejorando la estabilidad del entrenamiento, aunque con un pequeño costo en velocidad. Además, el *DataParallel* modificado distribuye los datos a múltiples GPUs utilizando diccionarios de Python, permitiendo que cada GPU procese imágenes de diferentes tamaños.

Independientemente de usar o no *DataParallel*, el número de imágenes que procesa cada GPU es el mismo, ya que este es definido por el tamaño del lote por GPU, el cual representa el número de imágenes que procesa una GPU a la vez. Este tamaño de lote por GPU se puede personalizar en los archivos de configuración.

El *DataLoader* también se ha ajustado para que su tamaño de lote sea igual al número de GPUs disponibles, garantizando que cada GPU reciba un sub-lote diferente. De esta forma, si se usan 4 GPUs, el lote generado por el *DataLoader* estará compuesto por 4 sub-lotes que serán enviados cada uno a una GPU diferente. Cada sub-lote es procesado en paralelo entre las GPUs asignadas, lo que permite una mayor eficiencia.

Después de procesar cada sublote, *DataParallel* recopila los gradientes de todas las GPUs y los promedia para actualizar los parámetros del modelo, asegurando que el modelo se entrene correctamente en un entorno multi-GPU, manteniendo la coherencia y sincronización de los gradientes y las actualizaciones de parámetros.

El problema de la implementación original es que cuando se entrena con una sola GPU, el módulo *DataParallel* no se utiliza. Esto hace que los datos permanezcan en la CPU, mientras que el modelo espera que los datos estén en la GPU, provocando errores en la primera capa de convolución. Simplemente moviendo los datos a la GPU, se asegura que los datos estén en el dispositivo correcto, permitiendo que la red neuronal funcione correctamente durante el entrenamiento.

A pesar de hacer esta adaptación para el funcionamiento del código con una única GPU, se probó también la versión original del código, comprobando de esta forma que ambos códigos presentasen la misma calidad en cuanto resultados. Se entrena así uno de los modelos ya implementados en el código usando el dataset ADE20K. Se entrena, por un lado, desde el Finisterrae III, el superordenador del Centro de Supercomputación de Galicia, (CESGA) haciendo uso de 4 GPUs y con el código original, y por otro lado; desde el (HPC) cluster ctcomp3 del CiTIUS usando un

única GPU y el código adaptado. No se puede realizar el entrenamiento desde la computadora personal, ya que no dispone de una GPU con la suficiente memoria para realizar esta tarea.

En cuanto a calidad de resultados, no se observa ninguna diferencia notable entre el entrenamiento con un equipo u otro. Cabe destacar que las colas de espera en el Finisterrae III suelen ser considerablemente más largas que en el cluster del CiTIUS, por los que los tiempos requeridos para entrenar y evaluar los modelos son difíciles de predecir.

4.2. Entrenamiento con GID

El siguiente objetivo es utilizar las imágenes del GID para entrenar y probar los modelos implementados en *semantic-segmentation-pytorch*. Para lograrlo, es necesario preparar las imágenes del GID adecuadamente para que puedan ser manejadas por los modelos. Además, también se deben realizar ciertos ajustes en el código para asegurar la compatibilidad y el correcto funcionamiento durante el proceso de entrenamiento y prueba.

4.2.1. Preprocesamiento del GID

La estructura actual de la carpeta, de donde se leen tanto las imágenes sin anotar como las anotadas, está dividida en datasets de entrenamiento y validación. Se utilizan dos archivos .odgt para mantener un registro de las rutas de las imágenes y sus respectivas máscaras para ambos datasets. El objetivo en esta sección es automatizar, mediante un script, la generación del contenido de esta carpeta usando imágenes del GID.

Este script comienza leyendo las imágenes multiespectrales del GID desde dos carpetas distintas: una para el entrenamiento y otra para la validación. Las correspondientes máscaras se leen desde una tercera carpeta. Este proceso de lectura se realiza utilizando funciones específicas para imágenes TIF y PNG. Aunque todas las imágenes son del GID, es necesario normalizarlas durante su lectura para asegurar la coherencia en el procesamiento.

La normalización de imágenes multiespectrales de sensado remoto es fundamental para garantizar la uniformidad de los datos, mejorar el rendimiento de los modelos y reducir sesgos causados por variaciones en las condiciones ambientales y de captura. Este proceso es crucial para poder entrenar los modelos con imágenes capturadas en diferentes momentos y condiciones. La normalización de las imágenes consiste en ajustar los valores de los píxeles a un rango específico para asegurar la consistencia y optimización en su procesamiento posterior. Este proceso comienza restando el valor mínimo de todos los píxeles de la imagen, lo que

establece que el rango de valores comience en 0. A continuación, se divide cada valor por la diferencia entre el máximo y el mínimo, normalizando así los datos al rango [0, 1]. Posteriormente, estos valores se escalan a 16 bits multiplicándolos por 65535, lo que ajusta los valores al rango de 0 a 65535.

Este procedimiento de normalización ofrece múltiples beneficios. En primer lugar, asegura que todos los valores de los píxeles estén en un rango consistente (de 0 a 65535), independientemente del rango original de la imagen. Esto es crucial para garantizar que los datos sean homogéneos y adecuados para análisis y procesamiento posteriores. Al escalar los datos al rango completo de 16 bits, se maximiza el uso del rango dinámico disponible, lo que puede mejorar la precisión y la sensibilidad de los algoritmos de procesamiento de imágenes y aprendizaje automático, aprovechando toda la resolución disponible. Además, las CNN a menudo funcionan mejor con datos normalizados, ya que así se facilita el entrenamiento y puede conducir a una mejor convergencia y estabilidad del mismo.

Por otro lado, las imágenes del GID son imágenes multiespectrales de grandes dimensiones y pesadas, esto provoca dos nuevas complicaciones:

Por un lado, el código del repositorio está diseñado para procesar imágenes mucho más ligeras y pequeñas. Las imágenes del ADE20K rara vez superan los 100 kB o dimensiones superiores a 1000x1000, mientras que las imágenes del GID ocupan 384,9 MB cada una y ocupan 7300x6908 píxeles. Esto puede ser un problema, ya que como se explicó anteriormente, el número de imágenes a procesar por cada GPU está definido en los archivos de configuración, siendo el mínimo una imagen por GPU. Si se usasen directamente estas imágenes, la GPU se quedaría sin memoria. Por otro lado, las imágenes del ADE2OK son de 3 bandas (RGB) y de 8 bits, almacenándose de forma sencilla con el formato JPEG. Sin embargo, las imágenes del GID que estamos utilizando tienen 4 bandas (R,G,B,Nir) y 16 bits, por lo que no se puede seguir guardando con este formato, ya que es incompatible con imágenes de estas características.

Para solucionar el problema del tamaño y peso de las imágenes, se opta por reducir su tamaño, recortándolas en parches de 500x500, un tamaño muy común en las imágenes del ADE20k, reduciendo así el peso de cada parche a 2MB. A pesar de recortar las imágenes, el peso de los parches sigue siendo muy alto. Esto se debe principalmente al uso de 4 bandas y 16 bits. Aun así, a diferencia de lo que ocurría cuando se intentaba entrenar cualquier modelo con el dataset ADE20K en la computadora personal, que daba un error por falta de memoria en la GPU, sí se puede realizar el entrenamiento con este dataset. Esto se debe a que el dataset del GID tiene 24 clases, mientras que el ADE20K utiliza 150. Este incremento en el número de clases impacta significativamente en el uso de memoria, ya que cada clase adicional requiere más memoria para los mapas de características y activaciones en la última capa del modelo, así como para los gradientes durante la retropropagación. Con 150 clases, la salida de la capa final tiene una dimensión

mucho mayor en comparación con solo 24 clases, lo que multiplica el uso de memoria en la capa de salida y los cálculos de la función de pérdida. Por lo tanto, aunque las imágenes del GID sean más grandes y pesadas, este menor número de clases permite que el entrenamiento se realice sin problemas de memoria, ya que la carga de memoria asociada a las clases es considerablemente menor.

De todas formas, por cuestiones de rendimiento, de aquí en adelante se utilizará el (HPC) cluster ctcomp3 del CiTIUS para realizar los entrenamientos de los modelos. El uso del Finisterrae III se descarta, ya que como ya se explicó en la sección anterior, no se obtiene prácticamente ningún beneficio al utilizar este equipo. Además, como se comentó en la sección 3.4.3, el directorio de trabajo que se nos ofrece en este servidor tiene un espacio de almacenamiento muy limitado, teniendo ya problemas para almacenar aquí el dataset del ADE20K, por lo que resulta inviable realizar el entrenamiento de los modelos con este nuevo dataset que ocupa mucho más espacio de almacenamiento.

Para abordar el problema del formato de las imágenes, se decide guardar los parches usando el formato GeoTiff [42]. GeoTiff es un formato abierto y ampliamente aceptado, lo que garantiza su compatibilidad con una gran variedad de software y herramientas de procesamiento de imágenes, como por ejemplo la librería GDAL. Pero lo más importante es que GeoTiff puede manejar imágenes multibanda y admite diferentes profundidades de bits, incluyendo 16 bits por banda.

Continuando con el funcionamiento del script: una vez se cortan las imágenes y las máscaras en parches, se guardan los parches de entrenamiento y los de validación en sus respectivos datasets. Al mismo tiempo se van anotando las rutas de los parches y sus correspondientes máscaras en dos archivos .odgt, uno para el entrenamiento y otro para la validación. De esta forma, las imágenes se mantienen localizadas para poder proceder con su lectura durante el entrenamiento.

Durante la elaboración de este script, es de gran utilidad la herramienta htool para asegurarnos de que la lectura, recorte y salvado de las imágenes se está realizando de forma correcta. Desde htool, mediante la opción *cut*, podemos recortar manualmente una imagen y obtener un parche del tamaño deseado. De esta forma logramos comprar visualmente los parches generados con htool y los generados por nuestro script. En la Figura 4.1 y Figura 4.2 podemos ver un ejemplo de una de las imágenes del GID antes de ser procesada y uno de los parches generados tras el postprocesado.

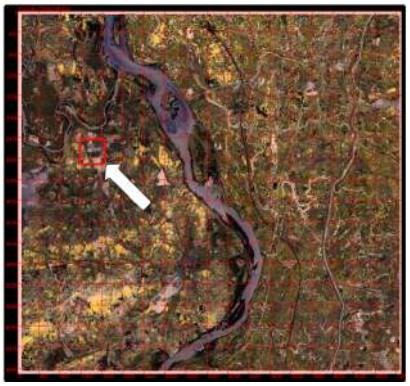


Figura 4.1: Imagen de 4 bandas extraída del GID

Figura 4.2: Ejemplo de parche generado tras el postprocesado

4.2.2. Adaptación del código para GID

Aparte de las transformaciones necesarias en el conjunto de datos de GID, es imprescindible adaptar el código original a las nuevas necesidades de trabajo con imágenes de 16 bits y 4 bandas en formato GeoTIFF. El código original, diseñado para manejar imágenes RGB de 8 bits en formato JPEG, utiliza la librería PIL [43] para leer, guardar y transformar las imágenes. Sin embargo, esta librería no permite procesar adecuadamente imágenes de mayor profundidad y bandas adicionales. Para resolver esto, se introduce la biblioteca GDAL, que es capaz de manejar imágenes multibanda y de alta profundidad de bits.

Primero, se modifica la función de lectura de imágenes para utilizar GDAL [36] en lugar de PIL. Esta nueva función, permite abrir y leer archivos GeoTIFF, extrayendo todas las bandas de la imagen y convirtiéndolas en un tensor de PyTorch. Este cambio es fundamental para garantizar que las imágenes se procesen correctamente, manteniendo toda la información espectral disponible en las imágenes de 16 bits.

Además, se ajusta el proceso de normalización de las imágenes. Se transforma el rango de valores de las imágenes a un rango de 0 a 1. Normalizar los datos a este rango antes de pasarlos a la red neuronal facilita el entrenamiento, ya que las redes neuronales tienden a converger más rápido y de manera más estable cuando los datos de entrada están en un rango pequeño y controlado.

A mayores, esta normalización es necesaria dado que muchas operaciones y transformaciones que se aplican durante el entrenamiento asumen que los datos están en el rango de 0 a 1. Esto incluye técnicas que se utilizan para ajustar las medias y desviaciones estándar de los canales de imagen. Por otro lado, trabajar con valores en el rango de 0 a 1 ayuda a evitar problemas de estabilidad numérica que pueden surgir cuando se operan con números muy grandes o muy pequeños,

especialmente durante la retropropagación de gradientes.

El código también se modifica para trabajar con imágenes de 4 bandas. Se ajustan así las operaciones de transformación de imágenes para que funcionen con cuatro canales en lugar de tres (RGB). Esto incluye actualizar las estadísticas de normalización (media y desviación estándar) para considerar la cuarta banda. Además, el tensor que almacena las imágenes ahora se dimensiona para cuatro canales, lo que asegura que todas las bandas se incluyan en el procesamiento.

Otro cambio significativo es la adaptación de las funciones de redimensionamiento de imágenes. Se modifican estas funciones para manejar tanto tensores de PyTorch como imágenes PIL, asegurando que las imágenes se redimensionen correctamente independientemente del formato de entrada. Para los tensores de PyTorch, se convierte primero el tensor a una imagen PIL, se redimensiona, y luego se convierte de nuevo a un tensor, asegurando que las operaciones de redimensionamiento se realicen correctamente.

En resumen, las modificaciones realizadas permiten adaptar el código para trabajar con las imágenes del GID, asegurando que todas las etapas del procesamiento de datos, desde la lectura y normalización hasta el redimensionamiento y transformación, sean compatibles con las nuevas especificaciones del dataset. Esto garantiza que los distintos modelos implementados puedan ser entrenados y evaluados de manera efectiva utilizando las imágenes postprocesadas del GID.

4.3. Técnicas y modelos empleados

Como ya se ha evidenciado en la revisión del estado del arte 2, las CNN se han consolidado como la técnica más popular y efectiva para tareas de segmentación semántica de imágenes. Esta popularidad se debe a su capacidad para capturar y representar características espaciales complejas a través de múltiples capas de convolución y operaciones de agrupamiento. En este contexto, se utilizan arquitecturas basadas en CNNs para abordar el problema de segmentación de imágenes multiespectrales del dataset GID.

De forma más específica, se hace uso de la arquitectura de codificador-decodificador. Esta arquitectura es una estructura común en tareas de segmentación semántica, donde el codificador actúa como una red de clasificación modificada y el decodificador consiste en capas convolucionales finales y de *upsampling*. Esta configuración permite extraer características relevantes de las imágenes y luego reconstruirlas a una resolución más alta para predecir etiquetas a nivel de píxel.

Antes de utilizar los distintos modelos implementados en el código base, se deben adaptar las estructuras de los codificadores y decodificadores para su correcto funcionamiento con las imágenes multiespectrales de 4 bandas del dataset GID.

Originalmente, los codificadores estaban diseñados para imágenes con 3 bandas (RGB). Por ello, es necesario ajustar la primera capa de convolución de estos modelos para imágenes multiespectrales de 4 bandas. Este ajuste implica cambiar el número de canales de entrada en la primera capa de convolución de 3 a 4. Esta modificación permite que los codificadores procesen adecuadamente la información adicional presente en las imágenes multiespectrales, manteniendo su capacidad de extraer características complejas y detalladas. Por otro lado, los decodificadores tienen definido un parámetro con el número de clases existentes para la clasificación. El número de clases manejadas por los decodificadores se ajusta a 24, permitiéndoles procesar y clasificar imágenes en 24 categorías distintas. Esta modificación asegura que los modelos sean capaces de manejar correctamente el nuevo número de clases durante el entrenamiento y la inferencia.

No se utilizan los modelos preentrenados ofrecidos por los autores del repositorio debido a que estos modelos fueron entrenados originalmente con imágenes de 3 bandas (RGB). Al modificar la arquitectura del modelo para aceptar imágenes de 4 bandas, el número de canales de entrada hace que los pesos preentrenados, optimizados para imágenes de 3 bandas, no sean directamente aplicables a la nueva arquitectura con 4 bandas de entrada. Utilizar estos pesos preentrenados sin ajustarlos adecuadamente resultaría en un desajuste dimensional y en una ineficiencia en el aprendizaje, ya que el modelo no podría procesar correctamente la información adicional de la cuarta banda desde el inicio del entrenamiento. Además, esto daría lugar a un error de carga de los pesos preentrenados debido a la incompatibilidad dimensional.

A continuación, se detallan los codificadores y decodificadores con los que se realizarán las pruebas en este trabajo, junto con sus particularidades y ventajas.

4.3.1. Codificadores

- **MobilNetV2dilated**: La estructura del MobilNetV2dilated [45] está basada en la versión anterior MobileNetV1, un modelo eficiente y de baja latencia que utiliza convoluciones separables en profundidad para reducir el número de parámetros y el costo computacional. MobileNetV2 introduce varias mejoras significativas. Mantiene el uso de convoluciones depthwise y pointwise, permitiendo una reducción considerable en el costo computacional sin sacrificar precisión.

  MobileNetV2 incorpora cuellos de botella lineales, que preservan mejor la información en capas profundas y evitan la saturación de activaciones. También introduce residuos invertidos, que conectan capas a través de cuellos de botella con un número reducido de canales, mejorando la propagación del gradiente y la transformación no lineal.

  La versión dilatada de MobileNetV2 extiende el campo receptivo de las

capas convolucionales sin reducir la resolución espacial, crucial para la segmentación semántica de imágenes de alta resolución. Sin embargo, esta extensión incrementa significativamente el costo computacional y el uso de memoria durante el entrenamiento y la inferencia.

- **HRNet**: HRNet [46] es un modelo reciente que mantiene representaciones de alta resolución a lo largo del proceso de aprendizaje. Este modelo se caracteriza por mantener múltiples resoluciones en paralelo y fusionar continuamente estas representaciones, lo que le permite conservar detalles finos de la imagen y, al mismo tiempo, incorporar información contextual a múltiples escalas. HRNet ha demostrado un rendimiento sobresaliente en segmentación semántica, mostrando mejoras significativas en precisión y eficiencia en varios benchmarks, alcanzando así el rendimiento SOTA (estado del arte) en tareas de etiquetado de píxeles. Además, su diseño modular facilita la integración con diferentes tipos de decodificadores, aumentando su versatilidad.

  La arquitectura HRNet se ilustra en la Figura 4.3. La red conecta convoluciones de alta a baja resolución en paralelo y realiza fusiones multi-escala de manera repetida. Este diseño permite que las representaciones de alta resolución sean no solo fuertes sino también espacialmente precisas, mejorando la capacidad del modelo para capturar y procesar detalles finos de las imágenes.

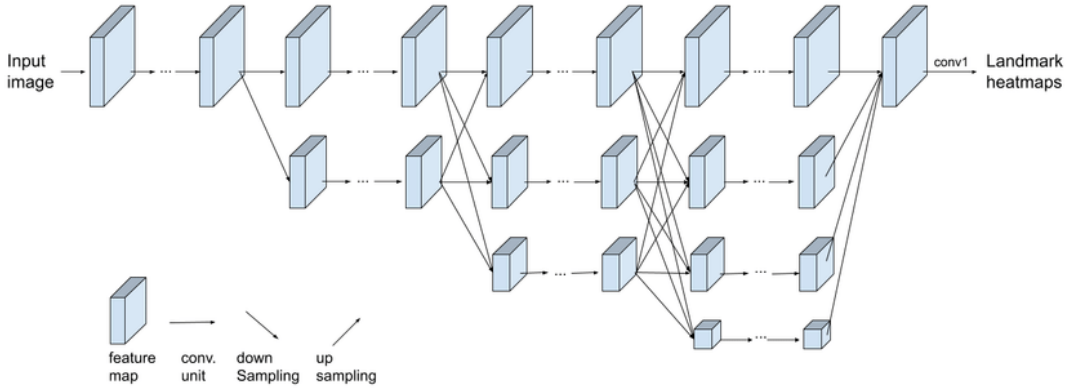


Figura 4.3: Ejemplo simple de una red de alta resolución. Hay cuatro etapas. La primera etapa consiste en convoluciones de alta resolución. La segunda (tercera, cuarta) etapa repite bloques de dos resoluciones (tres resoluciones, cuatro resoluciones). Imagen extraída de [47]

En resumen, HRNet es una arquitectura avanzada que conserva y fusiona continuamente múltiples resoluciones, lo que le permite mantener detalles finos y contextos amplios en las imágenes, destacándose en múltiples tareas

de visión por computadora.

- **ResNet y ResNetdilated**: Los modelos ResNet18, ResNet50 y ResNet101 son ampliamente reconocidos y utilizados en la comunidad de visión por computadora, como se comenta en la sección 2.1.1 del estado del arte. El número en sus nombres simplemente hace referencia al número de capas que tienen. Las versiones *dilated* utilizan convoluciones dilatadas para expandir el campo receptivo sin reducir la resolución espacial de las características, a costa de un incremento del costo computacional y el uso de memoria durante el entrenamiento y la inferencia. Este enfoque permite mantener un alto nivel de detalle en las características aprendidas, lo cual es crucial en tareas como la segmentación semántica y otros problemas de visión por computadora que requieren precisión espacial. El modelo base que viene implementado en el código, personaliza estos modelos más allá de las implementaciones estándar disponibles en torchvision, diseñados para optimizar su rendimiento en tareas específicas de segmentación.

4.3.2. Decodificadores

- **PSPNet**: PSPNet [48] es una red de segmentación que agrega representación global mediante el Módulo de Pirámide de Pooling (PPM) (véase Figura 4.4). Este módulo ayuda a integrar información contextual de diferentes escalas y regiones subyacentes del mapa de características, mejorando la capacidad del modelo para captar la información global y local de las imágenes. Utilizando cuatro niveles de escalas de pooling, la PPM fusiona características a diferentes niveles de granularidad, permitiendo una representación más rica y precisa. La PSPNet aprovecha esta información junto con las características locales extraídas por una red profunda, mejorando significativamente la precisión en tareas de segmentación de escenas.

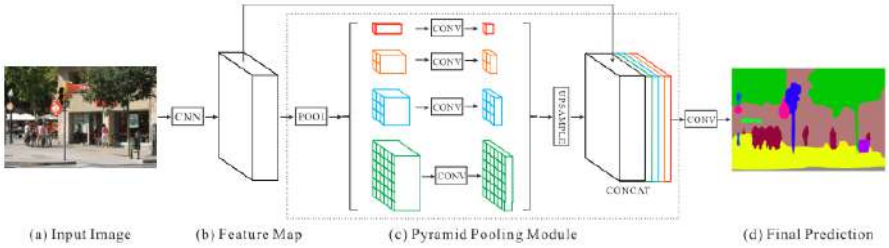


Figura 4.4: Vista general de un modelo PSPNet. Dada una imagen de entrada (a), primero se utiliza una CNN para obtener el mapa de características de la última capa de convolución (b), luego se aplica un módulo de parsing piramidal para recolectar diferentes representaciones de subregiones, seguido de capas de upsampling y concatenación para formar la representación final de características, que lleva tanto información de contexto local como global en (c). Finalmente, la representación se alimenta a una capa de convolución para obtener la predicción final por píxel (d). Figura obtenida en [48]

- **UPerNet**: El modelo UPerNet [49] es un modelo avanzado de segmentación semántica que combina la Red de Pirámide de Características (FPN) y el Módulo de Pirámide de Pooling (PPM) (véase Figura 4.5).

  La Red de Pirámide de Características (FPN) es un extractor de características genérico que explota representaciones de características a múltiples niveles en una jerarquía piramidal inherente. Utiliza una arquitectura de arriba hacia abajo con conexiones laterales para fusionar información semántica de alto nivel en los niveles medio y bajo con un costo marginal adicional. La FPN mejora la capacidad del modelo para capturar características de diferentes escalas, lo que es crucial para tareas que requieren un reconocimiento detallado y contextual de las imágenes.

  Este diseño permite que UPerNet aproveche las ventajas de ambas arquitecturas, realizando la captura de tanto las características locales como globales de las imágenes, mejorando la precisión y la calidad de la segmentación final. La arquitectura de UPerNet está diseñada para ser eficiente y efectiva, destacándose por su capacidad para entrenar más rápido y utilizar menos memoria GPU en comparación con otros modelos como PSPNet. Esto se logra mediante la eliminación de las convoluciones dilatadas y la integración del PPM, lo cual permite una mejor agregación de la información contextual global.

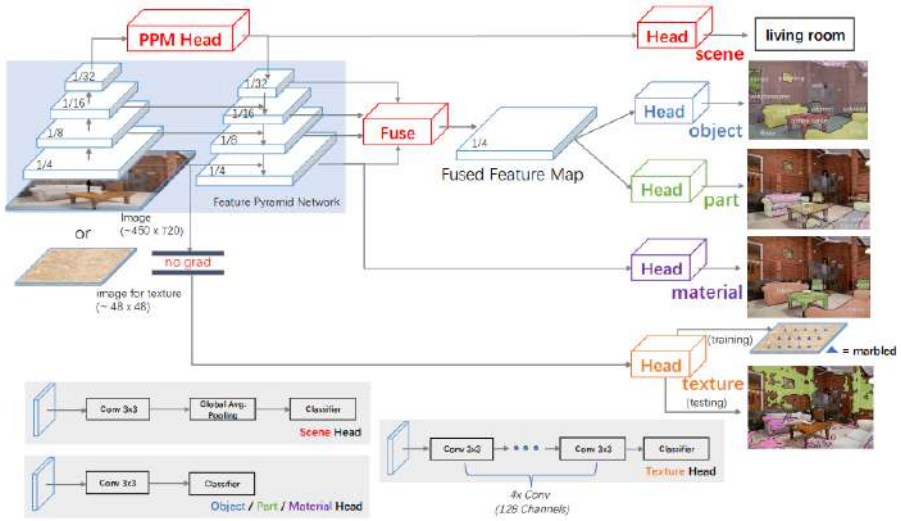


Figura 4.5: Arriba a la izquierda: La Red de Pirámide de Características (FPN) con un Módulo de Pooling Piramidal (PPM). Figura extraída de [49].

- **C1**: El decodificador C1 es una implementación simple y eficiente utilizada en redes de segmentación semántica. Su diseño se basa en el uso de una única capa convolucional que transforma directamente las características extraídas por el codificador en mapas de predicción de etiquetas a nivel de píxel. Esta simplicidad reduce la complejidad computacional y facilita la implementación, haciendo al decodificador C1 computacionalmente eficiente y rápido en términos de inferencia.

Tanto los decodificadores C1 como PSPNet pueden combinarse con la técnica de *deep supervision* [44], que añade supervisión adicional en capas intermedias para mejorar el entrenamiento de redes profundas. Esta técnica implica la adición de funciones de pérdida auxiliares en las capas intermedias de la red, proporcionando señales de error adicionales durante el entrenamiento. Esto ayuda a mitigar el problema de la desaparición del gradiente, que es común en redes profundas, y acelera la convergencia del modelo al proporcionar gradientes más robustos y estables en las etapas iniciales del entrenamiento.

En este trabajo, se ha implementado la técnica de *deep supervision* en el modelo PSPNet y C1, lo que permite a la red aprender representaciones intermedias más efectivas y mejorar el rendimiento general en tareas de segmentación semántica. Esta técnica es especialmente beneficiosa en redes como la PSPNet, donde la combinación de características a múltiples escalas se puede optimizar mejor gracias a estas señales de error adicionales, mejorando la precisión y la calidad de las predicciones finales.

Para asegurar un enfoque eficiente y manejable en la evaluación de los modelos,

se ha decidido restringir las combinaciones de codificadores y decodificadores a aquellas que han sido previamente probadas y documentadas en el repositorio original del MIT CSAIL. Probar todas las posibles combinaciones de codificadores y decodificadores resultaría en un número excesivo de experimentos, lo cual no solo sería inviable en términos de tiempo y recursos computacionales, sino que también complicaría el análisis y comparación de resultados.

Por tanto, las combinaciones seleccionadas son:

- **MobileNetV2dilated + C1_deepsup**
- **MobileNetV2dilated + PPM_deepsup**
- **ResNet18dilated + C1_deepsup**
- **ResNet18dilated + PPM_deepsup**
- **ResNet50dilated + PPM_deepsup**
- **ResNet101dilated + PPM_deepsup**
- **ResNet50 + UPerNet**
- **ResNet101 + UPerNet**
- **HRNet + C1**

En resumen, se desea probar todas las técnicas expuestas en esta sección para obtener una evaluación integral de sus capacidades y beneficios en la tarea específica de segmentación semántica de imágenes multiespectrales. Estas técnicas y combinaciones han sido elegidas no solo por su rendimiento documentado en estudios previos, sino también por su relevancia y novedad en el campo de la segmentación semántica. Esto nos permitirá no solo confirmar su efectividad en el contexto de nuestro conjunto de datos, sino también explorar oportunidades para mejoras adicionales a través de la optimización de hiperparámetros y técnicas avanzadas, asegurando así que los modelos seleccionados alcancen su máximo potencial en términos de rendimiento y precisión.

Capítulo 5

Pruebas

En este capítulo se presenta el plan de pruebas diseñado para verificar la funcionalidad y calidad global del experimento, así como los resultados obtenidos de dicho plan. Este proceso es crucial para evaluar la efectividad de los modelos de segmentación semántica entrenados y garantizar que cumplan con los objetivos planteados en el Trabajo de Fin de Grado.

Para asegurar la replicabilidad de estos experimentos y facilitar su comparación con otros estudios, se explicarán en detalle las condiciones en las que fueron realizados. Esto incluye una documentación exhaustiva de los conjuntos de datos utilizados, la configuración de los parámetros de entrenamiento, los métodos de evaluación aplicados y el equipo en el que se desarrollaron los experimentos.

5.1. Plan de pruebas

El experimento consiste en el entrenamiento de los modelos expuestos al final de la sección 4.3 para su posterior evaluación en la tarea de segmentación semántica en imágenes multiespectrales. El proceso se divide en dos fases de pruebas diferenciadas:

1. **Prueba preliminar**: En esta fase se evaluarán todos los modelos en múltiples épocas para obtener resultados preliminares detallados. El objetivo es identificar tanto la arquitectura del modelo más prometedora como la época en la que cada modelo alcanza su mejor rendimiento en términos de precisión y eficacia. Esta evaluación inicial permite detectar problemas de sobreajuste y asegurar que los modelos generalizan bien para datos no vistos. Los resultados de esta fase proporcionarán una base sólida para seleccionar el modelo y la época más prometedores para la evaluación final.

2. **Prueba final**: Basándose en los resultados obtenidos en la fase preliminar, se seleccionará la mejor arquitectura de modelo y la época óptima para una evaluación detallada de su rendimiento. El modelo seleccionado se someterá a una evaluación final utilizando el conjunto de datos de test. Esta fase tiene como objetivo proporcionar una medida objetiva y definitiva del rendimiento del modelo en un contexto real simulado. La evaluación final permitirá verificar la eficacia del modelo y su capacidad de generalización, asegurando que cumple con los objetivos planteados para la tarea de segmentación semántica en imágenes multiespectrales.

5.2. Condiciones del experimento

En esta sección se describen detalladamente las condiciones y procedimientos seguidos para llevar a cabo los experimentos de segmentación semántica de imágenes multiespectrales. Esto incluye la preparación y división de los conjuntos de datos, los parámetros utilizados para el entrenamiento de los modelos, y las metodologías aplicadas para la evaluación preliminar y final de los mismos. La correcta implementación de estas condiciones es crucial para asegurar la validez y reproducibilidad de los resultados obtenidos.

5.2.1. Conjuntos de datos

Para realizar el experimento de manera efectiva, es esencial dividir los datos utilizados en tres conjuntos distintos: entrenamiento, validación y test. Esta división asegura que los modelos sean entrenados y evaluados de manera robusta y precisa.

- **Conjunto de entrenamiento**: Es el grupo de datos utilizado para ajustar los pesos del modelo durante su entrenamiento. Durante cada época de entrenamiento (ciclo completo en el cual el modelo pasa por todo el conjunto de datos de entrenamiento una vez), el modelo aprende a reconocer patrones y características en las imágenes, ajustando sus pesos internos para minimizar el error en sus predicciones. Este conjunto debe ser lo suficientemente grande y representativo para que el modelo pueda generalizarse bien con nuevas imágenes. Las métricas impresas durante el entrenamiento, como la pérdida y la precisión, reflejan el rendimiento del modelo en este conjunto de datos.
- **Conjunto de validación**: Se utiliza para evaluar el rendimiento del modelo en datos que no ha visto antes durante el entrenamiento, una vez que el entrenamiento ha concluido. Este conjunto proporciona una medida independiente del rendimiento que ayuda a detectar el sobreajuste (*overfitting*), un fenómeno donde el modelo se ajusta demasiado a los datos de

entrenamiento y no generaliza bien para datos nuevos. Si el rendimiento en el conjunto de validación es significativamente peor que en el conjunto de entrenamiento, es una señal de sobreajuste. Además, el conjunto de validación se utiliza para seleccionar el modelo óptimo entre varias alternativas, asegurando que el modelo elegido tiene la mejor capacidad de generalización antes de realizar la evaluación final con el conjunto de test. El conjunto de validación, por lo tanto, es crucial para garantizar que el modelo generaliza bien para datos no vistos.

- **Conjunto de test**: Es un conjunto de datos completamente separado que se utiliza para evaluar el rendimiento final del modelo después de la evaluación con el conjunto de validación. Proporciona una estimación imparcial del rendimiento del modelo en situaciones del mundo real, ya que no se ha utilizado en ninguna etapa del entrenamiento o ajuste del modelo. Este conjunto es esencial para obtener una medida objetiva de la capacidad del modelo.

Es crucial que los datos en estos tres conjuntos sean distintos para evitar la contaminación del modelo. Si los mismos datos se utilizan en más de un conjunto, el modelo podría aprender características específicas de esos datos en lugar de generalizar patrones útiles para datos no vistos, resultando en una evaluación sesgada y en un modelo que no se desempeña bien en situaciones reales.

Estos tres conjuntos de datos están compuestos por imágenes del GID. Las imágenes multiespectrales utilizadas son de 4 bandas y 16 bits sin anotaciones. Además, se utilizan sus correspondientes imágenes anotadas, que están indexadas en 24 clases, como se describe en la sección 3.1.2. De las 150 imágenes ofrecidas por este dataset, usamos 13 imágenes. 9 de ellas (70 %) conforman el conjunto de entrenamiento, 2 (15 %) el conjunto de validación, y las dos (15 %) restantes el de test. Todas las imágenes se descomponen en parches de 500x500 obteniendo así 1890 parches en el conjunto de entrenamiento y 420 parches en el conjunto de validación y en el conjunto de test.

La decisión de utilizar un número reducido de imágenes en los conjuntos de entrenamiento, validación y test se debe a la alta resolución y complejidad de las imágenes multiespectrales de 16 bits. Estas características hacen que las imágenes sean extremadamente pesadas y costosas en términos de procesamiento y almacenamiento. Utilizar un mayor número de imágenes aumentaría significativamente el tiempo de entrenamiento y la demanda computacional, lo que no es práctico con los recursos disponibles. Al reducir la cantidad de imágenes, logramos un equilibrio entre la representatividad de los datos y la viabilidad práctica del experimento, permitiendo un entrenamiento y evaluación eficientes sin comprometer demasiado la calidad de los resultados.

5.2.2. Entrenamiento de los modelos

Para entrenar los modelos en la tarea de segmentación semántica, se indica mediante un archivo de configuración los parámetros del entrenamiento:

- **batch_size_per_gpu**: Define el número de muestras que se procesan en cada GPU durante una iteración del entrenamiento. Un tamaño de lote más grande puede llevar a un entrenamiento más rápido y estable, pero también requiere más memoria.
- ***num_epoch***: Especifica el número total de épocas para el entrenamiento. Una época corresponde a un ciclo completo a través del conjunto de datos de entrenamiento.
- ***start_epoch***: Indica la época inicial desde la cual se inicia o reinicia el entrenamiento. Este parámetro es útil para retomar el entrenamiento desde un punto guardado previamente.
- ***epoch_iters***: Define el número de iteraciones por cada época. Este valor determina cuántos lotes se procesan en cada época.
- ***optim***: Especifica el optimizador utilizado para ajustar los pesos del modelo. "SGD"(*Stochastic Gradient Descent*) es uno de los optimizadores más comunes. En la sección 3.5.3 se detalla su funcionamiento.
- ***lr_encoder***: Establece la tasa de aprendizaje inicial para el codificador del modelo. La tasa de aprendizaje controla el tamaño del paso que da el optimizador durante el ajuste de los pesos.
- ***lr_decoder***: Define la tasa de aprendizaje inicial para el decodificador del modelo.
- ***lr_pow***: Es el exponente utilizado en el ajuste de la tasa de aprendizaje a lo largo del tiempo. Un valor menor a 1 disminuirá la tasa de aprendizaje de forma más lenta, permitiendo ajustes más finos en las últimas etapas del entrenamiento.
- ***beta1***: Es el parámetro de momento utilizado en el optimizador SGD, que ayuda a acelerar el entrenamiento al considerar el gradiente acumulado de iteraciones anteriores.
- ***weight_decay***: Es un término de regularización que penaliza grandes pesos en el modelo para evitar el sobreajuste.
- ***deep_sup_scale***: Establece el peso de la pérdida auxiliar en la supervisión profunda. Ayuda a proporcionar señales de error adicionales en capas intermedias para mejorar el entrenamiento.

- ***fix_bn***: Un valor booleano que indica si las capas de normalización por lotes (*Batch Normalization*) deben ser fijadas (no actualizadas) durante el entrenamiento. Esto puede ser útil para estabilizar el entrenamiento en ciertas condiciones.
- ***workers***: Define el número de subprocesos de carga de datos utilizados durante el entrenamiento. Más trabajadores pueden acelerar la preparación de datos, pero también requieren más recursos del sistema.
- ***disp_iter***: Establece el número de iteraciones entre cada visualización de los resultados de entrenamiento en la consola. Esto ayuda a monitorear el progreso del entrenamiento en tiempo real.
- ***seed***: Es la semilla para el generador de números aleatorios, lo que garantiza la reproducibilidad del entrenamiento. Al usar la misma semilla, se pueden obtener resultados consistentes en diferentes ejecuciones del mismo experimento.

Todos los modelos se entrenan con la misma configuración de parámetros, en concreto, se utiliza la configuración predefinida en el repositorio del código base (veáse Cuadro 5.1)

Parámetro	**Valor**
batch_size_per_gpu	3
num_epoch	100
start_epoch	0
epoch_iters	5000
optim	SGD
lr_encoder	0.02
lr_decoder	0.02
lr_pow	0.9
beta1	0.9
weight_decay	1e-4
deep_sup_scale	0.4
fix_bn	False
workers	16
disp_iter	20
seed	304

Cuadro 5.1: Parámetros de entrenamiento utilizados en el experimento

El entrenamiento de todos los modelos se realiza desde el HPC Cluster ctcomp3 del CiTIUS, usando la versión del código adaptada para el entrenamiento con una GPU. El software y hardware utilizado por este equipo se especifica en la

sección 3.4.2. Se utilizan 100 épocas para entrenar los modelos. Durante el entrenamiento, se guardan los pesos del modelo en cada época. Esto permite evaluar el rendimiento del modelo en una época específica o reanudar el entrenamiento desde una época guardada. También se imprimen por pantalla algunos parámetros y métricas para poder monitorear el proceso del entrenamiento, a continuación se explican los más significativos:

- *Epoch*: Indica que la época en la que se encuentra el modelo.
- Time: El tiempo, en segundos, que tomó realizar la última iteración. Este valor es útil para monitorear la velocidad del entrenamiento.
- *lr_encode*r: La tasa de aprendizaje actual para el codificador.
- *lr_decoder*: La tasa de aprendizaje actual para el decodificador.
- *Accuracy*: La precisión del modelo en el conjunto de datos utilizado durante la época actual.
- *Loss*: El valor de la función de pérdida calculada durante esta época. La pérdida es una medida de cuán bien o mal el modelo está funcionando; un valor más bajo indica un mejor rendimiento.

La precisión del modelo impresa durante el entrenamiento nos ayudará a detectar posibles casos de sobreajuste posteriormente, al comparar esta medida con la obtenida en la evaluación con el dataset de validación. Por otro lado, los valores de pérdida nos permitirán saber la capacidad del modelo durante el entrenamiento de predecir correctamente. Monitorear la *Loss* junto con la precisión nos proporciona una visión más completa del rendimiento del modelo y su capacidad para aprender de los datos sin sobreajustarse.

Para calcular los tiempos de entrenamiento, registramos las marcas de tiempo al inicio y al final del proceso de entrenamiento. Esto nos permite calcular la duración total del entrenamiento, lo que es esencial para evaluar la eficiencia del modelo y del hardware utilizado. Todas estas métricas están expuestas en el Cuadro 5.2, para tener información sobre el proceso de entrenamiento y poder analizarlo.

5.2.3. Evaluación de los modelos

Una vez que los modelos han sido entrenados, se realizan evaluaciones en dos fases: la Prueba Preliminar y la Prueba Final. Ambas evaluaciones se llevan a cabo en una computadora personal, a diferencia del entrenamiento, que se ejecuta en un clúster de alto rendimiento. Esto se debe a que la evaluación requiere menos recursos computacionales y puede realizarse de manera más conveniente en un PC personal, proporcionando facilidad y comodidad sin comprometer el rendimiento. El software y hardware utilizado por este equipo se especifica en la

sección 3.4.1. Estas fases de evaluación nos proporcionan las métricas de precisión e IoU, descritas en la sección 3.5.1, para cada modelo, permitiendo compararlos de manera objetiva y analizar su eficiencia. Es importante destacar que para el cálculo de estas métricas se ignoró la clase sin etiquetar (0), al igual que durante el entrenamiento, esto se puede hacer a través de la función de pérdida utilizada, como se refleja en la sección 3.5.2.

- **Evaluación de la prueba preliminar**: La evaluación de la Prueba Preliminar utiliza el conjunto de datos de validación para evaluar todos los modelos entrenados en varias épocas específicas. Esta fase permite evaluar cómo se comporta cada modelo con datos no vistos durante el entrenamiento, ayudando a detectar problemas de sobreajuste y a seleccionar tanto el modelo más óptimo como la mejor época en la que el modelo muestra su mejor rendimiento. Las épocas utilizadas para esta evaluación son: 5, 10, 20, 50, 75 y 100. Estas épocas han sido seleccionadas estratégicamente para capturar el progreso y los cambios significativos en el rendimiento del modelo a lo largo del entrenamiento. Evaluar en estas épocas permite observar tanto el aprendizaje inicial como la estabilización y la capacidad de generalización del modelo, ayudando a identificar el mejor modelo y la mejor época para la evaluación final.

  El código de evaluación crea un *dataloader* para cargar los datos de validación, construye los modelos con pesos preentrenados y procesa cada parche por separado. El modelo predice la clase de cada píxel y compara las predicciones con las imágenes anotadas para evaluar el rendimiento, proporcionando una base para seleccionar el modelo y la época óptimos.

- **Evaluación de la prueba final**: La evaluación de la Prueba Final se realiza utilizando el conjunto de datos de test para evaluar el modelo y la época seleccionados en la prueba preliminar. El conjunto de datos de test no ha sido utilizado durante el entrenamiento ni la validación. Esta evaluación proporciona una medida imparcial del rendimiento del modelo en situaciones del mundo real, garantizando su capacidad de generalización.

  El código usado para esta evaluación comienza creando un *dataloader* para cargar los datos del conjunto de validación de manera eficiente. Se construyen los modelos de codificador y decodificador, cargando los pesos preentrenados guardados durante el entrenamiento. Cada parche del conjunto de validación se procesa por separado utilizando el modelo de segmentación entrenado. Durante esta fase de inferencia, el modelo predice la clase para cada píxel y estas predicciones se comparan con los parches anotados para evaluar el rendimiento del modelo. Los resultados de esta evaluación proporcionan una base para la selección del modelo y la época más prometedores.

Finalmente, se imprimen las métricas globales para todas las imágenes y se almacenan los resultados procesados. Además, se hace una reconstrucción de los parches inferidos para generar las imágenes completas. Se guardan así visualizaciones que comparan las imágenes originales, las imágenes anotadas y las predicciones del modelo, permitiendo una evaluación visual cualitativa adicional del rendimiento del modelo.

5.3. Resultados

A continuación se presentan y analizan los resultados de la Prueba Preliminar y la Prueba Final, para proseguir con su posterior discusión en el capítulo 6.

5.3.1. Prueba preliminar

Un análisis superficial de los datos en el Cuadro 5.3 revela las precisiones e IoU media más altas obtenidas por los modelos (destacadas en negrita). El modelo ResNet18dilated + C1_deepsup entrenado por 100 épocas presenta la mayor precisión con un valor de 86.04 %. Sin embargo, el modelo ResNet50dilated + PPM_deepsup obtiene el valor más alto de Mean IoU con 0.3044. La precisión y la IoU mejoran consistentemente con el aumento del número de épocas hasta alcanzar un punto de saturación donde los incrementos son marginales. Basado en estos resultados, se recomienda utilizar el modelo ResNet18dilated + C1_deepsup entrenado por 100 épocas para obtener la mejor precisión y la segunda Mean IoU más alta en la evaluación. Se escoge este modelo por su superior precisión global y su reducido tiempo de entrenamiento (veáse Cuadro 5.2).

Modelo	Métrica	5 épocas	10 épocas	20 épocas	40 épocas	60 épocas	80 épocas	100 épocas	Tiempo
MobileNetv2dilated + C1_deepsup	**Precisión**	86.40 %	90.27 %	93.18 %	94.76 %	98.05 %	98.62 %	99.29 %	**12h 44min**
	Pérdida	0.635604	0.456130	0.317467	0.223589	0.098036	0.071044	0.038788	
MobileNetv2dilated + PPM_deepsup	**Precisión**	84.19 %	89.96 %	93.27 %	96.64 %	97.68 %	98.70 %	99.31 %	17h 33min
	Pérdida	0.756929	0.489425	0.331899	0.177163	0.123783	0.072519	0.039767	
ResNet18dilated + C1_deepsup	**Precisión**	87.10 %	91.71 %	95.71 %	97.90 %	99.57 %	99.60 %	99.89 %	**14h 46min**
	Pérdida	0.579536	0.376770	0.205600	0.112523	0.028899	0.027909	0.012967	
ResNet18dilated + PPM_deepsup	**Precisión**	88.56 %	93.02 %	96.36 %	98.26 %	98.77 %	99.09 %	99.21 %	22h 3min
	Pérdida	0.557581	0.352797	0.198883	0.104749	0.076020	0.058370	0.050926	
ResNet50dilated + PPM_deepsup	**Precisión**	88.16 %	92.11 %	96.36 %	97.75 %	98.53 %	99.65 %	99.94 %	38h 35min
	Pérdida	0.551391	0.365149	0.205094	0.108285	0.071259	0.016514	0.004862	
Resnet50 + UperNet	**Precisión**	85.72 %	91.19 %	94.90 %	96.70 %	98.99 %	99.30 %	99.72 %	32h 48min
	Pérdida	0.444260	0.269886	0.149872	0.108285	0.028162	0.019544	0.007392	
Resnet101dilated + PPM_deepsup	**Precisión**	86.65 %	91.27 %	94.22 %	98.10 %	98.86 %	99.39 %	99.88 %	56h 4min
	Pérdida	0.625216	0.401390	0.265386	0.086570	0.022855	0.026047	0.005862	
Resnet101 + UperNet	**Precisión**	87.48 %	91.93 %	95.55 %	98.36 %	99.17 %	99.49 %	99.73 %	37h 21min
	Pérdida	0.388471	0.244610	0.131199	0.046345	0.022855	0.013708	0.007302	
HRNet + C1	**Precisión**	89.60 %	93.17 %	95.84 %	98.56 %	99.14 %	99.60 %	99.78 %	51h 47min
	Pérdida	0.325643	0.208209	0.122807	0.042639	0.010664	0.013708	0.005846	

Cuadro 5.2: Precisión y pérdida de los modelos durante el entrenamiento, para las épocas 5, 10, 20, 40, 60, 80 y 100. Se indica el tiempo de entrenamiento para cada modelo.

Modelo	Métrica	5 épocas	10 épocas	20 épocas	40 épocas	60 épocas	80 épocas	100 épocas
MobileNetv2dilated + C1_deepsup	**Precisión**	68.92 %	73.20 %	84.88 %	82.96 %	84.54 %	84.11 %	85.15 %
	Mean IoU	0.0912	0.1796	0.2626	0.2488	0.2636	0.2657	0.2832
Mobilenetv2dilated + PPM_deepsup	**Precisión**	47.40 %	66.08 %	80.15 %	82.28 %	85.10 %	85.66 %	85.62 %
	Mean IoU	0.0686	0.1932	0.2084	0.2560	0.2723	0.2940	0.3033
ResNet18dilated + C1_deepsup	**Precisión**	68.82 %	78.25 %	80.91 %	68.65 %	84.54 %	85.57 %	**86.04** %
	Mean IoU	0.1212	0.1273	0.2327	0.2560	0.2638	0.2950	0.3010
ResNet18dilated + PPM_deepsup	**Precisión**	58.95 %	77.00 %	81.39 %	82.19 %	84.73 %	84.47 %	84.01 %
	Mean IoU	0.1585	0.2252	0.2525	0.2595	0.2742	0.2785	0.2787
ResNet50dilated + PPM_deepsup	**Precisión**	54.64 %	53.27 %	52.92 %	83.70 %	84.87 %	84.72 %	85.72 %
	Mean IoU	0.1315	0.1190	0.1721	0.2772	0.2841	0.2882	0.3044
ResNet50 + UperNet	**Precisión**	56.04 %	54.14 %	74.57 %	72.59 %	79.63 %	79.45 %	83.00 %
	Mean IoU	0.1201	0.1379	0.1677	0.2087	0.2201	0.2279	0.2546
ResNet101dilated + PPM_deepsup	**Precisión**	68.87 %	60.86 %	73.31 %	80.46 %	84.17 %	84.68 %	83.66 %
	IoU	0.1778	0.1569	0.2158,	0.2442	0.2989	0.2766	0.2897
ResNet101 + UperNet	**Precisión**	61.52 %	60.70 %	73.93 %	76.49 %	81.27 %	84.02 %	84.27 %
	Mean IoU	0.1716	0.1267	0.2110	0.2196	0.2320	0.2651	0.2695
HRNet + C1	**Precisión**	78.94 %	65.63 %	68.61 %	58.32 %	83.75 %	84.25 %	84.25 %
	Mean IoU	0.1873	0.1860	0.2040	0.1862	0.2760	0.2839	0.2840

Cuadro 5.3: Precisión e IoU media de los modelos ante la evaluación con el conjunto de validación, para las épocas 5, 10, 20, 40, 60, 80 y 100.

5.3.2. Prueba final

A continuación, se presentan los resultados del modelo ResNet18dilated + C1_deepsup en términos de IoU por clase, IoU media y precisión para dos imágenes distintas del conjunto de test (veáse Cuadro 5.4), junto con una comparativa visual de las imágenes originales, anotadas e inferidas (veáse Figura 5.1).

Clase	Imagen 1 IoU	Imagen 2 IoU
industrial area	0.2805	0.5547
paddy field	0.0000	0.9435
irrigated field	0.6550	0.0000
dry cropland	No presente	0.0000
garden land	0.0775	0.1766
arbor forest	0.0679	0.0032
shrub forest	0.0004	0.3400
park	No presente	No presente
natural meadow	0.0069	No presente
artificial meadow	No presente	0.0032
river	0.7436	0.3400
urban residential	0.0157	0.0838
lake	No presente	0.0023
pond	0.0000	0.0192
fish pond	No presente	0.6049
snow	No presente	No presente
bareland	0.1458	0.0546
rural residential	0.8125	0.8753
stadium	No presente	No presente
square	No presente	0.0000
road	0.6295	0.5969
overpass	0.0000	0.5524
railway station	No presente	No presente
airport	No presente	No presente
Mean IoU	0.2454	0.3205
Precisión	57.56 %	87.74 %

Cuadro 5.4: Resultados de test del modelo ResNet18dilated + C1_deepsup: IoU por clase, IoU media (de clases presentes) y precisión para las imágenes 1 y 2 del conjunto de Test

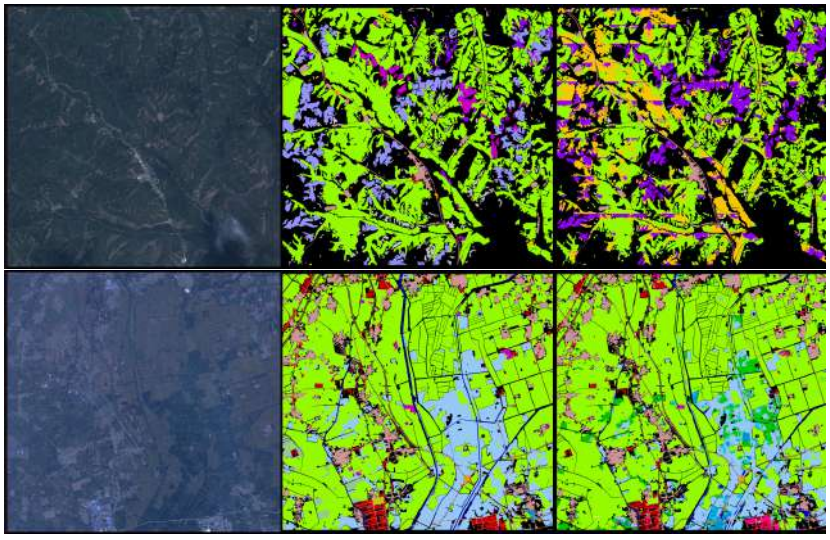


Figura 5.1: Comparativa visual de imagen 1 (superior) y 2 (inferior): De izquierda a derecha: imagen satelital, imagen anotada e imagen inferida

Capítulo 6

Discusión de los resultados

Los resultados obtenidos en el capítulo 5 muestran el rendimiento de varios modelos de segmentación semántica y permiten seleccionar el modelo más prometedor, ResNet18dilated + C1_deepsup, evaluado con dos imágenes de prueba.

En la prueba preliminar, todos los modelos muestran un incremento en precisión e IoU media con el aumento de épocas de entrenamiento, pero estos incrementos se vuelven mínimos después de aproximadamente 60 épocas, indicando un punto de saturación. Esto sugiere que utilizar más de 60 épocas no aporta mejoras significativas. Además, se observan signos de un ligero sobreentrenamiento en algunos modelos. Esto se refleja en el hecho de que algunos modelos, pasado el punto de saturación, tienen un peor rendimiento.

El modelo ResNet18dilated + C1_deepsup destaca con una alta precisión (86.04 %) y una buena Mean IoU (0.3010) con un tiempo de entrenamiento menor (14 horas y 46 minutos), haciéndolo eficiente y robusto. Por otro lado, aunque el modelo ResNet50dilated + PPM deepsup obtiene la mejor Mean IoU (0.3044), su tiempo de entrenamiento es significativamente mayor (38 horas y 35 minutos).

En la prueba final, la ResNet18dilated + C1_deepsup muestra variabilidad en el rendimiento entre las dos imágenes evaluadas, sugiriendo que el tamaño del conjunto de datos de entrenamiento podría ser insuficiente para capturar la variabilidad necesaria para un buen rendimiento generalizado. La falta de datos suficientes puede explicar la variabilidad en los resultados y la saturación temprana observada. Además, las arquitecturas más profundas, que requieren más tiempo de entrenamiento, no siempre obtienen mejores resultados, como se observa con el modelo ResNet101dilated + PPM_deepsup.

Es importante destacar que la métrica principal en este trabajo es la precisión, ya que la IoU media tiene problemas al ser muy sensible a las clases minoritarias y al ruido en los datos, lo que puede llevar a interpretaciones erróneas del

rendimiento del modelo. Este problema es relevante en nuestros experimentos, donde el conjunto de datos de entrenamiento limitado puede causar la ausencia de ciertas clases, afectando la IoU media. Por lo tanto, se prioriza la precisión para reflejar mejor la exactitud del modelo en la clasificación de píxeles.

En comparación con otras técnicas del estado del arte 2, los resultados obtenidos no se quedan atrás e incluso pueden ser superiores. En [14], se prueban técnicas como la fusión de información multiescalar y la segmentación jerárquica usando una ResNet50 con el dataset GID, obteniendo una precisión del 87.12 %, aunque usando solo 5 clases, lo que simplifica la tarea. Similarmente, en [16], se utilizan técnicas de detección de borde con una FCN, logrando precisiones en torno al 90 %, pero también con solo 5 clases.

De esta forma, se confirma la hipótesis planteada: los algoritmos de aprendizaje profundo diseñados para imágenes RGB, adaptados para imágenes multiespectrales, pueden producir una segmentación semántica de alta calidad. Este trabajo ha demostrado que es posible adaptar modelos de segmentación semántica utilizando PyTorch para trabajar con imágenes multiespectrales. Los resultados han validado su rendimiento, mostrando que el modelo desarrollado puede igualar o superar a los modelos del estado del arte en precisión y efectividad, contribuyendo significativamente a la investigación en visión por computadora y procesamiento de imágenes multiespectrales.

Capítulo 7

Conclusiones y posibles ampliaciones

En este Trabajo de Fin de Grado se estudió y se experimentó con la técnica de segmentación semántica de imágenes multiespectrales de sensado remoto, utilizando algoritmos de aprendizaje profundo en el entorno de PyTorch. Se desarrolló una metodología de procesado de imágenes multiespectrales para explotar la inmensa cantidad de datos disponible en este tipo de imágenes. Se adaptaron modelos de redes neuronales profundas diseñadas para su uso con imágenes estándar RGB, para el trabajo con imágenes multiespectrales. El modelo ResNet18dilated + C1_deepsup destacó al alcanzar una alta precisión con tiempos de entrenamiento relativamente bajos, demostrando ser una opción eficiente y robusta.

De esta forma, las principales contribuciones de este trabajo son las siguientes:

1. Se demostró experimentalmente que los modelos de segmentación semántica diseñados para imágenes RGB pueden adaptarse eficazmente a imágenes multiespectrales, logrando resultados de alta calidad. Esto valida la hipótesis planteada inicialmente.

2. Se identificó que, para el conjunto de datos utilizado, después de aproximadamente 60 épocas de entrenamiento, los incrementos en precisión y Mean IoU se vuelven mínimos, indicando un punto de saturación. Sugiriendo así que, para el conjunto de datos de entrenamiento utilizado, entrenar los modelos por más de 60 épocas no proporciona mejoras significativas en el rendimiento, pero sí incrementa considerablemente el tiempo de computación.

3. Se comprobó que las arquitecturas más profundas no necesariamente ofrecen los mejores resultados. Modelos más simples, como la ResNet18dilated + C1_deepsup, alcanzaron una alta precisión y buen rendimiento con mucho

menos tiempo de entrenamiento en comparación con modelos más complejos y profundos como la ResNet101dilated + PPM_deepsup.

4. En las pruebas finales, se evidenció una notable variabilidad en el rendimiento del modelo entre diferentes imágenes de prueba. Esto sugiere que el tamaño del conjunto de datos de entrenamiento podría ser insuficientemente grande y/o variado para presentar un rendimiento consistente en diversas escenas.

En cuanto a las posibles ampliaciones del trabajo, se identificaron dos principales áreas de mejora:

1. Aumentar el tamaño y la diversidad del conjunto de datos de entrenamiento. Esto permitiría capturar mayor variabilidad en las imágenes multiespectrales y podría mejorar la robustez y consistencia de los modelos, postergando el punto de saturación encontrado.

2. Optimizar los hiperparámetros de los modelos para revelar configuraciones más eficientes y efectivas, reduciendo el tiempo de computación y logrando un mejor rendimiento global. Esto incluye ajustar parámetros como la tasa de aprendizaje, el tamaño del lote, el número de épocas de entrenamiento...

Estas mejoras no solo podrían aumentar la precisión y eficiencia de los modelos desarrollados, sino que también podrían ampliar significativamente las aplicaciones prácticas de la segmentación semántica de imágenes multiespectrales en diversos campos como la agricultura, la gestión de recursos naturales y la planificación urbana.

Apéndice A

Manual técnico

Este manual está diseñado para proporcionar todas las instrucciones necesarias para la instalación, configuración, utilización y solución de problemas del software desarrollado en este TFG para la segmentación semántica de imágenes multiespectrales. Aquí se detallan los pasos para obtener el código, las dependencias de software y hardware, la replicación de los experimentos, la modificación de pruebas y la resolución de problemas comunes en la instalación de dependencias, asegurando una experiencia fluida y accesible para los usuarios.

A.1. Obtención del código

Todo el código desarrollado para la realización de las pruebas y sus respectivas evaluaciones está subido en un repositorio de GitHub, de acceso completamente público. De esta forma, cualquiera puede replicar los experimentos presentados.

A.2. Dependencias software y hardware

Los requerimientos de software incluyen Ubuntu 16.04.3 LTS (aunque también puede funcionar con otras distribuciones más modernas), CUDA$\geq$8.0, Python$\geq$3.5 y PyTorch$\geq$0.4.0. Las dependencias de software se especifican en el archivo *requirements.txt*, incluido en el repositorio anterior. De todas formas, las versiones de las dependencias software usadas en este TFG están descritas en las secciones 3.4.2 y 3.4.1, pudiendo usarse cualquiera de ellas. Dependiendo de las versiones CUDA disponibles en el dispositivo, puede ser necesario instalar versiones de estas dependencias distintas. Para evitar cualquier tipo de problemas de compatibilidad con dependencias de otros proyectos, se recomiendo el uso de un entorno virtual.

En cuanto al hardware requerido, lo mínimo necesario es una GPU. Dependiendo de si queremos entrenar o simplemente evaluar los modelos, la memoria mínima de la GPU requerida puede variar, siendo mucho menor para la evaluación. Asimismo, dependiendo tanto de los modelos utilizados como del tamaño de los datasets, puede aumentar o disminuir este requerimiento. Para los experimentos desarrollados en este TFG, la memoria de la GPU presentada en la sección 3.4.1 era insuficiente para realizar el entrenamiento de algunos modelos; sin embargo, sí se podía realizar su evaluación. Por otro lado, la memoria de la GPU utilizada por el equipo descrito en la sección 3.4.2 puede abordar ambas tareas sin ningún problema.

A.3. Replicación del experimento

Si se desea replicar los experimentos presentados en este TFG, se pueden usar los mismos datasets utilizados durante estos experimentos descargándolos desde la siguiente carpeta de OneDrive. Las imágenes satelitales de los datasets de validación y entrenamiento se encuentran en sus respectivas carpetas. Sus parches ya cortados y procesados están guardados en la carpeta */data*, para poder usarlos directamente sin necesidad de procesarlas previamente. También esta subida una carpeta con las máscaras de 24 clases de todas las imágenes multiespectrales utilizadas.

En esta carpeta también están guardados los modelos utilizados en los experimentos ya entrenados, de esta forma, si simplemente se quiere utilizar o revaluar los modelos no hace falta entrenarlos de nuevo.

A.4. Modificación de las pruebas

Para modificar las pruebas realizadas en este experimento, basta con editar los archivos de configuración y/o el código fuente correspondiente. Es recomendable seguir una estructura modular en el código para facilitar las modificaciones y asegurarse de que cualquier cambio no afecte negativamente a otras partes del proyecto.

A.5. Posibles problemas y soluciones

Durante la configuración y ejecución de los experimentos, pueden surgir algunos problemas comunes relacionados con las dependencias:

A.5.1. Problemas de dependencias

- **Mensaje:** `ModuleNotFoundError: No module named 'xxx'`
  - **Causa:** Algún módulo o librería no está instalada.
  - **Solución:** Asegúrese de que todas las dependencias están instaladas correctamente ejecutando:

    ```
    pip install -r requirements.txt
    ```

- **Mensaje:** `ImportError: cannot import name 'xxx'`
  - **Causa:** Puede ser debido a versiones incompatibles de librerías.
  - **Solución:** Verifique las versiones de las librerías y asegúrese de que las versiones instaladas son las correctas. Puede usar:

    ```
    pip install some_package==some_version
    ```

- **Mensaje:**
  `ERROR: Could not find a version that satisfies the requirement pytorch==0.4.1 (from versions: 0.1.2, 1.0.2)`
  - **Causa:** La versión especificada de PyTorch no está disponible para la versión de Python utilizada.
  - **Solución:** Actualice el archivo `requirements.txt` para usar una versión compatible de PyTorch. Por ejemplo, `torch==2.2.0`.

    ```
    pip install torch==2.2.0+cu118 torchvision==0.17.0+cu118
    torchaudio==2.2.0+cu118
    ```

- **Mensaje:** `ERROR: No matching distribution found for opencv3`
  - **Causa:** El paquete `opencv3` no existe en PyPI.
  - **Solución:** Reemplace `opencv3` con `opencv-python`

    ```
    pip install opencv-python
    ```

Apéndice B

Manual de usuario

Este manual está diseñado para proporcionar todas las instrucciones necesarias para la instalación, utilización y configuración del software desarrollado para este proyecto de segmentación semántica de imágenes multiespectrales. Además, se incluyen soluciones a problemas de ejecución comunes que pueden surgir durante el uso del software.

B.1. Instalación

B.1.1. Descarga del código

El código fuente del proyecto está disponible públicamente en GitHub. Para descargar el código, ejecute el siguiente comando:

```
git clone https://github.com/jecamanda/GID-semantic-segmentation.git
```

B.1.2. Instalación de Dependencias

Las dependencias de software están listadas en el archivo `requirements.txt`. Para instalar todas las dependencias, use el siguiente comando:

```
pip install -r requirements.txt
```

Se recomienda el uso de un entorno virtual para evitar conflictos con otras dependencias del sistema.

B.2. Uso del software

B.2.1. Preparación del entorno

Antes de ejecutar cualquier experimento, asegúrese de que todas las dependencias están correctamente instaladas y de que el entorno de trabajo está configurado. Active su entorno virtual si es necesario:

```
source venv/bin/activate
```

B.2.2. Archivos de configuración

Los archivos de configuración controlan los parámetros de los experimentos. Estos archivos están ubicados en la carpeta `config` y deben ser modificados según sus necesidades específicas. A continuación se explican de forma breve todos los parámetros:

- **DATASET**:
  - **root_dataset**: Directorio raíz donde se almacenan los datos del dataset.
  - **list_train**: Ruta al archivo que contiene la lista de imágenes para el entrenamiento.
  - **list_val**: Ruta al archivo que contiene la lista de imágenes para la validación.
  - **num_class**: Número de clases en el dataset.
  - **imgSizes**: Tamaños de las imágenes para el entrenamiento (aumentación de datos).
  - **imgMaxSize**: Tamaño máximo permitido para las imágenes.
  - **padding_constant**: Valor de padding constante para ajustar las dimensiones de las imágenes.
  - **segm_downsampling_rate**: Tasa de reducción de la segmentación.
  - **random_flip**: Indica si se debe aplicar un volteo aleatorio a las imágenes durante el entrenamiento.
- **MODEL**:
  - **arch_encoder**: Arquitectura del codificador utilizada en el modelo.
  - **arch_decoder**: Arquitectura del decodificador utilizada en el modelo.
  - **fc_dim**: Dimensionalidad de la capa completamente conectada.

- **TRAIN**:
  - **batch_size_per_gpu**: Tamaño del lote de datos por cada GPU.
  - **num_epoch**: Número total de épocas para el entrenamiento.
  - **start_epoch**: Época de inicio para el entrenamiento.
  - **epoch_iters**: Número de iteraciones por cada época.
  - **optim**: Tipo de optimizador utilizado en el entrenamiento.
  - **lr_encoder**: Tasa de aprendizaje para el codificador.
  - **lr_decoder**: Tasa de aprendizaje para el decodificador.
  - **lr_pow**: Exponente utilizado para el decaimiento de la tasa de aprendizaje.
  - **beta1**: Parámetro beta1 para el optimizador.
  - **weight_decay**: Tasa de decaimiento de peso para la regularización.
  - **deep_sup_scale**: Escala de la supervisión profunda.
  - **fix_bn**: Indica si se deben fijar los parámetros de Batch Normalization.
  - **workers**: Número de trabajadores para la carga de datos.
  - **disp_iter**: Número de iteraciones entre cada visualización de resultados intermedios.
  - **seed**: Semilla para la inicialización aleatoria.
- **VAL**:
  - **visualize**: Indica si se deben visualizar los resultados de la validación.
  - **checkpoint**: Archivo de checkpoint para cargar el modelo.
- **TEST**:
  - **checkpoint**: Archivo de checkpoint para cargar el modelo durante el test.
  - **result**: Directorio para almacenar los resultados del test.
- **DIR**:
  - Directorio para almacenar checkpoints específicos del modelo.

B.2.3. Preparación de datast

Si se desea usar un dataset propio a partir de imágenes satelitales completas, usar el siguiente script, modificando las rutas de las carpetas a leer.

```
python cutGID4BV2.py
```

B.2.4. Entrenamiento de modelos

Puede iniciar el entrenamiento de un modelo, seleccionando el número de GPUs ($GPUS) y el archivo de configuración ($CFG) con el siguiente comando:

```
python train.py --gpus $GPUS --cfg $CFG
```

B.2.5. Evaluación de modelos

Para evaluar un modelo previamente entrenado con el dataset de validación, utilice el script de evaluación:

```
python eval_multipro.py --gpus $GPUS --cfg $CFG
```

Para evaluar el modelo en imágenes satelitales completas de forma individual, y obtener la imagen inferida, se utiliza el script de Test y Evaluación. En el archivo de configuración se debe de indicar en el parámetro list_val el dataset con las imágenes que se quieren evaluar. Se utiliza el modelo con los pesos indicados en el checkpoint de VAL:

```
python TestAndEval.py --gpus $GPUS --cfg $CFG
```

B.3. Solución de Problemas

Aquí se listan algunos problemas comunes, sus causas y posibles soluciones:

- **RuntimeError: CUDA error: CUDA-capable device(s) is/are busy or unavailable Causa**: Esto puede ocurrir si no hay GPU disponible o si la GPU está siendo utilizada por otro proceso.

  Solución: Verifique que su GPU esté disponible y no esté ocupada por otros procesos. Puede usar el comando `nvidia-smi` para revisar el estado de la GPU. Si es necesario, finalice otros procesos que estén usando la GPU.

- **RuntimeError: CUDA out of memory Causa**: Este error ocurre cuando la memoria de la GPU no es suficiente para cargar el modelo y los datos.

  Solución: Intente reducir el tamaño del lote (`batch_size`) o utilice una GPU con mayor capacidad de memoria. También puede optimizar el uso

de memoria utilizando técnicas como el uso de modelos de menor tamaño o la reducción de la resolución de las imágenes de entrada.

Con estas instrucciones, debería poder instalar, utilizar y solucionar problemas comunes del software desarrollado para este proyecto de segmentación semántica de imágenes multiespectrales.

Bibliografía

[1] Eleni Aloupogianni, Masahiro Ishikawa, Naoki Kobayashi, Takashi Obi, "Hyperspectral and multispectral image processing for gross-level tumor detection in skin lesions: a systematic review", J. Biomed. Opt. 27(6) 060901, junio 2022, doi: https://doi.org/10.1117/1.JBO.27.6.060901.

[2] Jiang, B., An, X., Xu, S. et al. "Intelligent Image Semantic Segmentation: A Review Through Deep Learning Techniques for Remote Sensing Image Analysis", J Indian Soc Remote Sens 51, 1865–1878, (enero 2022). doi:https://doi.org/10.1007/s12524-022-01496-w .

[3] Dhruv Matani, "Efficient Image Segmentation Using PyTorch: Part 1", Towards Data Science, junio 2023, URL: https://towardsdatascience.com/efficient-image-segmentation-using-pytorch-part-1-89e8297a0923.

[4] Xin-Yi Tong, Gui-Song Xia, Qikai Lu, Huanfeng Shen, Shengyang Li, Shucheng You, Liangpei Zhang, "Land-Cover Classification with High-Resolution Remote Sensing Images Using Transferable Deep Models", Remote Sensing of Environment 237, 2020, URL: https://x-ytong.github.io/project/GID.html.

[5] Osorio, Kavir, Andrés Puerto, Cesar Pedraza, David Jamaica, and Leonardo Rodríguez. 2020"A Deep Learning Approach for Weed Detection in Lettuce Crops Using Multispectral Image", AgriEngineering 2, no. 3: 471-488, junio 2020, doi:https://doi.org/10.3390/agriengineering2030032

[6] S. Yu, S. Jia e C. Xu, "Convolutional neural networks for hyperspectral image classification", Neurocomputing, vol. 219, pp. 88–98, enero 2017, doi:https://doi.org/10.1016/j.neucom.2016.09.010.

[7] Convolutional Neural Network (CNN): A Complete Guide, [fecha de consulta: 15 de junio 2024] URL: https://learnopencv.com/understanding-convolutional-neural-networks-cnn/.

[8] Jolliffe, I. T., "Principal Component Analysis", (2nd ed.). Springer, Capítulo 13, "Principal Component Analysis for Special Types of Data", 2002, doi:https://doi.org/10.1007/0-387-22440-8_13.

[9] Aapo Hyvärinen, "Independent Component Analysi", University of Helsinki, 2001, URL: https://www.ma.imperial.ac.uk/~nsjones/TalkSlides/HyvarinenSlides.pdf.

[10] G. Hughes, "On the mean accuracy of statistical pattern recognizers,ïn IEEE Transactions on Information Theory", vol. 14, no. 1, pp. 55-63, Enero 1968, doi: https://www.doi.org/10.1109/TIT.1968.1054102.

[11] Chen, L.; Wei, Z.; Xu, Y. "A Lightweight Spectral–Spatial Feature Extraction and Fusion Network for Hyperspectral Image Classification", Remote Sens. 12, 1395, 2020, doi:https://doi.org/10.3390/rs12091395.

[12] Kong, F.; Hu, K.; Li, Y.; Li, D.; Zhao, S. "Spectral–Spatial Feature Partitioned Extraction Based on CNN for Multispectral Image Compression". Remote Sens. 2021, 13, 9. doi: https://doi.org/10.3390/rs12091395

[13] Z. Zhong, J. Li, L. Ma, H. Jiang and H. Zhao, "Deep residual networks for hyperspectral image classification," 2017 IEEE International Geoscience and Remote Sensing Symposium (IGARSS), Fort Worth, TX, USA, pp. 1824-1827, 2017, doi:https://www.doi.org/10.1109/IGARSS.2017.8127330.

[14] X. -Y. Tong, Q. Lu, G. -S. Xia and L. Zhang, "Large-Scale Land Cover Classification in Gaofen-2 Satellite Imagery", IGARSS 2018 - 2018 IEEE International Geoscience and Remote Sensing Symposium, Valencia, España, pp. 3599-3602, 2018, doi:https://www.doi.org/10.1109/IGARSS.2018.8518389.

[15] Jonathan Long, Evan Shelhamer, Trevor Darrell, "Fully Convolutional Networks for Semantic Segmentation", 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 3431-3440, 2015, URL:https://openaccess.thecvf.com/content_cvpr_2015/html/Long_Fully_Convolutional_Networks_2015_CVPR_paper.html.

[16] He C, Li S, Xiong D, Fang P, Liao M. "Remote Sensing Image Semantic Segmentation Based on Edge Information Guidance", Remote Sensing. 12(9):1501, 2020, doi:https://doi.org/10.3390/rs12091501.

[17] V. Badrinarayanan, A. Kendall and R. Cipolla, "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation", in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 12, pp. 2481-2495, 1 diciembre 2017, doi: https://doi.org/10.1109/TPAMI.2016.2644615.

[18] O., Fischer, P., Brox, T. (2015). "U-Net: Convolutional Networks for Biomedical Image Segmentation", In: Navab, N., Hornegger, J., Wells, W., Frangi, A. (eds) Medical Image Computing and Computer-Assisted Intervention –

MICCAI 2015, Lecture Notes in Computer Science(), vol 9351. Springer, Cham, 2015, doi: https://doi.org/10.1007/978-3-319-24574-4_28.

[19] Li, X., Xu, F., Lyu, X., Gao, H., Tong, Y., Cai, S., Liu, D., "Dual attention deep fusion semantic segmentation networks of large-scale satellite remote-sensing images", International Journal of Remote Sensing, 42(9), 3583–3610, 2021, doi:https://doi.org/10.1080/01431161.2021.1876272.

[20] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, Antonio Torralba, "Scene Parsing through ADE20K Dataset", 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, doi:https://doi.org/10.1109/CVPR.2017.544.

[21] "CSAILVision/semantic-segmentation-pytorch" Repositorio de GitHub, [fecha de consulta: 22 de junio 2024] URL: https://github.com/CSAILVision/semantic-segmentation-pytorch.

[22] Xin-Yi Tong, Gui-Song Xia, Xiao Xiang Zhu, "Enabling Country-Scale Land Cover Mapping with Meter-Resolution Satellite Imagery", ISPRS Journal of Photogrammetry and Remote Sensing, v. 196, pp. 178-196, URL: https://x-ytong.github.io/project/Five-Billion-Pixels.html.

[23] Canonical, Ubuntu, [fecha de consulta: 22 de junio 2024]. URL:https://ubuntu.com/.

[24] NVIDIA DEVELOPER, CUDA Toolkit 8.0 - Feb 2017, [fecha de consulta: 22 de junio 2024] URL: https://developer.nvidia.com/.

[25] Python, [fecha de consulta: 22 de junio 2024] URL: https://www.python.org/.

[26] Pytorch, [fecha de consulta: 22 de junio 2024] URL: https://pytorch.org/get-started/previous-versions/.

[27] Numpy, [fecha de consulta: 22 de junio 2024] URL: https://numpy.org/.

[28] Scipy, [fecha de consulta: 22 de junio 2024] URL: https://scipy.org/.

[29] Opencv, [fecha de consulta: 22 de junio 2024] URL: https://opencv.org/.

[30] Yacs, [fecha de consulta: 22 de junio 2024] URL: https://pypi.org/project/yacs/.

[31] Tqdm, [fecha de consulta: 22 de junio 2024] URL: https://pypi.org/project/tqdm/.

[32] AlmaLinux, [fecha de consulta: 22 de junio 2024] URL: https://almalinux.org/es/.

[33] DataParallel, [fecha de consulta: 22 de junio 2024] URL: https://pytorch.org/docs/stable/generated/torch.nn.DataParallel.html.

[34] Familia GeForce RTX 3060, [fecha de consulta: 28 de junio 2024] URL: https://www.nvidia.com/es-es/geforce/graphics-cards/30-series/rtx-3060-3060ti/.

[35] Procesador Intel Core i7-12700H, [fecha de consulta: 22 de junio 2024] URL: https://www.intel.la/content/www/xl/es/products/sku/132228/intel-core-i712700h-processor-24m-cache-up-to-4-70-ghz/specifications.html.

[36] GDAL documentation, URL: https://gdal.org/index.html.

[37] High Performance Computing (HPC) cluster ctcomp3 User Guide, [fecha de consulta: 22 de junio 2024] URL: https://wiki.citius.gal/en:centro:servizos:hpc.

[38] Tesla V100S-PCIE-32GB, [fecha de consulta: 22 de junio 2024] URL: https://www.nvidia.com/es-es/data-center/tesla-v100/.

[39] Intel(R) Xeon(R) Gold 5220R CPU @ 2.20GHz, [fecha de consulta: 28 de junio 2024] URL: https://www.intel.la/content/www/xl/es/products/sku/199354/intel-xeon-gold-5220r-processor-35-75m-cache-2-20-ghz/specifications.html.

[40] FinisTerrae III User Guide, [fecha de consulta: 28 de junio 2024] URL: https://cesga-docs.gitlab.io/ft3-user-guide/index.html.

[41] Descenso de gradiente estocástico, [fecha de consulta: 28 de junio de 2024] URL: https://pytorch.org/docs/stable/generated/torch.optim.SGD.html.

[42] GTiff – GeoTIFF File Format, [fecha de consulta: 28 de junio de 2024] URL: https://gdal.org/drivers/raster/gtiff.html.

[43] PIL, [fecha de consulta: 28 de junio 2024] URL: https://pypi.org/project/pillow/.

[44] Liwei Wang, Chen-Yu Lee, Zhuowen Tu, "Training Deeper Convolutional Networks with Deep Supervision", 2015, arXiv:https://doi.org/10.48550/arXiv.1505.02496.

[45] Mark Sandler Andrew Howard Menglong Zhu Andrey Zhmoginov Liang-Chieh Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks", 2018, arXiv: https://arxiv.org/pdf/1801.04381.

[46] K. Sun1, Y. Zhao B. Jiang, T. Cheng, B. Xiao, D. Liu, Y. Mu, X. Wang, W. Liu, J. Wang, "High-Resolution Representations for Labeling Pixels and Regions", 2019, arXiv:https://doi.org/10.48550/arXiv.1904.04514.

[47] Bo Chen, Jiewei Cao, Alvaro Parra, Tat-Jun Chin, "Satellite Pose Estimation with Deep Landmark Regression and Nonlinear Pose Refinement", 2019, URL: https://www.researchgate.net/publication/335564768.

[48] H. Zhao, J. Shi, X. Qi, X. Wang, J. Jia, "Pyramid Scene Parsing Network", 2017, arXiv:https://doi.org/10.48550/arXiv.1612.01105.

[49] T. Xiao, Y. Liu, B. Zhou, Y. Jiang, J. Sun, "Unified Perceptual Parsing for Scene Understanding", Accepted to European Conference on Computer Vision (ECCV) 2018, 2018, arXiv:https://doi.org/10.48550/arXiv.1807.10221.

[50] NLLLoss, [fecha de consulta: 27 de junio 2024] URL: https://pytorch.org/docs/stable/generated/torch.nn.NLLLoss.html.