Universidad Rey Juan Carlos

ESCUELA DE MÁSTERES OFICIALES

MÁSTER UNIVERSITARIO EN VISIÓN ARTIFICIAL

CURSO ACADÉMICO 2024/2025

TRABAJO FIN DE MÁSTER

SEGMENTACIÓN SEMÁNTICA PARA PERCEPCIÓN VISUAL EN ROBOTS DE CONDUCCIÓN AUTÓNOMA EN ENTORNOS NO ESTRUCTURADOS

Autor: Rebeca Villarraso

Directores: José María Cañas Plaza

Sergio Paniego Blanco

Agradecimientos

A mis tutores, José María y Sergio, por su tiempo, paciencia y acompañamiento durante todo el desarrollo del trabajo. Agradecer también las correcciones y consejos que me brindaron para redactar el presente documento con la calidad y profesionalidad suficiente.

A David, por la ayuda brindada en el proceso de entrenamiento de algunos modelos y sus buenos consejos.

A mi familia y amigos por su paciencia.

Y a mi gata *Alice* por hacerme repetir algún que otro entrenamiento.

Resumen

Los recientes avances en conducción autónoma y robots móviles han incrementado significativamente el interés en la navegación en exteriores. En numerosas aplicaciones, como la minería, la asistencia en caso de desastres, la robótica agrícola o la topografía ambiental, el robot debe navegar en terrenos irregulares o escenarios que carecen de una estructura clara o características de navegación bien identificadas.

Un aspecto clave y desafiante en el desarrollo de capacidades de percepción visual para navegación en entornos no estructurados, es encontrar regiones seguras y navegables que puedan ser utilizadas por un robot autónomo. Por ejemplo, terrenos como hormigón o asfalto son lisos y muy transitables, mientras que un charco con agua podría ser transitable, un lago o un río no lo serían para un robot con ruedas.

En el presente Trabajo Final de Máster se realizan diferentes experimentos basados en el aprendizaje automático en visión artificial para identificar las regiones seguras y navegables en entornos no estructurados a partir de imágenes RGB. Este enfoque propone dotar a un robot autónomo con percepción visual a partir de la identificación de las áreas del entorno según sus niveles de navegabilidad mediante segmentación semántica de grano grueso. Durante el desarrollo del presente estudio, se analizan los modelos de aprendizaje profundo para segmentación semántica y se entrenan dichos modelos con conjuntos de imágenes seleccionados que son representativos de entornos no estructurados a los que puede enfrentarse el robot. A partir de los resultados obtenidos en los diferentes experimentos, se analiza la viabilidad de las estrategias utilizadas para su uso como percepción visual para un robot de conducción autónoma en entornos no estructurados.

Abstract

Recent advances in autonomous driving and mobile robots have significantly increased interest in outdoor navigation. In numerous applications, such as mining, disaster relief, agricultural robotics, or environmental surveying, the robot must navigate in rough terrain or scenarios that lack clear structure or well-identified navigational features.

A key and challenging aspect in developing visual perception capabilities for navigation in unstructured environments is to find safe and navigable regions that can be used by an autonomous robot. For example, terrain such as concrete or asphalt is smooth and easily traversable, while a puddle of water might be traversable, a lake or river would not be for a wheeled robot.

In this Master's thesis, different experiments based on machine learning in computer vision are carried out to identify safe and navigable regions in unstructured environments from RGB images. This approach proposes to provide an autonomous robot with visual perception based on the identification of the areas of the environment according to their navigability levels by means of coarse-grained semantic segmentation. During the development of the present study, deep learning models for semantic segmentation are analyzed and these models are trained with selected sets of images that are representative of unstructured environments that the robot may face. From the results obtained in the different experiments, the feasibility of the strategies used for use as visual perception for an autonomous driving robot in unstructured environments is analyzed.

Índice general

Capítulo 1

Introducción

En el presente Trabajo Final de Máster (TFM) se pretende dotar a un robot de *percepción visual en entornos no estructurados para conducción autónoma*. En el contexto visual, se entiende como percepción, a la interpretación de imágenes que describen el entorno en el que se sitúa el robot en el tiempo. Siendo en el caso que se estudia, un entorno no estructurado, en el que *a priori*, se desconocen los objetos o elementos de la escena y en el que estos pueden ser dinámicos.

Esto implica dos desafíos importantes. Por una parte, el coste computacional elevado que implica el análisis de cada imagen para la interpretación del entorno en tiempo real o semi-real. En un entorno estructurado **(Figura 1.1(a))**, aunque el coste computacional también pueda ser elevado, está más limitado a la interpretación de los elementos de la imagen que a priori son conocidos o predecibles. Por ejemplo, en una ciudad, existen las carreteras, calles y aceras, que están delimitados por líneas o bordes, fácilmente detectables. Pero en entornos no estructurados **(Figura 1.1(b))**, no se conocen los elementos que componen la imagen a interpretar, estos pueden ser móviles y su forma es desconocida.

Otro desafío importante, es la escasa disponibilidad de conjuntos de imágenes que nos permitan entrenar un modelo de Aprendizaje Profundo *(Deep Learning)* en dichos entornos, y que además, tengan la variabilidad y cantidad suficiente de elementos que permitan representar el entorno real en el que se situará el robot.

El auge de robots móviles y los avances en conducción autónoma han aumentado significativamente el interés de la navegación al aire libre, desde su aplicación en drones, a robots terrestres para aplicaciones topográficas, medioambientales, análisis de cultivos, salvamento, catástrofes, entre otras aplicaciones. El sistema debe ser capaz de navegar de forma segura y fiable en entornos sin una estructura claramente definida, enfrentándose a desafíos como la inestabilidad del terreno, desniveles, pendientes, obstáculos fijos y móviles, presencia de agua, entre otros.

(a) Entorno estructurado.

(b) Entorno no estructurado

Figura 1.1: Tipos de entorno

Una de las características clave para que un robot desarrolle capacidades de percepción visual destinadas a la conducción autónoma en terrenos no estructurados, reside en que pueda identificar con suficiente antelación, las zonas transitables para sus características de diseño. Por ejemplo, no identificar una caída libre, puede conllevar a que el robot se dañe o a su pérdida. Cabe destacar, que las características de diseño del robot son importantes. Si está equipado con ruedas, navegar por nieve puede ser más dificultoso que si dispone de orugas, o si no se detecta como no navegable un charco con agua, puede dañarse si no está protegido. Por lo tanto, uno de los grandes desafíos en entornos no estructurados, es que se pueda interpretar el mayor número de elementos relevantes para su navegación con suficiente antelación, de forma segura y fiable.

1.1. Conducción autónoma

En la actualidad, el mayor desafío es conseguir un nivel de autonomía total en el que no se requiera intervención humana. Si nos basamos en los sistemas de conducción autónoma en vehículos, que en la actualidad están bastante avanzados en la navegación en entornos estructurados, de acuerdo al estándar SAE J3016 [1], hay 6 niveles de autonomía que puede tener un sistema de conducción autónoma. Los niveles de 0 a 2, ofrecen diferentes ayudas al usuario, mientras que en los niveles de 3 a 5, el usuario ya no es el conductor:

- Nivel 0: no hay ningún tipo de automatización. Sistema convencional en el que el humano tiene el control total del vehículo.
- Nivel 1 o de conducción asistida: incluye algún sistema de asistencia a la conducción y requiere intervención humana constante.
- Nivel 2 o autonomía parcial: dispone de automatización parcial a la conducción, algunas tareas son automáticas pero requiere intervención constante del usuario.

- Nivel 3 o de autonomía condicional: el usuario interviene sólo si el sistema autónomo lo solicita o si ocurre algún fallo.
- Nivel 4 o de alta autonomía: no es necesaria la presencia del conductor, ya que cuenta con un sistema de respaldo que actúa en caso de fallo en el sistema principal o situaciones de riesgo mínimo.
- Nivel 5 o de autonomía total: vehículo completamente autónomo, sin condiciones limitantes específicas para su funcionamiento.

Un vehículo Tesla [2], podría situarse entre un nivel SAE 3-4, en el que se ofrece asistencia al conductor y además hay automatización parcial de la navegación. Otro ejemplo, es el vehículo Waymo [3] que se situaría en un nivel 3-4 sin conductor.

Para poder realizar estas tareas, el sistema necesita por un lado, la percepción del entorno y por otro, actuar en consecuencia. Para ello, es necesario un gran equipamiento de sensores como: sensores de movimiento, cámaras, GPS, LIDAR, etc., actuadores que permitan el movimiento del vehículo y un software que gestione ambos y tome decisiones.

Trasladando estos requisitos al objetivo que se pretende conseguir y dotar a un robot de navegación autónoma en entornos no estructurados, este trabajo se centra en dotar al robot de la percepción del entorno exclusivamente con imágenes (percepción visual con visión artificial).

1.2. Percepción visual

En el contexto de Aprendizaje Profundo en visión artificial con Redes Neuronales Convolucionales (CNN), la interpretación de imágenes para percepción visual, se puede enfocar de distintas formas. Por ejemplo, clasificación y localización, detección de objetos o segmentación a diferentes niveles. La *Figura 1.2* muestra las diferencias básicas entre dichas tareas. A continuación, se resume el enfoque de cada una de estas técnicas.

1.2.1. Clasificación + localización

Esta técnica trata de clasificar y localizar un objeto dentro de una imagen.

- **Clasificación**: identifica qué objeto está presente en una imagen. Por ejemplo, puede ser la imagen de un perro, un gato, etc. La red neuronal solo tiene que asignar una etiqueta de clase a la imagen completa.
- **Localización**: no solo se tiene que identificar qué objeto está presente, sino también determinar en qué parte de la imagen se encuentra. Usa una caja delimitadora *(bounding*

box) que marca la ubicación del objeto. El modelo predice las coordenadas de la caja, representadas por el centro de la caja y sus dimensiones.

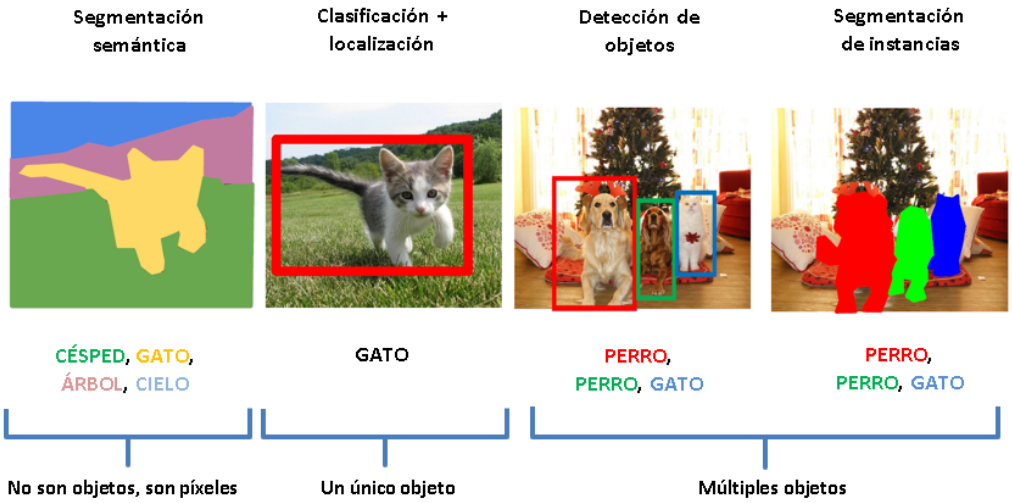


Figura 1.2: Ejemplo de técnicas de percepción visual.

Un ejemplo de este enfoque es el modelo *Fast R-CNN* [4] que realiza clasificación y localización de objetos de manera simultánea.

1.2.2. Detección de objetos

La detección de objetos es similar al anterior pero con mayor complejidad, ya que, es multiclase y multiobjeto, siendo capaz de clasificar y localizar las diferentes instancias de objetos del mismo tipo con cajas delimitadoras. Algunos modelos para detección de objetos son:

- **YOLO**. *You Only Look Once* [5]: una red neural que predice las cajas delimitadoras y las clases de los objetos en una sola pasada de la imagen, permitiendo la detección de objetos en tiempo real.
- **R-CNN**. *Region-based Convolutional Neural Networks* [6]: este modelo primero genera propuestas de regiones de la imagen y luego las clasifica utilizando una red neuronal convolucional (CNN).
- **SSD**. *Single Shot MultiBox Detector* [7]: similar a YOLO, pero con un enfoque diferente en cómo realiza la detección de objetos en imágenes de diferentes escalas.

La detección de objetos es útil en aplicaciones como la vigilancia de seguridad, la conducción autónoma (donde los vehículos necesitan detectar otros vehículos, peatones, señales de tránsito, etc.) y la automatización de procesos industriales.

1.2.3. Segmentación de imágenes

Existen diferentes enfoques de segmentación de imágenes con aprendizaje profundo: *segmentación semántica, segmentación de instancias* y *segmentación panóptica*, cada una de las cuales se adapta mejor al objetivo al que vayan destinadas. Teniendo en cuenta que la segmentación de una imagen, es encontrar las zonas o regiones diferentes que pertenecen a diferentes objetos o clases, la forma en que se realiza y el nivel de identificación de las regiones es lo que caracteriza a cada una de estas técnicas.

Segmentación semántica a nivel de píxel

La segmentación semántica a nivel de píxel trata de asignar a cada píxel de una imagen una etiqueta preestablecida **(Figura 1.3)**.

Figura 1.3: Segmentación semántica a nivel de píxel.

Los pixeles asignados a cada etiqueta **(Figura 1.4)**, forman regiones que permiten identificar objetos o zonas de interés (interpretación semántica), en este caso: cielo, árboles, gato y césped.

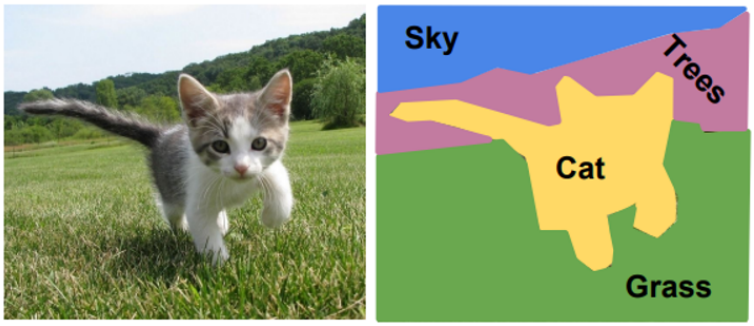


Figura 1.4: Segmentación semántica. Forma regiones.

Si en una imagen existen diferentes instancias de una misma etiqueta, la segmentación a nivel de píxel, no es capaz de distinguirlos. Es decir, interesa únicamente identificar las regiones

de forma categórica o semántica, pero no identificar dos o más elementos de la misma categoría como se muestra en la *Figura 1.5*.

Figura 1.5: Segmentación semántica. No distingue instancias.

En este caso, se identifican las regiones: vaca, árboles, cielo y césped, pero existen dos vacas, ambas pertenecen a la misma categoría: vaca. No se distingue entre vaca 1 y vaca 2, ya que éste no es el objetivo.

Para segmentación semántica a nivel de pixel con aprendizaje profundo se necesitarán máscaras con el identificador para cada región. En el caso de la *Figura 1.5* (vacas), se tendrían 4 identificadores o etiquetas: árboles, cielo, vaca y césped.

- **Ventajas de la segmentación semántica:**
  - Proporciona un análisis global de la escena.
  - Menor coste computacional.
  - Diferenciación de zonas que están relacionadas.
- **Ejemplos de uso**: diferenciación de zonas transitables y no transitables en conducción autónoma.

Segmentación a nivel de instancias

La segmentación de instancias aumenta la complejidad respecto a la segmentación a nivel de píxel, no sólo identifica regiones de la imagen de una etiqueta, sino que además, identifica las instancias diferentes. Se aprecia en la *Figura 1.6* que permite distinguir el número diferente de vacas: vaca 1, vaca 2, etc.

- **Ventajas de la segmentación de instancias:**
  - Identificación más precisa de los elementos o regiones de una imagen.
  - Identificación de múltiples objetos de la misma etiqueta.
- **Ejemplos de uso:** conteo de personas que transitan por una determinada zona.

Figura 1.6: Segmentación de instancias.

Segmentación panóptica

La segmentación panóptica combina la segmentación a nivel de pixel y a nivel de instancias, asignando a cada pixel de la imagen dos etiquetas: una etiqueta semántica y otra de identificación de instancia. Los identificadores de instancias distinguen las diferentes instancias que pertenecen a un mismo tipo de objeto (interpretación semántica). A diferencia de la segmentación de instancias, la segmentación panóptica asigna una etiqueta distinta a cada píxel correspondiente a una instancia individual para evitar interpretaciones erróneas de la información, por lo que, se necesitarán máscaras con un identificador diferente para cada instancia.

Para la *Figura 1.7*, se tendrían por ejemplo, tres clases semánticas: suelo, fondo y animales, con una máscara con 6 identificadores de instancias: un identificador "césped" para la clase semántica "suelo", un identificador "fondo" para la clase semántica "fondo" y tres identificadores: "perro", "oveja 1" y "oveja 2", para la clase semántica "animales".

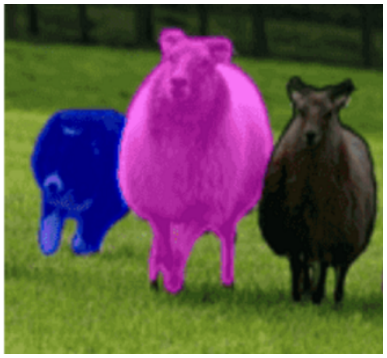


Figura 1.7: Segmentación panóptica.

- **Ventajas de la Segmentación panóptica:** interpretación e identificación más real de los diferentes elementos de una escena.

- **Ejemplos de uso**: interacción de coche autónomo con peatón concreto (por ejemplo, evitar atropello de coche autónomo a peatón concreto en paso de cebra).

1.3. Percepción visual en entornos no estructurados

Los aspectos clave a tener en cuenta en el desarrollo de la percepción visual en entornos no estructurados son los siguientes:

- El coste computacional debe ser lo suficientemente ágil para permitir la capacidad de actuar al robot de forma segura y evitar los obstáculos o zonas no navegables, sin que esto comprometa la capacidad de segmentación.
- Capacidad de definir los límites de las regiones de forma fiable, ya que pueden existir regiones muy similares o superpuestas, que no pertenezcan a la misma región. Por ejemplo, un charco con poca agua puede ser transitable, no siendo así un lago.
- Algunas regiones del terreno de una determinada etiqueta, pueden ocupar pocos píxeles en una escena y muchos en otra, es decir, una misma etiqueta que representa la zona de una imagen puede estar a diferentes escalas, lo que puede conllevar a clasificaciones erróneas. Un ejemplo, es el diferente tamaño en la imagen de una persona en primer plano o en la lejanía.
- Algunas regiones o etiquetas en la escena que ocupen pocos píxeles, requieren una segmentación lo suficientemente fina, para que esta región se clasifique correctamente y no se agrupe con otra etiqueta con mayor cantidad de píxeles en la imagen.
- Para el enfoque de segmentación semántica con aprendizaje profundo se requerirá un conjunto de imágenes y máscaras con las etiquetas lo suficientemente grande para que permita el aprendizaje de las regiones o etiquetas en las imágenes.
- Posiblemente el conjunto de datos esté desbalanceado, es decir, contenga muchas etiquetas de un tipo y pocas de otro, lo que lleve a un rendimiento inferior en algunas clases respecto a otras.

Teniendo en cuenta las características anteriores y respetando un compromiso compartido entre coste computacional y segmentación fina, se opta por la técnica de *segmentación semántica*, en el que la identificación del entorno no es tan burda como a nivel de transitable o no transitable (dos etiquetas), ni tan fina como a nivel de identificar varios tipos de objetos e instancias como en la detección de objetos y en la segmentación de instancias. La hipótesis de partida, es que la segmentación semántica a nivel de pixel, se cree suficiente para abordar el problema que nos ocupa: *percepción visual para navegación autónoma en entornos no estructurados* y que se pretende valorar en el presente trabajo, ya que no es necesario interactuar con diferentes instancias de los elementos de la imagen.

1.4. Objetivos

El objetivo general del presente TFM se centra en evaluar la viabilidad de las técnicas de segmentación semántica para percepción visual con Redes Neuronales Convolucionales (CNNs) para robots de conducción autónoma en entornos no estructurados. A continuación se detallan los objetivos específicos:

- **Objetivo 1. Modelos**: selección de modelos basados en CNNs para segmentación semántica visual y estudio de sus características más relevantes.
- **Objetivo 2. Conjuntos de imágenes**: selección de conjuntos de imágenes en entornos no estructurados para segmentación semántica y estudio de sus propiedades más importantes.
- **Objetivo 3. Experimentos**: entrenamiento de modelos con los conjuntos de imágenes seleccionados para segmentación semántica visual en entornos no estructurados.
- **Objetivo 4. Análisis**: valoración de la viabilidad de los modelos y los conjuntos de imágenes para dotar de percepción visual a un robot para conducción autónoma a partir de los resultados obtenidos.

1.5. Estructura del documento

En el presente capítulo se realiza una introducción para describir las necesidades y características del problema a resolver. Se enumeran los objetivos a alcanzar y se hace referencia al repositorio de imágenes y código.

Para poder seleccionar y explotar diferentes modelos de segmentación semántica basados en CNN, así como los conjuntos de imágenes en entornos no estructurados, en el *Capítulo 2* se detalla el Estado del Arte. Aquí, se recogen los modelos de segmentación semántica visual basados en redes neuronales convolucionales que se han considerado interesantes. Se indican de forma general, sus características importantes y las métricas reportadas por sus autores. Estos modelos sentarán las bases para la selección de algunas redes de segmentación que se utilizarán en el presente estudio.

Las especificaciones técnicas del equipo utilizado, el lenguaje de programación, librerías y demás detalles de *software* y *hardware*, se describen en el *Capítulo 3*.

Se realiza una pequeña recolección de conjuntos de imágenes no estructurados *Capítulo 4* para su selección y posterior entrenamiento con los modelos seleccionados. Se describe para cada uno de ellos, las condiciones de adquisición, las plataformas en las que va montada la cámara, los entornos que atraviesa el vehículo donde está instalada la cámara. Se incluye información

sobre las etiquetas anotadas a nivel de píxel, número de imágenes, división de los conjuntos de imágenes, etc.

Este estudio no sólo se ha limitado al uso de modelos de redes de segmentación semántica visual de terceros, se ha realizado un estudio de sus características destacables para su comprensión y qué estrategias más importantes utilizan cada una de las redes para la posible solución de los problemas intrínsecos de la propia segmentación y los propios que se encuentran en los entornos no estructurados. Esta comprensión, ha permitido seleccionar aquellos de mayor relevancia e interés para ser entrenados. Los detalles de su arquitectura y características clave se recogen en el *Capítulo 5*.

La descripción de las métricas de evaluación de los diferentes modelos, las condiciones de entrenamiento y la descripción general del algoritmo utilizado se describe en el *Capítulo 6*.

Una vez planteado el problema a resolver, definidos los modelos y conjuntos de imágenes a utilizar así como el *hardware* y *software* para realizar los entrenamientos, en el *Capítulo 7* se describen los experimentos planteados, el procesado y distribución de las imágenes seleccionadas y el análisis y comentarios de los resultados obtenidos en los diferentes experimentos

Por último, se presentan las conclusiones finales basadas en los resultados obtenidos en los experimentos y centrados en la consecución de los objetivos planteados. También se comentan las posibles líneas futuras para ampliación y/o continuación de la investigación presentada *(Capítulo 8)*.

1.6. Repositorios

- Github: almacenamiento de código.
    - Disponible en https://github.com/RoboticsLabURJC/2024-tfm-rebeca-villaraso.
- HuggingFace: pesos de entrenamiento y conjuntos de imágenes.
    - Disponible en https://huggingface.co/GAIA-URJC.

Capítulo 2

Estado del Arte

La percepción visual para robots autónomos capaces de operar en entornos no estructurados requiere sistemas avanzados de visión artificial. La capacidad para interpretar imágenes y extraer información relevante del entorno es clave para garantizar la navegación, la toma de decisiones y la interacción con el medio. En este contexto, se divide este capítulo en dos bloques: el primero parte contiene un análisis del estado del arte de técnicas de aprendizaje profundo basadas en Redes Neuronales Convolucionales para segmentación semántica a nivel de píxel, y el segundo más específico, se centra en el estado del arte basado en CNN para conducción autónoma. Ambos sentarán las bases para la selección de los modelos y las estrategias de entrenamiento utilizados en este estudio.

2.1. Segmentación Semántica con CNNs

Las CNN clásicas utilizadas en la clasificación de imágenes, no son efectivas en la segmentación semántica debido a que aplican operaciones de agrupamiento y submuestreo de los datos de entrada para clasificar la imagen. Esto provoca una pérdida de resolución e información local y global en la imagen de entrada. Para la tarea de clasificación, estas pérdidas no son un gran problema, ya que, para la mayoría de situaciones no se requieren detalles muy finos. Es decir, es suficiente dar una etiqueta global como salida para la clasificación. Sin embargo, para la tarea de segmentación semántica, se requiere una predicción más detallada de una etiqueta de clase para cada píxel. Por lo tanto, estas pérdidas afectan negativamente al rendimiento. Para resolver éste y otros problemas, se han desarrollado diferentes arquitecturas de CNN que recuperan la pérdida de información espacial, global y local. Los estudios en esta área han avanzado al incorporar información local obtenida de la CNN e información global obtenida de partes más profundas de la red.

Diferentes estudios [8, 9, 10, 11, 12] utilizan algoritmos de aprendizaje profundo con Redes Neuronales Convolucionales para la tarea de segmentación semántica de imágenes, extendiendo un problema de clasificación a la tarea de segmentación.

Los modelos de segmentación semántica basados en CNN generalmente utilizan una red convolucional base (*backbone*) para extraer características de las imágenes de entrada que está conectada a un cabezal de segmentación. En el caso de la segmentación a nivel de píxel asigna una etiqueta a cada píxel, formando las regiones de segmentación. Estas predicciones de regiones, se comparan con las máscaras previamente etiquetadas utilizadas como patrón o referencia y se calcula el error cometido en la predicción para ir optimizando las predicciones. Se suelen utilizar *backbones* de las familias: *ResNet* [13], *VGGNet* [14] o *EfficientNet* [15] por citar algunas, cada una con sus propias ventajas en términos de precisión y eficiencia computacional.

En el interesante trabajo [8] se propuso una Red Totalmente Convolucional (FCN) para clasificación densa o segmentación semántica. La FCN contiene dos bloques principales: el codificador y el decodificador. El codificador se utiliza como una CNN general para extraer características. El decodificador a partir de convoluciones traspuestas o dilatadas sobremuestrea los mapas de características a la resolución espacial original de la imagen de entrada, obteniendo como salida una imagen con las regiones que representan los elementos u objetos de la imagen. La clave de este trabajo reside en convertir las capas completamente conectadas en capas totalmente convolucionales que generan la imagen segmentada **(Figura 2.1)**. Para ello combinaron diferentes arquitecturas de CNN como *AlexNet* [16], *VGGNet* [14] o *GoogLeNet* [17] como redes troncales modificadas, que posteriormente entrenaron de extremo a extremo.

En [8] reportan unos resultados de $62,2\,\%$ mIoU (métrica de intersección sobre la unión) con el conjunto de datos *PASCAL VOC-2012* [18] y aunque no parecen muy buenos resultados, mejoraron trabajos anteriores y sentaron las bases para estudios posteriores. Por ejemplo, en [19] proponen un modelo de segmentación semántica basada en una red troncal *VGG16* modificada, aplicando transferencia de aprendizaje con los pesos del conjunto de datos *Imagenet* [20] en lugar de entrenar de extremo a extremo. Este modelo se aplicó en varios conjuntos de imágenes para clasificar árboles, obteniendo una exactitud a nivel de pixel entre el $87\,\%$-$96,5\,\%$ dependiendo del tipo de árbol.

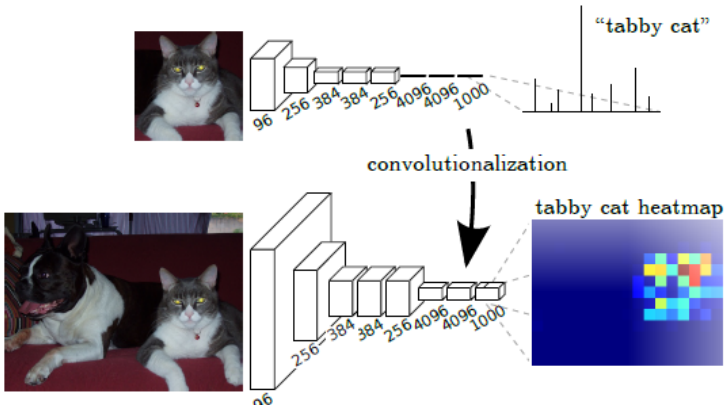


Figura 2.1: Transformación de capas completamente conectadas en capas completamente convolucionales que permiten que una red de clasificación genere una imagen segmentada añadiendo capas y una pérdida espacial.

En [21] proponen la red *U-Net* **(Figura 2.2)** en forma de U entrenada de extremo a extremo para clasificar imágenes médicas. Se basa en la estrategia del uso intensivo de la ampliación de datos de forma eficiente cuando el conjunto de datos es pequeño, como suele ocurrir en entornos médicos. Reportaron mejoras significativas respecto a anteriores estudios con $92,03\,\%$ mIoU y $77,6\,\%$ mIoU para los conjuntos de datos *PhC-U373* [22] y *DIC-HeLa* [23] en la tarea de segmentación de células en imágenes de microscopía.

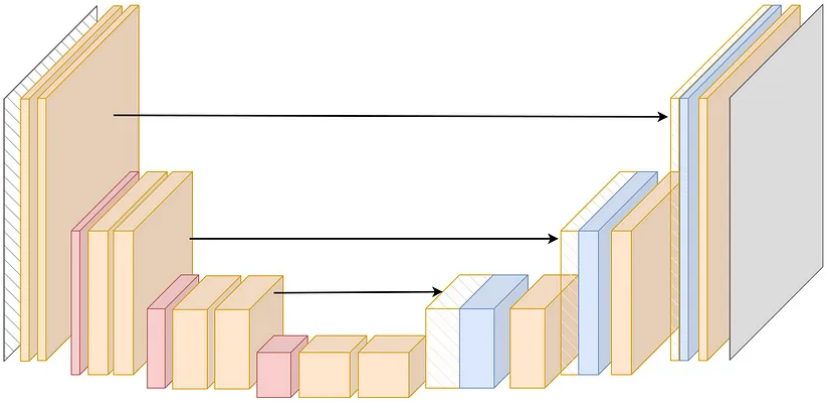


Figura 2.2: U-Net con su característica forma de U.

En el año 2016, en [24] sugieren la red *PSPNet* con módulo *pooling* piramidal **(Figura 2.3)** que captura información contextual a múltiples escalas mejorando la segmentación de objetos grandes y pequeños. Obtuvieron el primer puesto en el desafío *Imagenet 2016* y alcanzaron un $85,4\,\%$ mIoU en *PASCAL VOC-2012* y $80,2\,\%$ de exactitud por clase en *Cityscapes* [25]. Aunque en el último mapa de características se codifica información semántica rica, falta información detallada relacionada con los límites de los objetos debido a la agrupación o convoluciones con

operaciones de avance dentro de la red troncal.

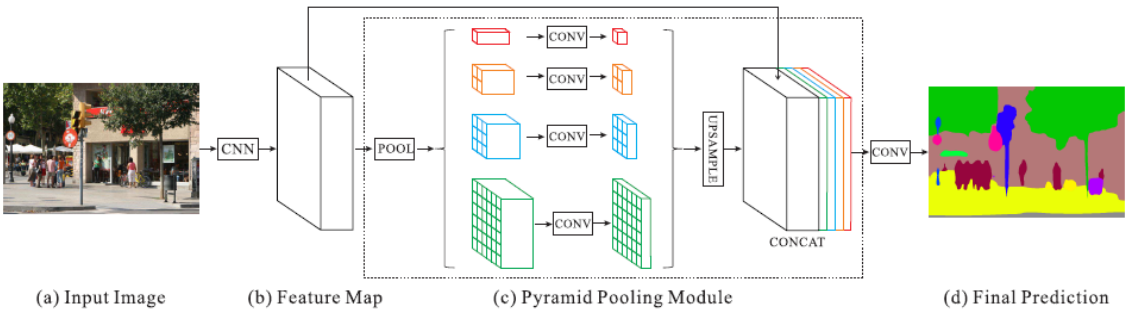


Figura 2.3: Módulo Pooling Piramidal de PSPNet.

Otras arquitecturas aplicadas a los conjuntos de datos *PASCAL VOC-2012* y *Microsoft COCO* [26] demuestran mejoras significativas. Algunos trabajos [27, 28, 29] proponen convoluciones con filtros muestreados llamadas convoluciones atroces o dilatadas (*atrous convolutions*) **(Figura 2.4)** que permiten ampliar el campo de visión del codificador sin aumentar el número de parámetros, la cantidad de cálculos y sin perder resolución, lo que permite segmentar elementos de la imagen a diferentes escalas. En *PASCAL VOC-2012* el modelo *DeepLab* [27] consigue 79.7 % mIoU, *DeepLabV3* [28] con convoluciones atroces en cascada o paralelo consigue 86, 9 % mIoU. Por su parte, *DeepLabV3+* [29] incorporó un decodificar adicional: el módulo *Atrous Spatial Pyramid Pooling* (ASPP) **(Figura 2.5)**, lo que permitió refinar los bordes de los objetos y mejorar la precisión en escenas complejas mejorando los rendimientos anteriores en 89, 0 % mIoU en *PASCAL VOC-2012* y un 82, 1 % mIoU en el conjunto de datos *Cityscapes*.

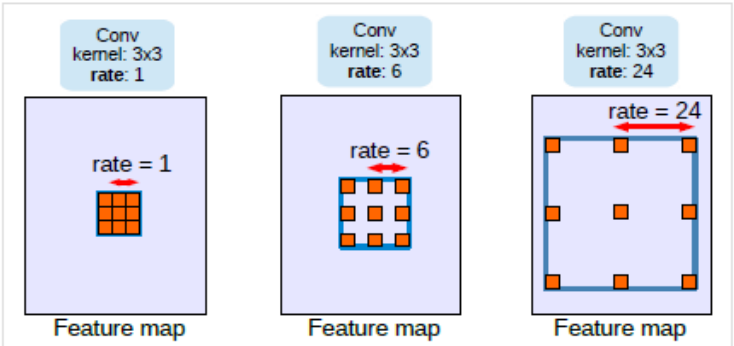


Figura 2.4: Convolución dilatada o atroz con tamaño de kernel 3x3 y diferentes ratios. La convolución estándar corresponde a la convolución atroz con ratio = 1. Emplear un valor grande de tasa atroz aumenta el campo de visión del modelo, lo que permite la codificación de objetos en múltiples escalas.

Hacia el año 2019 surgieron gran variedad de modelos, cada uno proponiendo diferentes soluciones basadas en CNN. El modelo *DANet* [30] es una red de atención dual que integra de forma adaptativa las características locales con sus dependencias globales. Concretamente, agregan dos módulos de atención sobre la FCN dilatada, que modelan las interdependencias semánticas en las dimensiones espaciales y de canal respectivamente. Entrenaron la red con los *backbones ResNet-50* y *ResNet-101*, obteniendo un $81,5\,\%$ mIoU en *Cityscapes* con *ResNet-101* como red troncal. De forma similar, en [31] crean un módulo llamado *Attentional Class Feature* (ACF) para crear la red *ACFNet*, que también se centra de forma adaptativa en las características locales para los centros de clase según cada pixel para obtener una segmentación más fina. En este trabajo informan de un $81,8\,\%$ mIoU en *Cityscapes* con *ResNet-101* como red troncal.

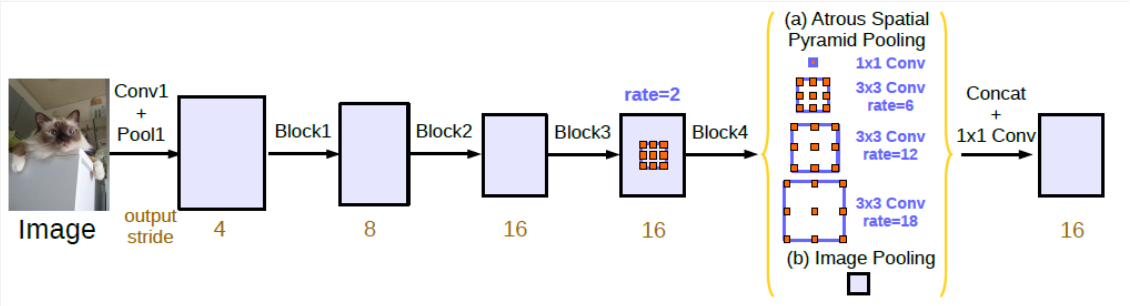


Figura 2.5: DeepLabV3. Agrupamiento piramidal con convoluciones dilatadas (ASPP).

A diferencia de los modelos de la familia *DeepLab* que se basan en el aumento del campo receptivo a través de bloques piramidales, en [32] van más allá del contexto global y utilizan un modelo de Características Co-ocurrentes Agregadas (ACF) invariantes espacialmente, que permiten la segmentación de grano fino. Con el módulo ACF propuesto construyen la red *CFNet* que consigue un $87,2\,\%$ mIoU en *Pascal VOC 2012*.

Como se comentó anteriormente, los modelos de la familia *DeepLab* con bloques ASPP y PPM explotan la información espacial multiescala, en cambio, los modelos *DANet* [30], *CFNet* [32] y *OCNet* [33], consideran las relaciones entre una posición de píxel y sus posiciones contextuales, y agregan las representaciones de las posiciones contextuales con pesos más altos para representaciones similares. En esta línea, los estudios [34, 35] investigan la representación contextual para explorar la relación entre una posición de pixel y su contexto, ya que la etiqueta de clase asignada a un píxel es la categoría del objeto al que pertenece el píxel. Estos estudios proponen el bloque Representación Contextual de Objeto (OCR) **(Figura 2.6)** para usarlo en las redes *SpatialOCR-Net* [34] y *ASPOCRNet* [34] y *HRNet+OCR* [35], que se centran en aumentar la representación de un píxel explotando la representación de la región del objeto

de la clase correspondiente. Reportando en *SpatialOCR-Net* un 83,6 % mIoU, *ASPOCRNet* un 81,8 % mIoU y *HRNet+OCR* un 82,2 % mIoU en el conjunto de test de *Cityscapes*.

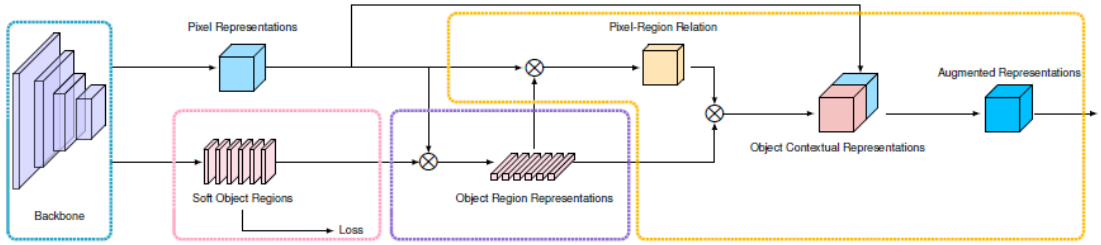


Figura 2.6: Bloque OCR en ASPOCRNet, SpatialOCR-Net y HRNet+OCR.

Para los modelos de CNN residuales, la red *FPNet* [36] consigue deficientes resultados en el conjunto de datos *COCO 2017* con 28,8 % mIoU, sin embargo, la red residual *SUIMNet* [37] reporta un 84,1 % mIoU con *VGG* como red troncal en un conjunto de imágenes subacuáticas.

En los últimos años, se han introducido mejoras significativas en la segmentación semántica con el uso de Transformadores Visuales (ViT), como los modelos *Segmenter* [25], *MaskFormer* [33] y *SETR* [33], que reportan 81,3 % mIoU, 78,5 % mIoU y 81,6 % mIoU en *Cityscapes*. Por otro lado *Trans-Unet* [38] utiliza como codificador un ViT reportando un 77,5 % de Coeficiente medio de Similaridad Dice (mDSC) en conjunto de imágenes médicas. De forma similar a la red *U-Net* [21] en forma de U, la red *SwinT-UNet* [39] tiene un codificador-decodificador en forma de U basada en ViT **(Figura 2.7)** aplicada también en imágenes médicas con la que informan un 79,1 % DSC en el conjunto de datos *Synapse* [40]. Y por último, en [41] basándose también en la red *U-net* y ViT, crean la red *ResUNet-a* [41] aplicada a un conjunto de imágenes aéreas *ISPRS 2D Potsdam* [42] en el que alcanzan una puntuación 92,9 % en mF1 (F1 score medio).

2.2. Segmentación Semántica para conducción autónoma

Los estudios mencionados en la sección anterior se aplican a diferentes conjuntos de datos relacionados o no con la conducción autónoma. Por ejemplo, los que utilizan imágenes médicas no tienen relación con la conducción autónoma, pero los que utilizan por ejemplo *Cityscapes*, sí. Esto no es una limitación puesto que el objetivo es la recolección de diferentes modelos de segmentación semántica. En esta sección se describen algunos trabajos centrados en segmentación semántica para la conducción autónoma.

La mayoría de trabajos de aprendizaje profundo se centran en la percepción visual para conducción autónoma en entornos estructurados utilizando conjuntos de datos estructurados como *Cityscapes* [43], *CamVid* [44], *Mapillary Vistas* [45], *D2-City* [46] y *BDD100k* [47].

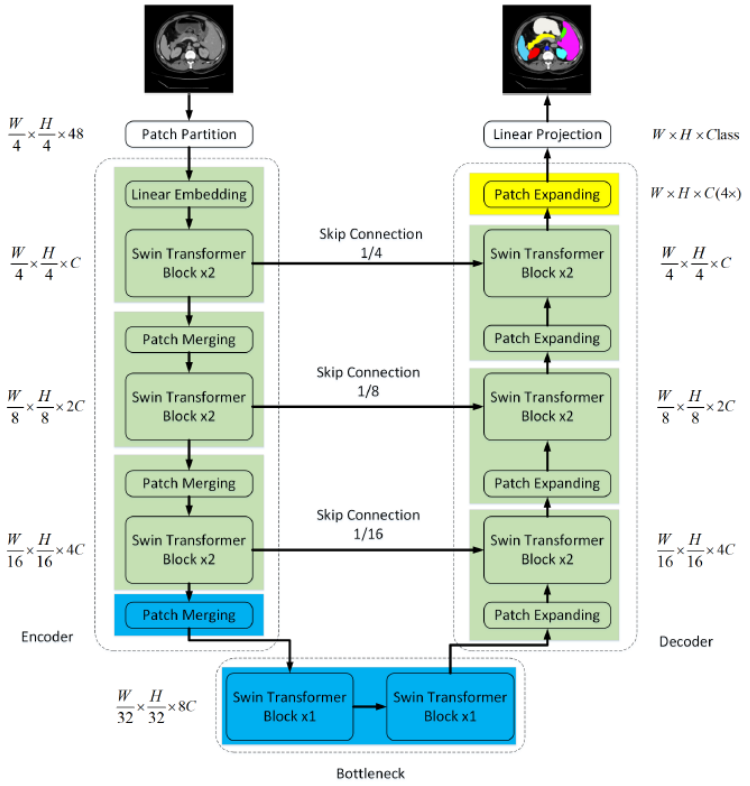


Figura 2.7: Arquitectura en U de SwinT-UNet

Para el conjunto de datos *Cityscapes* se vieron por ejemplo los modelos *PSPNet* [24] y *HRNet+OCR* [35]. La familia de redes *DeepLab* se utiliza en diferentes trabajos [27, 28, 29] con los conjuntos de datos *Cityscapes*, *Mapillary Vistas*, *BDD100k* y *D2-City*. Por otro lado, los estudios [6, 48] con sus redes *SegNet* 56,1 % mIoU y *ENet* 58,3 % mIoU se aplican a *CamVid*.

El desafío actual es la percepción visual para conducción autónoma en entornos no estructurados, ya que los trabajos anteriores pueden o no generalizar bien en dichos entornos. Para ello es necesario el entrenamiento de los modelos con conjuntos de datos no estructurados a diferencia de los vistos anteriormente.

En la *Tabla 2.1* se recoge cronológicamente el estado del arte de conjuntos de imágenes y sus máscaras etiquetadas para segmentación semántica obtenidas en entornos no estructurados.

A partir de la revisión del estado del arte tanto de modelos, como de conjuntos de imágenes en entornos no estructurados, se sientan las bases para la elección de los modelos que se entrenarán con los conjuntos de imágenes también seleccionados.

Año	**Nombre**	**Bibliografía**
2017	DeepScene	[49]
2018	YCOR	[50]
2019	RUGD	[51]
2020	RELLIS-3D	[52]
2021	OFFSED	[53]
2021	TAS500	[43]
2022	SynPhoRest	[44]
2023	WildScenes	[45]
2024	BotanicGarden	[46]
2024	GOOSE	[47]

Tabla 2.1: Conjuntos de imágenes y bibliografía relacionada

En el **Capítulo 4** se describe qué conjuntos de imágenes de los vistos, se utilizarán para el presente estudio, así como una descripción de cada uno de ellos. Es importante conocer el modo de adquisición de las imágenes y las características del entorno para seleccionar el conjunto de imágenes que sea más representativo para la percepción de un robot autónomo en dichos entornos.

Por otro lado, el detalle de los modelos elegidos y los criterios de selección se desarrollan en el **Capítulo 5**, incluyendo también las características importantes de cada una de las redes que se utilizaron como estrategia en su diseño.

Capítulo 3

Herramientas

En esta sección se describen las herramientas utilizadas en este trabajo tanto a nivel de hardware como de software. En primer lugar, se describen los componentes del hardware utilizado para el procesamiento de imágenes, entrenamiento y evaluación de los modelos. Es cuestión importante a la hora de evaluar los tiempos de entrenamiento, ya que, el coste computacional está directamente relacionado con la capacidad de procesamiento y memoria del equipo utilizado. En segundo lugar, se describe el lenguaje de programación y el entorno de trabajo. Y en tercer lugar y último, se describen las librerías necesarias para tareas específicas. La descripción de esta sección permite describir los componentes de hardware mínimos y el software necesarios a la hora de poder replicar los experimentos.

3.1. Hardware

En este trabajo no es necesaria una arquitectura computacional compleja, pero cabe destacar que cuanto mayor capacidad de cómputo tenga la computadora, menor tiempo de procesamiento requerirá el entrenamiento de los modelos. Además es necesaria una capacidad suficiente de memoria para poder almacenar y procesar los diferentes conjuntos de imágenes. La Tabla 3.1, describe el equipo y componentes utilizados en este estudio.

3.2. Lenguaje Python

Se decide utilizar *Python* (versión 3.9.18) como lenguaje de programación por su uso sencillo y la amplia disponibilidad de modelos CNN en este lenguaje. Algunas de las ventajas que ofrece, son las siguientes:

- **Plataforma independiente**: *Python* es un lenguaje interpretado, lo que significa que

puede funcionar en distintos sistemas operativos sin necesidad de modificar el código. Esto hace que sea muy versátil y portátil.

- **Modelos CNN**: amplia y activa comunidad de desarrolladores que contribuyen al crecimiento y el desarrollo de modelos en diferentes ámbitos, incluídos la visión por computador, aprendizaje automático y aprendizaje profundo, áreas de interés para este trabajo.

- **Bibliotecas**: *Python* ofrece una amplia colección de bibliotecas. Están disponibles bibliotecas para manipulación de imágenes como *OpenCV* y *Pillow*, manipulación de datos con *Pandas*, Aprendizaje automático como *Pytorch* y *TensorFlow*, que facilitan las tareas sin tener que programar código desde cero.

Computadora	Portátil Dell Precision-3561 64-bit
Procesador Modelo	11th Gen Intel(R) Core(TM) i7-11850H
Procesador Capacidad	@ 2.5GHz 2496 Mhz
Procesadores Principales	8
Procesadores Lógicos	16
Memoria RAM	32 GB
Disco Duro	952 GB
Tarjeta Gráfica	NVIDIA T1200 Laptop GPU
Sistema Operativo	Windows 10 Enterprise

Tabla 3.1: Especificaciones del portátil

3.3. Plataforma Anaconda

Se opta por la plataforma de desarrollo *Anaconda Navigator* (versión 2.3.1), que incluye la capacidad de desarrollo en lenguaje *Python*. *Anaconda* tiene disponibles aplicaciones en diferentes lenguajes de programación que se centran en diferentes tareas o áreas específicas como pueden ser: el análisis estadístico, manipulación y visualización de datos o el aprendizaje automático. En el proceso de instalación de *Anaconda* se pre-cargan un gran conjunto de librerías, que dependiendo en el área que se desee trabajar, se pueden instalar para su uso sin ser necesario la búsqueda y descarga de éstas librerías en diferentes lugares de terceros. Desde la consola de *Anaconda* se pueden utilizar los comandos *pip* y *conda*, para entre otras tareas, descargar las bibliotecas necesarias desde un único lugar y de forma centralizada.

Otra característica importante de *Anaconda*, es que se pueden crear entornos en los que instalar diferentes versiones de software y librerías específicas para un uso determinado. Esto permite aislar el entorno de trabajo para no incurrir en problemas de compatibilidad y requisitos entre versiones. Por ejemplo, se puede utilizar un modelo con el código creado en una versión antigua de la librería *TensorFlow* e incompatible con su última versión y ejecutarlo en un entorno específico para dicha versión que evite la modificación del código.

Las aplicaciones utilizadas en este trabajo son *Spyder* (versión 3.12.2) y *JupyterLab* (versión 3.4.4) para los cuadernos de código e integrado en *Anaconda Navigator*. *Spyder* es un entorno de desarrollo en forma de scripts, en cambio, *JupyterLab* es otro entorno de desarrollo *Python* pero basado en cuadernos *Jupyter*, en el que puede verse y guardarse el proceso de ejecución del código. Esto permite ver gráficas, datos y métricas que se han utilizado u obtenido en una ejecución de código para su posterior revisión.

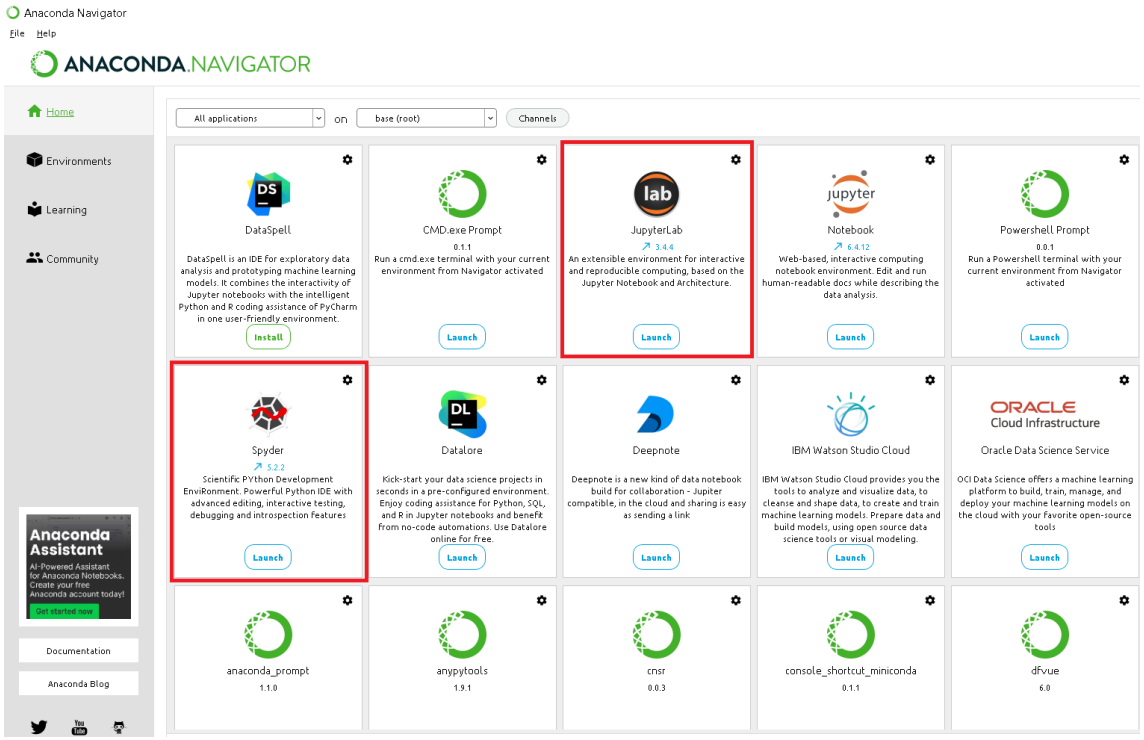


Figura 3.1: Plataforma Anaconda Navigator. En rojo JupyterLab y Spyder.

El uso de *Spyder* se limitará a tareas de procesado de imágenes en el que se desea un tiempo de ejecución corto y no se requiere supervisión ni evaluación de métricas, como puede ser la organización de imágenes en carpetas o su cambio de tamaño. Para el entrenamiento de modelos, evaluación e inferencia de los modelos CNN, que si requieren su posterior revisión se utilizan los cuadernos *Jupyter* generados con *JupyterLab*.

3.4. Librerías

Las librerías más importantes destinadas a las tareas generales de los experimentos realizados, se detallan a continuación:

- **NumPy** (versión 1.26.4): es un paquete fundamental para la computación científica en *Python*. Permite una gran variedad de rutinas para operaciones rápidas con números, cadenas de texto, objetos y matrices, incluidas matemáticas, lógicas, manipulación de formas, clasificación, selección, E/S, transformadas discretas de Fourier, álgebra lineal básica, operaciones estadísticas básicas, simulación aleatoria y mucho más.

- **matplotlib** (versión 3.8.3): es una biblioteca completa para la visualización gráfica, permite crear figuras o gráficos estáticos, dinámicos y visualizaciones interactivas en *Python* de forma sencilla.

- **pandas** (versión 2.2.2): pandas es una herramienta de análisis y manipulación de datos de código abierto rápida, potente, flexible y fácil de usar. Algunas tareas destacadas son el almacenaje de datos complejos en tablas y su manipulación, obtención de estadísticos y guardado en disco en diferentes formatos.

Las librerías específicas de visión artificial y aprendizaje profundo utilizadas en este estudio son:

- **TensorFlow** (versión 2.10.1): facilita a principiantes y expertos la creación de modelos de aprendizaje automático para escritorio, dispositivos móviles, web y la nube.

- **Keras** (versión 2.10.0): es la API de alto nivel de TensorFlow para construir y entrenar modelos de aprendizaje profundo. Se utiliza para la creacion rapida de prototipos, la investigacion del estado del arte (SOTA) y/o en produccion.

- **mmcv** (versión 1.7.2): es una biblioteca para la investigación de visión artificial y proporciona funcionalidades como: procesamiento de imagen/video, visualización de imágenes y anotaciones (máscaras), transformación de imágenes y aceleración de procesamiento con GPU (Cuda).

Las librerías *Tensorflow* y *Keras* se utilizan exclusivamente en *JupyterLab* y *mmcv* únicamente en el procesado de imágenes en Spyder. Todo el software utilizado en este estudio es de licencia y uso libre.

Capítulo 4

Conjunto de Imágenes

4.1. Selección de conjuntos de imágenes

Para poder entrenar los modelos seleccionados en base al objetivo de dotar a robots con percepción visual en entornos no estructurados, se deben seleccionar algunos conjuntos de imágenes que sean representativos. Para ello, se utilizan los siguientes criterios: más recientes en el tiempo, entornos exteriores y todoterreno y que las imágenes sean obtenidas a partir de una cámara RGB situada en un sistema móvil (robot o vehículo).

En la *Figura 9*, se muestra el año de publicación de los conjuntos de imágenes en entornos no estructurados recopilados en la *Tabla 2.1* de la *sección 2.2*. Todos ellos incorporan como mínimo imágenes RGB y sus máscaras con etiquetas.

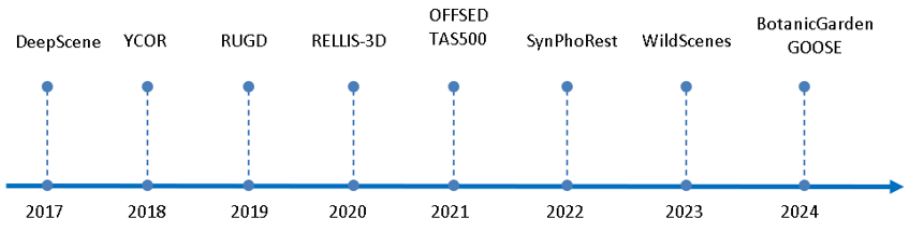


Figura 4.1: Año de publicación de los conjuntos de imágenes no estructurados.

DeepScene y *YCOR* se descartan por ser los más antiguos. *SynPhoRest* se descarta por contener imágenes sintéticas de bosques. Las imágenes de *BotanicGarden* se han obtenido con un sistema que mueve una persona, y no en un dispositivo como un robot o vehículo, por lo tanto queda descartado. *WildScenes* contiene imágenes recopiladas por un robot en jardines y zonas que están bastante delimitadas y *TAS500* contiene imágenes obtenidas con un vehículo por caminos también bastante delimitados, por lo que se asemejan más a un conjunto de imágenes estructurado y se opta por descartarlos. *OFFSED* es un conjunto de imágenes muy pequeño

obtenido a partir de imágenes estéreo, contiene 203 imágenes con 19 etiquetas y se considera insuficiente para poder entrenar las CNNs.

Tras descartar los conjuntos de datos anteriores, se seleccionan los conjuntos de imágenes *RUGD* y *RELLIS-3D*, ambos obtenidos con un robot en entornos todoterreno no estructurados y *GOOSE* de tamaño considerable y reciente lanzamiento (2024) contiene imágenes todoterreno obtenidas desde un vehículo. A continuación se describen los conjuntos de datos seleccionados.

4.2. RUGD

RUGD [51] contiene las secuencias de vídeo continuo del movimiento de un robot terrestre controlado de forma remota por un operador humano. A diferencia de otros conjuntos de imágenes, RUGD contiene más tipos de terreno, límites de clase irregulares, marcas estructuradas mínimas y presenta propiedades visuales desafiantes que a menudo se experimentan en la navegación todoterreno, por ejemplo, fotogramas borrosos, inclinaciones de terreno, ofuscación o ausencia de puntos de fuga.

Robot y sensores

El Robot "Husky" **(Figura 4.2)** está equipado con un sensor de Detección y Medición de Distancias mediante Imágenes Láser (LiDAR), un Sistema de Posicionamiento Global (GPS), una Unidad de Medición Inercial (IMU) y una cámara. Este trabajo de visión artificial se centra únicamente en la información obtenida de la cámara Prosilica GT2750C de 6,1 megapíxeles a 19,8 fotogramas por segundo (fps). La mayoría de imágenes las obtuvieron a mitad de resolución y a 15 fps para respetar un compromiso entre tamaño de archivo y calidad. La cámara está equipada con una lente de 8mm con filtro polarizado con amplia profundidad de campo para condiciones de iluminación exterior, con ajustes de exposición y ganancia para minimizar el desenfoque de movimiento. El robot normalmente funcionaba a su velocidad máxima de 1,0 m/s.

La cámara está montada en la plataforma del robot, de forma que ofrece un punto de vista de cámara único en comparación con otros conjuntos de imágenes. El Husky es significativamente más pequeño que los vehículos que se utilizan habitualmente en entornos urbanos, con unas dimensiones externas de 990 x 670 x 390 mm. El sensor de la cámara está montado en la parte delantera de la plataforma justo por encima de la altura externa del robot, lo que permite obtener un punto de vista del entorno a menos de 25 centímetros del suelo.

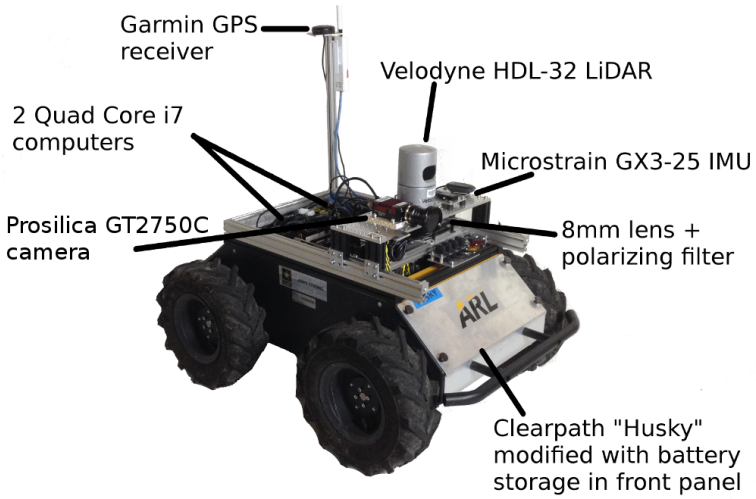


Figura 4.2: Robot "Husky" utilizado para la adquisición de datos en RUGD.

Adquisición

RUGD contiene las 18 secuencias de video de exploración del robot Husky capturadas mientras un humano teleopera el robot en el entorno. La exploración del robot realizada por operador imita el comportamiento que tendría un robot autónomo destinado a tratar de observar visualmente diferentes regiones del entorno. Por lo tanto, las secuencias muestran al robot atravesando no solo carreteras, sino también a través de vegetación, sobre pequeños obstáculos y otros terrenos presentes en el área. La duración promedio de una secuencia de video es de unos 3 minutos.

Entorno

Las rutas de exploración por las que circula el robot, se realizan de forma general, en cuatro categorías ambientales **(Figura 4.3)**:

- **Arroyo** (creek): áreas cerca de un cuerpo de agua con algo de vegetación.
- **Parque** (park): áreas boscosas con edificios y caminos pavimentados.
- **Camino** (trail): áreas que representan terrenos de grava sin pavimentar en bosques.
- **Pueblo** (village): áreas con edificios y caminos pavimentados limitados.

Figura 4.3: RUGD. Categorías ambientales por las que explora el robot.

En RUGD existen fotogramas borrosos, cambios de iluminación, regiones descoloridas, cambios de orientación causados por planos inclinados y perspectivas ocluidas causadas por la travesía a través de regiones de vegetación alta. La inclusión de dichas escenas garantiza que el conjunto de datos refleje fielmente las condiciones realistas que un robot autónomo encontraría en estos entornos, teniendo en cuenta que es difícil capturar y anotar datos a priori de todos los entornos posibles en los que se puede actuar un robot móvil. Las características principales de RUGD se resumen de la siguiente forma:

1. La mayoría de las escenas no contienen bordes geométricos discernibles ni puntos de fuga, y los límites semánticos son altamente irregulares.

2. La travesía de exploración del robot crea rutas irregulares que no se adhieren a un solo tipo de terreno; se atraviesan ocho terrenos distintos.

3. Se encuentran puntos de vista de fotograma únicos debido a terrenos inclinados y oclusión de vegetación.

4. La conducción todoterreno por terrenos accidentados genera un enfoque de encuadre complicado. Estas características proporcionan escenarios de conducción todoterreno realistas que presentan una variedad de nuevos desafíos para la comunidad de investigación, además de los ya existentes, por ejemplo, cambios de iluminación y sombras.

Anotación y Etiquetas

Para que el robot sea capaz de desplazarse por los terrenos deseables y evitar zonas peligrosas y obstáculos para no dañarse ni atascarse, en RUGD se definen 24 etiquetas que representan a los objetos o elementos presentes en los fotogramas. Aunque otras categorías de objetos se ven escasamente en este conjunto de datos, la anotación de etiquetas se limita a las siguientes: *tree, grass, mulch, sky, gravel, bush, rock-bed, asphalt, building, fence, log, vehicle, concrete, water, dirt, bridge, sand, rock, pole, picnic table, container-object, person, sign, bicycle.*

Con el fin de proporcionar anotaciones de calidad para RUGD (máscaras segmentadas), contratan a una empresa externa para dicha tarea y en base a las etiquetas descritas. RUGD incluye un conjunto denso de etiquetas de datos reales por cada cinco fotogramas de cada secuencia de vídeo, resultando en un total de 7.456 fotogramas anotados. Por lo tanto, el conjunto de datos a gran escala completo incluye más de 37.000 imágenes, de las cuales el 20 % incluye anotaciones de datos reales. En la *Figura 4.3* se pueden ver ejemplos de anotaciones de datos reales.

La *Figura 4.4* muestra la distribución de píxeles asignados a su correspondiente etiqueta para todo el conjunto de datos RUGD. Como se aprecia, algunas clases están escasamente representadas en la anotación de la verdad fundamental o *ground truth*. Esto se debe en parte, a la ausencia total de etiquetas en muchas secuencias de videos como *rock-bed, bicicleta, bridge* y *picnic table*. En otros casos, los elementos visuales están escasamente representados. Estos sesgos, demuestran los desafíos reales para la interpretación semántica en entornos no estructurados.

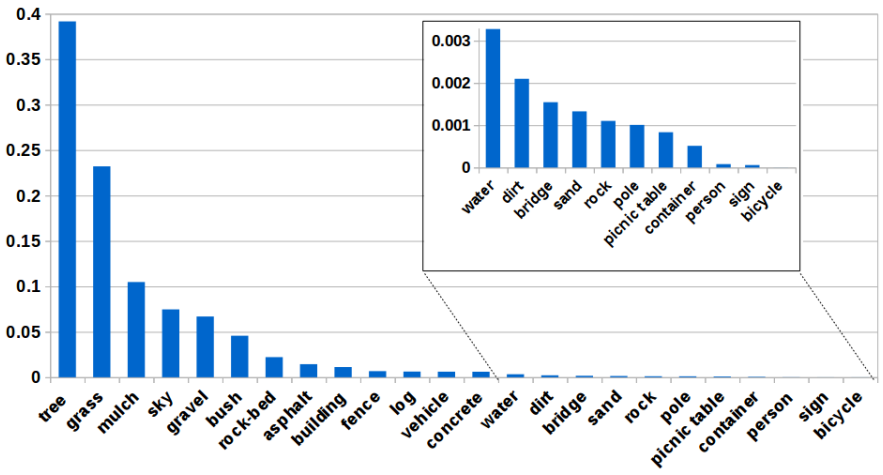


Figura 4.4: RUGD. Porcentaje general de las anotaciones de etiquetas en RUGD completo.

Respecto a la división del conjunto de datos para entrenamiento (train), validación (val) y test (test), proponen en [51] una distribución de 63, 83 %, 9, 83 % y 26, 34 %. Utilizando para la fase de entrenamiento los subconjuntos *train* y *val*, y para inferencia el subconjunto *test*. Como puede apreciarse en la *Figura 4.5*, sólo hay imágenes de la etiqueta *creek* en el subconjunto de *test*, según [51] esto es así para evaluar cómo se comporta el algoritmo de segmentación cuando el robot se encuentre con un entorno nuevo, por el contrario la etiqueta *village* sólo aparece en las imágenes del subconjunto *train*, y no permite su evaluación realizando inferencia. El sesgo en la cantidad de imágenes en el resto de clases hace notar la dificultad de obtener etiquetadas las regiones de las imágenes de forma que represente la realidad del entorno variante que puede encontrarse un robot en entornos no estructurados.

	C	P1	P2	P8	T	T3	T4	T5	T6	T7	T9	T10	T11	T12	T13	T14	T15	V	Total	%
train			x		x	x	x		x		x	x	x	x		x	x	x	4759	63.83
val				x				x											733	9.83
test	x	x								x					x				1964	26.34

Figura 4.5: RUGD. (T: Trail, C: Creek, V: Village, P: Park). Distribución de etiquetas.

4.3. RELLIS-3D

El conjunto de imágenes Rellis-3D [52] se ha obtenido con un robot teleoperado por un humano en el campus Rellis de la Universidad de Texas A&M. Al igual que RUGD, presenta desafíos para los algoritmos existentes, ya que presenta un desequilibrio importante de clases y las condiciones ambientales todoterreno no estructuradas. A diferencia del conjunto de imágenes RUGD, Rellis-3D se caracteriza por ser multimodal.

Robot y sensores

El robot utilizado en Rellis-3D **(Figura 4.6)** está equipado con un sensor LiDAR situado en una montura estabilizadora, una cámara 3D estéreo, una cámara RGB con lentes de 16 mm/F18 con resolución de 1920x1200 a 10Hz y un sistema de navegación inercial (IMU) con GPS.

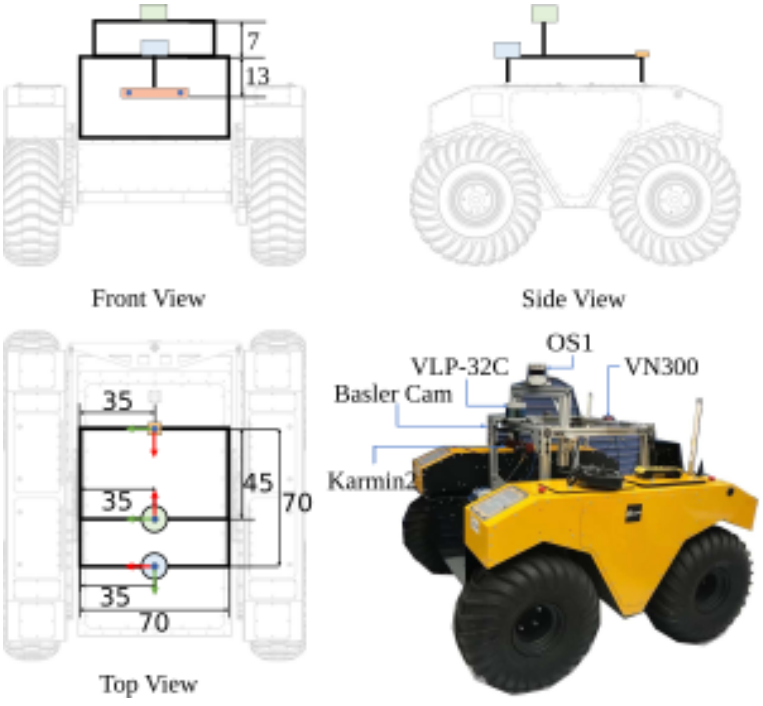


Figura 4.6: Rellis-3D. Robot con sensores de adquisición.

Se integran en el robot dos computadoras, una encargada del control del robot y la otra para la recolección y procesamiento de los datos adquiridos, ambas están conectadas vía Ethernet y están sincronizadas.

Las cámaras del robot se calibran de acuerdo a la suposición del modelo de proyección estenopeica (pinhole) en el que se proyecta una escena 3D en un plano de imagen mediante una transformación de perspectiva. Ajustan los parámetros de proyección estándar expresados por la matriz de calibración K y las distorsiones propias de la lente (radial y tangencial) utilizando la biblioteca *ROS Camera Calibrator* que proporciona *OpenCV*. Además ajustan los parámetros extrínsecos entre el LiDAR y la cámara para tener la posibilidad de fusionar los datos adquiridos por ambos sensores.

Adquisición

Los datos de Rellis-3D se han adquirido en cinco secuencias recopiladas mientras el robot atravesaba tres senderos diferentes no pavimentados del campus A&M de la Universidad de Texas. El primer sendero está cubierto de arbustos y árboles dispersos. Estas secuencias se tomaron en días diferentes y difieren en la dirección en la que se mueve el robot. En el segundo sendero el robot atraviesa zonas de pasto y una zona boscosa. En el tercer sendero se muestra una colina rodeada por un lago y una carretera. La duración de cada una de las tres secuencias es de unos cinco minutos. Pueden verse algunas imágenes adquiridas en la *Figura 4.7*.

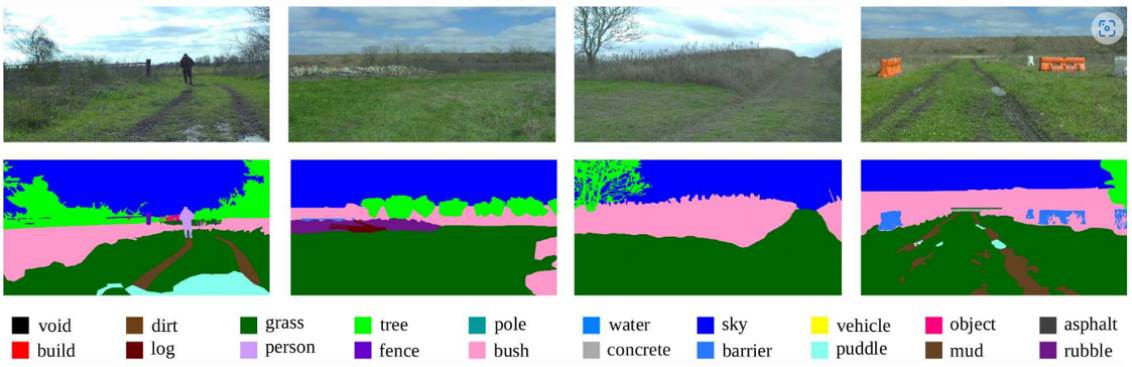


Figura 4.7: Rellis-3D. Ejemplo de imágenes-máscaras RGB.

En [52] definen las clases de objetos y terrenos derivado del conjunto de datos RUGD [51], pero además incluye clases de objetos y terrenos que no están en RUGD, como las clases: barro, barreras artificiales y montones de escombros. También ofrece una estructura de clases más detallada para fuentes de agua, es decir, charcos y aguas profundas, ya que presentan diferente navegabilidad para un robot. En general, hay 20 clases (incluida la clase vacía) presentes en los datos.

Anotación y etiquetas

Rellis-3D contiene 13556 escaneos LiDAR y 6235 imágenes RGB y sus correspondientes máscaras con etiquetas. Las anotaciones a nivel de pixel para las imágenes *ground truth*, las realiza una empresa externa con la reducción de la frecuencia de muestreo de la cámara a 5 Hz, resultando un total de 6235 imágenes y anotaciones a nivel de pixel.

La *Figura 4.8* muestra la distribución de clases para las anotaciones de imágenes y como puede apreciarse está muy desequilibrada. El 94 % del total de píxeles etiquetados corresponden cielo, hierba, árboles y arbustos. La distribución de clases no balanceada presente en Rellis-3D es común entre los conjuntos de datos no estructurados. Aunque es un gran desafío para los algoritmos de segmentación semántica, representa la situación real en la que se encontraría un robot y en la que pueda encontrarse elementos desconocidos o no habituales.

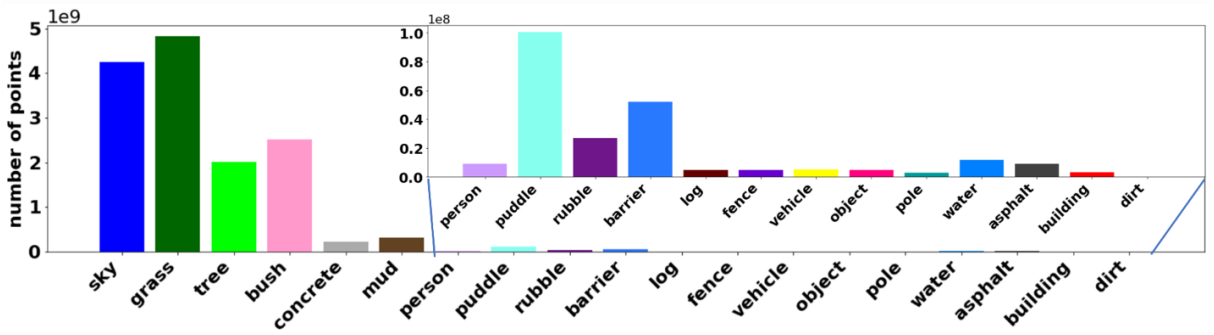


Figura 4.8: Rellis-3D. Número de pixeles que contiene el conjunto de datos para cada etiqueta.

En [52] sugieren dividir RELLIS-3D con 20 etiquetas, en un subconjunto de entrenamiento con 3302 imágenes, validación con 983 imágenes y prueba con 1672 imágenes, intentando mantener el conjunto de entrenamiento lo mayor posible y un conjunto de prueba diverso que incluya escenarios similares y diferentes a los del entrenamiento.

4.4. GOOSE

El recientemente publicado (2024) conjunto de datos GOOSE [47] se ha obtenido a partir de los sensores montados en la plataforma MuCAR-3. GOOSE contiene 10000 pares de imágenes RGB y sus anotaciones a nivel de píxel. También proporcionan un conjunto de datos extendido GOOSE-Ex [54] que es un conjunto de 5000 imágenes adicionales con anotaciones de ajuste fino obtenido en entornos completamente diferentes a los de GOOSE y con dos plataformas adicionales: una excavadora llamada ALICE y robot cuádruple llamado SPOT. A continuación se describe cada una de las plataformas y sus sensores.

Plataforma MuCAR-3

El MuCAR-3 **(Figura 4.9)** está basado en un Volkswagen Touareg con control electrónico. Está equipado con sensor LiDAR, una cámara a color e infrarrojo cercano (NIR) situada detrás del parabrisas. Incluye además 4 cámaras a color montadas cerca del sistema LiDAR del techo, proporcionando una vista de 360° alrededor del vehículo, sensores de radar de corto (SRR) y medio (MRR) alcance, además de un sistema INS/GNSS que permite navegación satelital Cinética en Tiempo Real (RTK) para una localización precisa.

Figura 4.9: GOOSE. Sensores montados en MuCAR-3.

Plataforma ALICE

La plataforma ALICE **(Figura 4.10)** se basa en una excavadora de orugas Liebherr R924 con capacidad de accionamiento por cable con una configuración de sensor personalizada. Los sensores de percepción montados en la plataforma son los siguientes:

- 3x Escáner láser Ouster OS0-64 Revisión C (izquierda, derecha, atrás).
- 1x Escáner láser Ouster OS0-128 (brazo).
- Cámara estéreo Jai FS-3200D-10GE (canales RGB+NIR).
- SBG Ekinox D.
- 4x Cámaras envolventes RGB Alvium G1-240C.

Los escâneres láser y las cámaras están sincronizados.

Figura 4.10: GOOSE. Sensores montados en Alice.

Plataforma Spot

El sistema SPOT **(Figura 4.11)** se basa en el robot Boston Dynamics Spot estándar con dos configuraciones de sensores diferentes, y detalladas a continuación:

Configuración 1

- Escáner láser Ouster OS0-64 Revisión C.
- Cámaras Realsense L515 RGB y escáner láser de estado sólido.
- Sistema SBG Ellipse D INS con dos antenas.

Configuración 2

- Escáner láser Ouster OS0-128 Rev. 7.
- Cámara RGB.
- Sistema SBG Ellipse D INS con dos antenas.

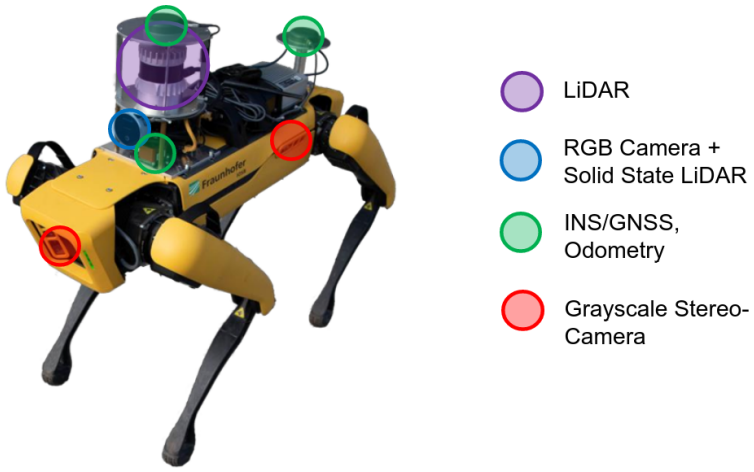


Figura 4.11: GOOSE. Sensores montados en Spot.

Adquisición y entornos

La adquisición de imágenes a color en GOOSE se realiza con una cámara frontal RGB+NIR situada en el techo del MuCAR-3. Se registraron a lo largo de un año, siendo cada día de registro un escenario que consta de múltiples secuencias de cada estación del año. Además incluye una amplia gama de condiciones climáticas (lluvia, soleado, nublado o nevado) y diferentes entornos (campus universitario, caminos forestales, caminos de tierra, jardines, etc.). La división del conjunto de entrenamiento, validación y prueba que proponen, se realiza intentando que compartan una distribución similar entre las clases semánticas de cada división. GOOSE se divide en 12 categorías de entornos: vegetación, terreno, cielo, construcción, vehículo, carretera/camino, objeto, vacío, señal, humano, agua y animal.

Del subconjunto de 5000 imágenes GOOSE-Ex, 2800 se obtuvieron con la excavadora ALICE y 2200 con la plataforma robot SPOT. Al igual que en GOOSE, la división de GOOSE-Ex propuesta para entrenamiento, validación y prueba, se basa en la selección de secuencias de diferentes escenarios, con 3989 imágenes para entrenamiento, 407 para validación y 604 para pruebas. En este caso el subconjunto de prueba, sí incluye las anotaciones. Goose-Ex se divide en las siguientes cuatro categorías de entornos:

- Entorno genérico: combina regiones industriales y todoterreno típicas.
- Entorno de vertedero: incluye estructuras del interior y los alrededores de un vertedero, como entorno operativo típico de una excavadora.
- Entorno de cantera: entorno operativo especial para máquinas grandes, con geometrías de superficie complejas.
- Entorno de obra: área de formación de excavadoras con diversas máquinas pesadas.

Anotación y etiquetas

Las imágenes etiquetadas de GOOSE se obtuvieron por anotación humana. Casi el 90 % de los píxeles anotados están en las categorías vegetación, terreno y cielo, lo esperado en entornos exteriores no estructurados. El 10 % restante de los píxeles anotados componen diferentes tipos de obstáculos como vallas, edificios y postes, así como objetos dinámicos como automóviles, personas y animales. La segmentación es muy granular e incluye 64 clases semánticas **(Tabla 4.1)**. Algunas clases como el agua, están submuestreadas, es decir, aparecen muy poco en el conjunto de imágenes y como es de esperar, las etiquetas está muy desbalanceadas **(Figura 4.12)**.

Categoría	Etiqueta
Animal	animal
Construction	bridge, building, container, debris, fence, guard_rail, tunnel, wall, wire
Human	person, rider
Object	barrel, obstacle, pipe, pole, rock, street_light
Road	bikeway, curb, pedestrian_crossing, rail_track, road_marking, sidewalk
Sign	barrier_tape, misc_sign, traffic_cone, traffic_light, traffic_sign, road_block, boom_barrier
Sky	sky
Terrain	asphalt, cobble, gravel, soil, snow
Vegetation	bush, crops, forest, hedge, high_grass, leaves, low_grass, moss, scenery_vegetation, tree_crown, tree_root, tree_trunk
Vehicle	bicycle, bus, car, caravan, heavy_machinery, kick_scooter, military_vehicle, motorcycle, on_rails, trailer, truck
Void	ego_vehicle, outlier, undefined
Water	water

Tabla 4.1: GOOSE. Categorías y etiquetas

La división que proponen del conjunto de entrenamiento contiene 7830 imágenes, 960 imágenes en validación y 1210 imágenes para pruebas. Esta última a diferencia de RUGD y RELLIS-3D no contiene anotaciones.

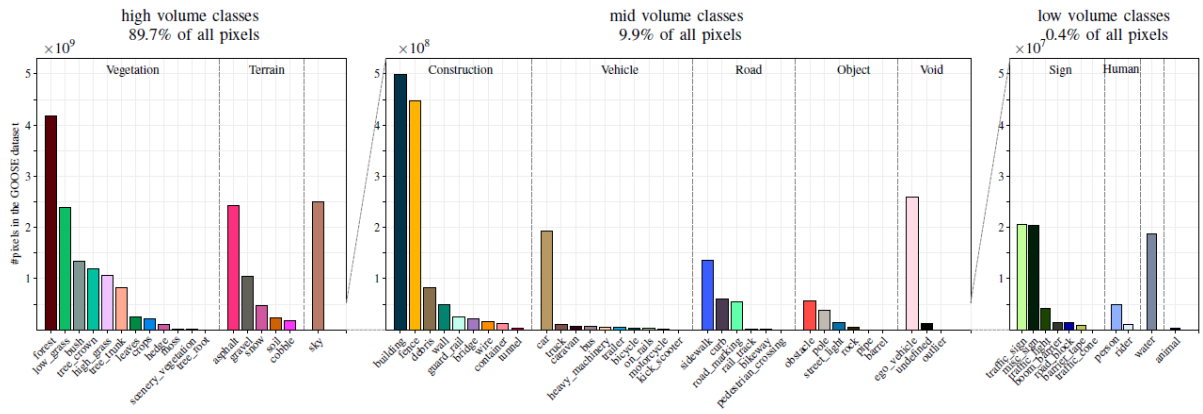


Figura 4.12: GOOSE. Distribución de píxeles por etiquetas.

GOOSE-Ex contiene las mismas 64 etiquetas que GOOSE 4.1, pero las imágenes se han obtenido en los 4 entornos comentados. En los entornos exteriores no estructurados de GOOSE-Ex *(Figura 4.13)*, las categorías de vegetación y terreno forman la mayor parte del conjunto de datos. El suelo es una clase dominante en la distribución en todos los entornos. El entorno genérico contiene la mayor cantidad de vegetación y numerosas clases de volumen medio. Debido a las pronunciadas pendientes del terreno, el entorno del vertedero contiene más cielo que los demás. En las pilas de basura en el vertedero existe un aumento de escombros y obstáculos. La grava y la roca son las clases más frecuentes en el entorno de la cantera. Existen otras clases muy raras en el entorno de la cantera, excepto la maquinaria pesada debido al gran tamaño de dichos vehículos en este entorno. Finalmente, el entorno de obra contiene una apariencia similar de clases con algunos valores atípicos como bloqueos de carreteras y vallas.

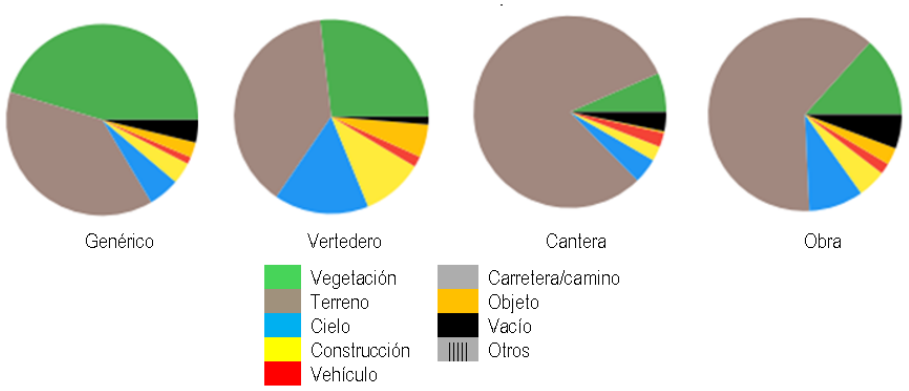


Figura 4.13: GOOSE-Ex. Distribución de píxeles por categorías de entorno.

4.5. Diferencias entre Conjuntos de imágenes

En esta sección se puntualizan de forma general las diferencias más significativas entre los tres conjuntos de imágenes seleccionados.

En RUGD el campo de visión de la cámara se sitúa a 25 cm del suelo, en RELLIS-3D y GOOSE no se tiene dicha información, pero a partir de las plataformas en las que están montadas las cámaras, en RELLIS-3D el campo de visión es muy similar a RUGD. En GOOSE y GOOSE-Ex es donde hay más diferencias, las imágenes GOOSE están obtenidas desde la altura de un vehículo (posiblemente un metro o más), mientra que en GOOSE-Ex los campos de visión son muy dispares, algunas secuencias están tomadas desde una excavadora (superior a un metro) y otras desde un robot semejante a RUGD y RELLIS-3D. Estas diferencias tienen implicación en cómo se "ven"las escenas desde puntos de vista diferentes para un mismo objeto de escena.

Otras diferencias destacables son los entornos de adquisición de las imágenes. Mientras RUGD y GOOSE contienen algunas secuencias con imágenes algo estructuradas, RELLIS-3D y GOOSE-Ex no. En RUGD se atraviesan zonas categorizadas como arroyo, parque, camino y pueblo, mientras que en RELLIS-3D se atraviesan tres tipos de senderos no estructurados. GOOSE incluye de forma general las categorías de entornos de RELLIS-3D y RUGD. Por su parte, GOOSE-Ex contiene entornos no vistos en GOOSE, RELLIS-3D y RUGD, y GOOSE.

Respecto a la división de los conjuntos de imágenes para entrenamiento, validación y prueba propuestos en la bibliografía de cada conjunto de imágenes, en RUGD [51] el conjunto de entrenamiento no contiene etiquetas de la categoría *creek*, en validación no se incluyen etiquetas de la categoría *creek* y *village*, y en *test* no incluyen la categoría *village*. Esto implica que los modelos sólo podrán aprender etiquetas de las categorías *trail* y *park*; las etiquetas de la categoría *village* no tienen información de validación y *test*; y la categoría *creek* sólo se utiliza para inferencia, lo cual permite evaluar como un modelo responde a la hora de segmentar etiquetas de una categoría no vista o entrenada por el modelo. Esta distribución de etiquetas en RUGD hace presuponer que el rendimiento de un modelo será inferior respecto a una división de subconjuntos de entrenamiento, validación y prueba con una distribución de forma que todas las etiquetas estén presentes en los tres subconjuntos. A diferencia de RUGD, en Rellis-3D [52] la división de los subconjuntos de entrenamiento y validación incluyen las 20 etiquetas de los tres tipos de senderos que atraviesa el robot, por lo tanto el entrenamiento se realiza de todas las etiquetas. En el subconjunto de *test* para inferencia algunas etiquetas no se incluyen, por lo que no tendrán representación de inferencia y se presupone que aumentará el rendimiento de un modelo respecto a si se incluyen todas las etiquetas. De forma similar a Rellis-3D, en GOOSE [47] la división del conjunto de imágenes en entrenamiento, validación y prueba propuesta, dicen asegurar que cada división contiene secuencias de cada estación del año y una amplia gama de condiciones climáticas, manteniendo una distribución similar entre las etiquetas que aparecen en cada categoría de etiquetas. En GOOSE-Ex [54] también incluyen todas las categorías de etiquetas en los tres subconjuntos de imágenes e incluyendo imágenes obtenidas de las dos plataformas (excavadora y robot). En todos los conjuntos de imágenes comentados en esta sección se pueden incluir o no imágenes borrosas, con cambios de luz, regiones descoloridas o "quemadas"por el sol, cambios de orientación, planos inclinados y perspectivas ocluidas, entre otros. Presentan también el desafío más importante para aprendizaje automático y es que muchas clases presentan desbalanceo importante, es decir, hay clases muy poco representadas.

La información contenida en esta sección es muy relevante a la hora de evaluar los resultados de los diferentes experimentos, por lo que se tendrá en cuenta para posteriores análisis.

Capítulo 5

Modelos

En esta sección se procede a seleccionar algunos modelos CNN vistos en el **Capítulo 2** que se consideran interesantes para entrenar y evaluar con los conjuntos de imágenes seleccionados en el **Capítulo 4**. Además se incluyen las características más importantes y funcionalidades destacables en los modelos seleccionados, con la finalidad de comprender su funcionamiento y no sólo su aplicabilidad.

5.1. Selección de Modelos CNNs

5.1.1. Modelos de segmentación semántica

A partir de las CNNs para segmentación semántica descritas anteriormente y resumidas en la *Tabla 5.1*, se seleccionan los mejores modelos que se utilizarán en los diferentes experimentos del presente estudio. Se emplea como criterio de selección, aquellos modelos con mayor rendimiento ($> 80\,\%$ mIoU) reportados en la bibliografía y que el tiempo de entrenamiento por época sea inferior a 2 horas aproximadamente.

En base a lo anterior, y después de ejecutar cada modelo en una prueba inicial, se descartan los modelos *HRNet+OCR*, *Segmenter* y *TransUnet* por tener un tiempo de entrenamiento por época demasiado elevado. Se descartan los modelos *MaskFormer* y *SETR* por tener bajo rendimiento ($< 80\,\%$ mIoU) según la bibliografía. El modelo *DeepLab* aunque reporta un rendimiento inferior al $80\,\%$ mIoU no se descarta por ser de la familia *DeepLab* y permitirá estudiar la mejora del rendimiento respecto a *DeepLabV3* y *DeepLabV3+*.

Por otro lado, los modelos *ResUnet-a* y *SwinTUnet* se aceptan porque son modelos más actuales y prometedores, además de que las métricas disponibles no son equivalentes al resto de modelos. En total se seleccionan para este estudio 15 modelos de redes de segmentación.

Año	Modelo	mIoU PASCAL VOC	mIoU Cityscapes	mIoU Otros	mDice Otros	F1-Score Otros
2014	FCN	62,2 %	-	-	-	-
2015	UNET	-	-	-	92,0 %	-
2021	FPNNet	-	-	28,8 %	-	-
2016	PSPNet	85,4 %	80,2 %	-	-	-
2017	DeepLab	79,7 %	71,4 %	-	-	-
2017	DeepLabV3	86,9 %	81,3 %	-	-	-
2018	DeepLabV3+	89,0 %	82,1 %	-	-	-
2019	DANet	-	-	81,5 %	-	-
2019	CFNet	87,2 %	-	-	-	-
2019	SpatialOCRNet	-	83,6 %	-	-	-
2019	ASPOCRNet	81,8 %	-	-	-	-
2019	ACFNet	-	81,9 %	-	-	-
2019	HRNet+OCR	82,2 %	-	-	-	-
2021	Segmenter	59,0 %	81,3 %	51,3 %	-	-
2021	MaskFormer	-	78,5 %	-	-	-
2020	SETR	55,8 %	79,5 %	50,3 %	-	-
2021	TransUnet	-	-	-	77,5 %	-
2020	ResUnet-a	-	-	-	-	92,9 %
2021	SwinTUNet	-	-	79,1 %	-	-
2020	SUIMNet	-	84,1 %	-	-	-

Tabla 5.1: Métricas reportadas para cada modelo SOTA

5.1.2. Backbones

Una vez seleccionados los modelos de segmentación, es necesario seleccionar las redes base para extraer características y conectarlas al bloque de segmentación. En la *Tabla 5.2* se recogen el año de publicación, el número de parámetros y resultados obtenidos con *Imagenet* [20].

Tomando como referencia los *backbones* utilizados en los estudios revisados en la bibliografía, se descartan las CNNs *AlexNet* y *GoogleLeNet*, puesto que en ninguno de los trabajos se han aplicado. De las restantes CNNs de la *Tabla 5.2* se selecciona *VGG16*, *ResNet50* y *EfficientNetb3* [55] que ofrecen un buen equilibro entre precisión y velocidad. El criterio es seleccionar arquitecturas distintas, con mayor rendimiento y con cantidad de parámetros variado. Esto permitirá evaluar cuáles se comportan mejor de forma discriminante.

Año	Nombre	Parámetros	Acc. Imagenet*
2012	AlexNet	˜57 millones	81,10 %
2015	VGG16 Net	˜138 millones	85,74 %
2015	VGG19 Net	˜144 millones	82,40 %
2014	GoogLeNet	6,8 millones	74,80 %
2016	ResNet50	˜25,6 millones	78,50 %
2017	EfficientNetb3	˜12 millones	81,60 %
2019	EfficientNetb5	˜30 millones	83,60 %
2018	EfficientNetb7	˜66 millones	84,30 %

Tabla 5.2: Backbones: parámetros y precisión en Imagenet

Puesto que este trabajo se centra en modelos y *backbones* basados en CNNs, conviene hacer una breve descripción de la convolución y de las características propias de las redes convolucionales.

5.1.3. Redes Neuronales Convolucionales (CNNs)

La operación clave en las CNNs es la convolución. El proceso de convolución en una CNN para una imagen 2D consiste en tomar parches (campo receptivo local) de la imagen de entrada de tamaño m (filas) por n (columnas) y multiplicar elemento a elemento por un filtro de tamaño k (kernel), la suma de los elementos del parche por cada elemento del *kernel*, será el valor del primer píxel de la salida **(Figura 5.1)**.

En el siguiente paso, se obtendrá el siguiente parche de la imagen de entrada mediante un desplazamiento o zancada (*stride*) y se procede a realizar las operaciones como en el paso anterior, obteniendo así el segundo valor de salida **(Figura 5.2)**.

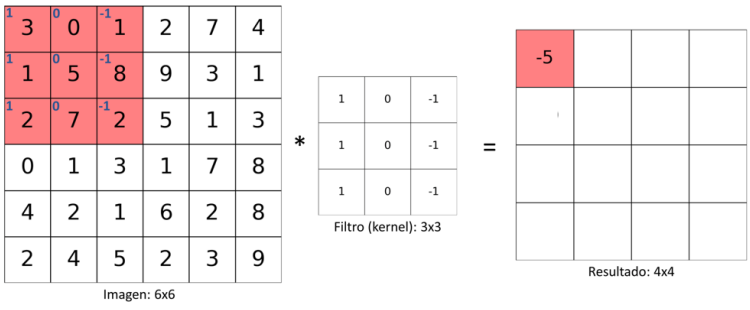


Figura 5.1: Primera iteración de la convolución. A la izquierda la imagen original (6x6) con el kernel (3x3) superpuesto (rojo) y sus valores (azul). A la derecha, y en rojo, el valor resultante de la primera iteración.

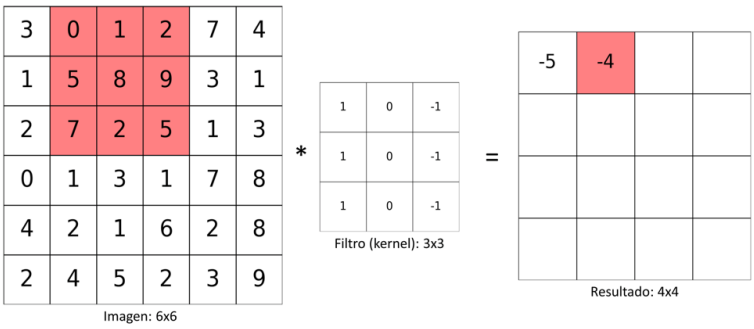


Figura 5.2: Segunda iteración de la convolución. Obtención del segundo valor de salida.

El proceso continúa reiterativamente, hasta que el *kernel* se sitúa en la parte inferior derecha de la imagen original, habiendo recorrido toda la imagen de entrada **(Figura 5.3)**.

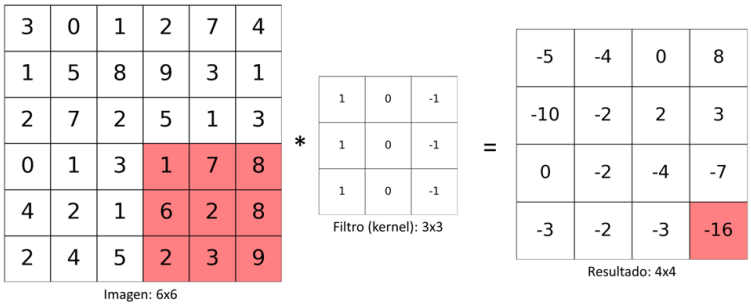


Figura 5.3: Última iteración de la convolución. Obtención del último valor de salida.

Para el ejemplo anterior con una imagen de entrada 6x6 y un *kernel* 3x3, la imagen resultante tiene un tamaño de 4x4. Esta reducción de tamaño o pérdida de resolución es propia de la convolución. En una imagen de entrada de tamaño m (filas) por n (columnas) y un *kernel* de tamaño k por k, la imagen resultante vendrá dada por: $m-k+1$ (filas) por $n-k+1$ (columnas).

Para el caso de imágenes RGB, se tendrá una imagen de entrada (m x n x c), dónde c corresponde al número de canales, en este caso 3. El kernel de tamaño k, también tendrá tres canales. Entonces, la convolución se realiza para cada canal de imagen por cada canal del *kernel*, obteniendo una imagen de salida de un canal y siguiendo la reducción de dimensión comentada anteriormente.

Si se quiere mantener el tamaño de salida igual al de la entrada, se puede utilizar el relleno con ceros o valores intermedios (*padding*). Pero mantener el tamaño de la imagen de entrada en redes profundas es computacionalmente costoso, y en las redes de clasificación se asume que perder resolución tiene un efecto mínimo en la pérdida de características extraídas. Este efecto de pérdida de resolución es denominado *downsampling*, y puede realizarse por el propio proceso de convolución o especificando valores de *stride* > 1 o con capas de agrupamiento (*pooling*) o *max-pooling*. En el caso del *max-pooling* la imagen es dividida en regiones del mismo tamaño,

y para cada región se extrae simplemente el valor máximo que corresponderá a un píxel en la imagen resultante.

Una vez comprendido que una CNN clásica conlleva una pérdida de resolución producida propiamente por la convolución u otras técnicas como el *pooling*, a continuación, se verá cómo se las han ingeniado en los diferentes estudios evaluados para modificar las CNN para tareas de segmentación, en el que se requiere un salida como mapa de características denso.

5.2. Descripción de modelos CNN seleccionados

A continuación, se describen las arquitecturas CNN y las características destacables de los modelos de segmentación semántica seleccionados para los experimentos. La *Tabla 5.3*, recoge el año de publicación del modelo, su nombre, el tamaño de las imágenes de entrada y la red troncal de extracción de características (si utiliza).

Año	**Modelo**	**Entrada**	**VGG16**	**ResNet50**	**EfficientNet**
2014	FCN	320x320	Sí	Sí	Sí
2015	UNet	320x320	No	No	No
2021	FPNet	320x320	Sí	Sí	Sí
2016	PSPNet	300x480	Sí	Sí	Sí
2017	DeepLab	320x320	Sí	Sí	Sí
2017	DeepLabV3	320x320	Sí	Sí	Sí
2018	DeepLavV3+	320x320	Sí	Sí	Sí
2019	DANet	320x320	Sí	Sí	Sí
2019	CFNet	320x320	Sí	Sí	Sí
2019	SpatialOCRNet	320x320	Sí	Sí	Sí
2019	ASPOCRNet	320x320	Sí	Sí	Sí
2020	ACFNet	320x320	Sí	Sí	Sí
2020	ResUNet-a	128x128	No	No	No
2020	SwinTUNet	128x128	No	No	No
2020	SUIMNet	256x320	Sí	No	No

Tabla 5.3: Resumen de Modelos CNN seleccionados a partir de SOTA

5.2.1. FCN (Fully Convolutional Net)

En el estudio [8] titulado "Fully Convolutional Networks for Semantic Segmentation"(Redes Neuronales Completamente Convolucionales para Segmentación Semántica) se presentó la arquitectura FCN en el año 2014. Fue innovadora en su fecha para la segmentación semántica de imágenes porque adaptó las CNNs que originalmente se diseñaron para tareas de clasificación de imágenes, a tareas de segmentación semántica.

Transformación de CNNs a FCN:

A diferencia de las CNNs clásicas que terminan en capas totalmente conectadas (*fully connected layers*) y producen un vector de salida fijo **(Figura 5.4)**, las FCN reemplazan estas capas por operaciones convolucionales. Esto permite que la red maneje entradas de cualquier tamaño y produzca mapas de salida espacialmente densos (es decir, una salida por píxel), conservando la información espacial de la imagen de entrada y obteniendo mapas de características **(Figura 5.5)**, en lugar de vectores de clasificación.

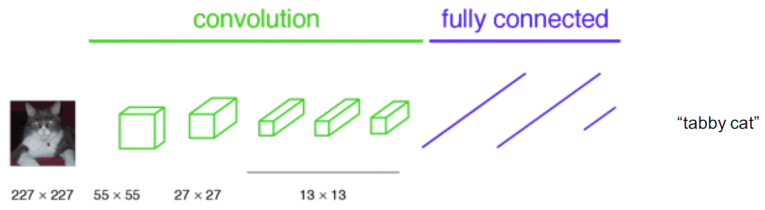


Figura 5.4: CNN de clasificación completamente conectada generando un vector de salida.

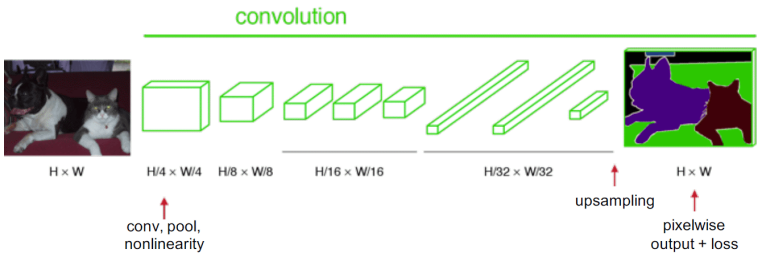


Figura 5.5: CNN de segmentación con *upsampling* generando un mapa de características.

Las CNNs de clasificación típicas, suelen tener entradas de tamaño fijo y producen salidas no espaciales. Con las imágenes *ground truth* que contienen la etiqueta a la que corresponde cada píxel, tanto la propagación hacia adelante como hacia atrás son sencillas y aprovechan la eficiencia computacional propia de la convolución junto a una optimización agresiva.

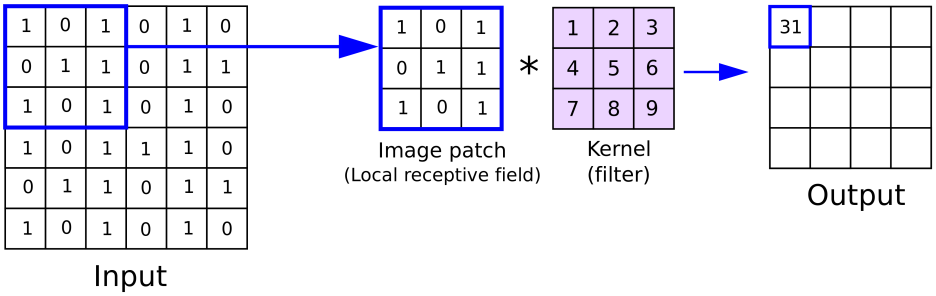


Figura 5.6: Pérdida de resolución: *downsampling*

Downsampling

Como se describió anteriormente, en las CNNs de clasificación a medida que se profundiza en la red con capas convolucionales, la imagen original se reduce, lo que resulta en la pérdida de información espacial **(Figura 5.6)**. Cuanto más profunda es la red, menos información a nivel de píxel queda.

Upsampling

Para la tarea de segmentación semántica, en la que queremos etiquetar cada píxel y obtener información semántica de conjuntos de ellos, es necesario preservar la información espacial. Para producir una salida del mismo tamaño que la imagen de entrada **(Figura 5.7)**, las FCN modifican las capas totalmente conectadas por capas totalmente convolucionales. La salida de una red de clasificación (pérdida de resolución), es modificada por una CNN que expande la resolución para generar una imagen con clasificación densa (segmentación). Dicha expansión provoca un aumento de resolución (*upsampling*) mediante interpolación o convoluciones transpuestas (*transposed convolutions*).

Existen diferentes métodos de sobremuestreo o *upsampling*. En el estudio mencionan la interpolación y la convolución transpuesta. La interpolación más simple, el método del vecino más cercano (*Nearest Neighbor*), consiste en repetir un número de veces dado, los valores de píxel de la entrada, expandiendo la resolución original **(Figura 5.8)**. En cambio, la convolución transpuesta, utiliza diferentes valores de *stride* y *padding* para expandir la resolución. Cada píxel de la entrada se multiplica por el filtro y coloca la salida (del tamaño del filtro) en el mapa de características de salida final. Se mueve nuevamente con un paso (pero esta vez avanza hacia la salida), por lo que aquí aumentar el *stride* aumentará el tamaño de salida, por otro lado, aumentar el *padding* disminuirá el tamaño de salida. **(Figura 5.9)**.

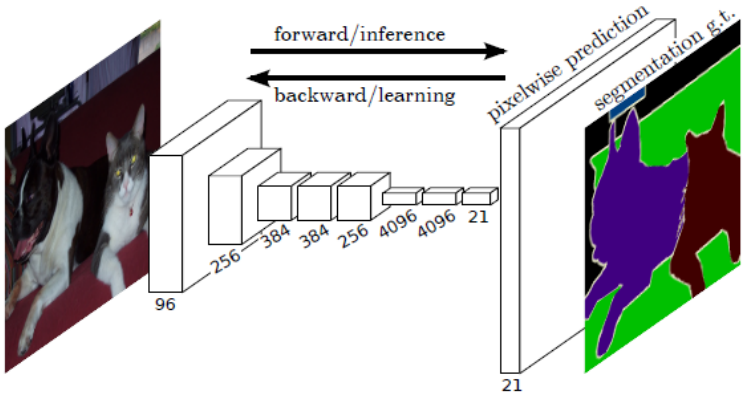


Figura 5.7: CNN de segmentación completamente convolucional que genera un mapa de características denso.

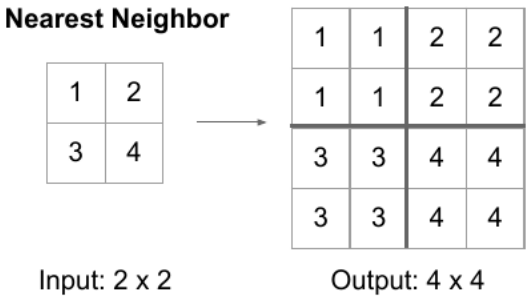


Figura 5.8: *Upsampling* con interpolación *Nearest Neighbor*.

Convolución transpuesta y deconvolución

La convolución transpuesta utilizada en la parte del decodificador en la FCN, es denominada también deconvolución en el ámbito del aprendizaje profundo. Pero en términos estrictamente matemáticos, la deconvolución y la convolución transpuesta no son equivalentes.

En *deep learning*, la convolución transpuesta es una operación que aumenta la resolución espacial de un mapa de características de las CNNs para el aumento de resolución o *upsampling*. A diferencia de una convolución tradicional, que reduce el tamaño del mapa de características, la convolución transpuesta aplica un filtro de manera que expande el mapa de características. Esto se logra insertando ceros o valores intermedios entre los píxeles de entrada y luego aplicando una convolución estándar. En cambio, la deconvolución, en el sentido matemático tradicional, es una operación que invierte el efecto de una convolución. Es decir, dado un mapa de características y un filtro, intenta recuperar la entrada original. La deconvolución es un proceso más complejo, puede ser inestable y no siempre tiene una solución única. En el contexto de las CNNs, el término deconvolución a menudo se usa incorrectamente para referirse a la convolución transpuesta.

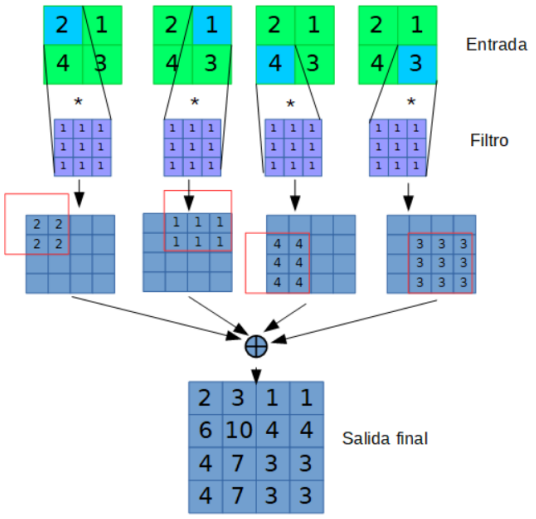


Figura 5.9: *Upsampling* con convoluciones transpuestas.

Arquitecturas FCN y *Skip connections*

En el trabajo [8] proponen tres arquitecturas FCN, la original (FCN-32) no aplica conexiones de salto *skip connections*, pero FCN-16 y FCN-8 si. Las conexiones de salto, combinan información de capas intermedias (con mayor resolución espacial pero menor nivel semántico) con capas más profundas (con menor resolución pero mayor nivel semántico). Esto mejora la precisión en la segmentación al recuperar los detalles finos. Las tres variantes FCN **(Figura 5.10)**, difieren en cómo se combinan las capas mediante las *skip connections*:

- **FCN-32:** Solo utiliza la última capa convolucional para generar la salida con *upsampling* directo, no utiliza *skip connections*.
- **FCN-16**: Combina información de la capa última e intermedia, mejorando la precisión.
- **FCN-8**: Combina información de tres capas (última, intermedia y otra anterior), logrando la mejor precisión al preservar más detalles espaciales.

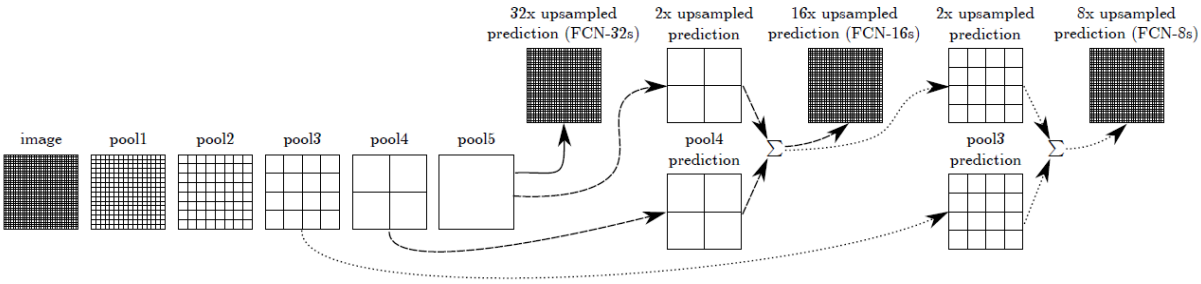


Figura 5.10: Arquitecturas FCN-32, FCN-16 y FCN-8.

Las FCN son una arquitectura poderosa y flexible para la segmentación semántica, que combina la capacidad de las CNNs para aprender características con técnicas para preservar y recuperar la información espacial. Este enfoque sentó las bases para muchas de las arquitecturas modernas de segmentación semántica.

5.2.2. U-Net

Al igual que las FCN, la arquitectura *U-Net* se basa en el modelo codificador-decodificador (*encoder-decoder*), pero se diferencian en que la arquitectura FCN no es simétrica (**Figura 5.7**), mientras *U-Net* sí (**Figura 5.11**).

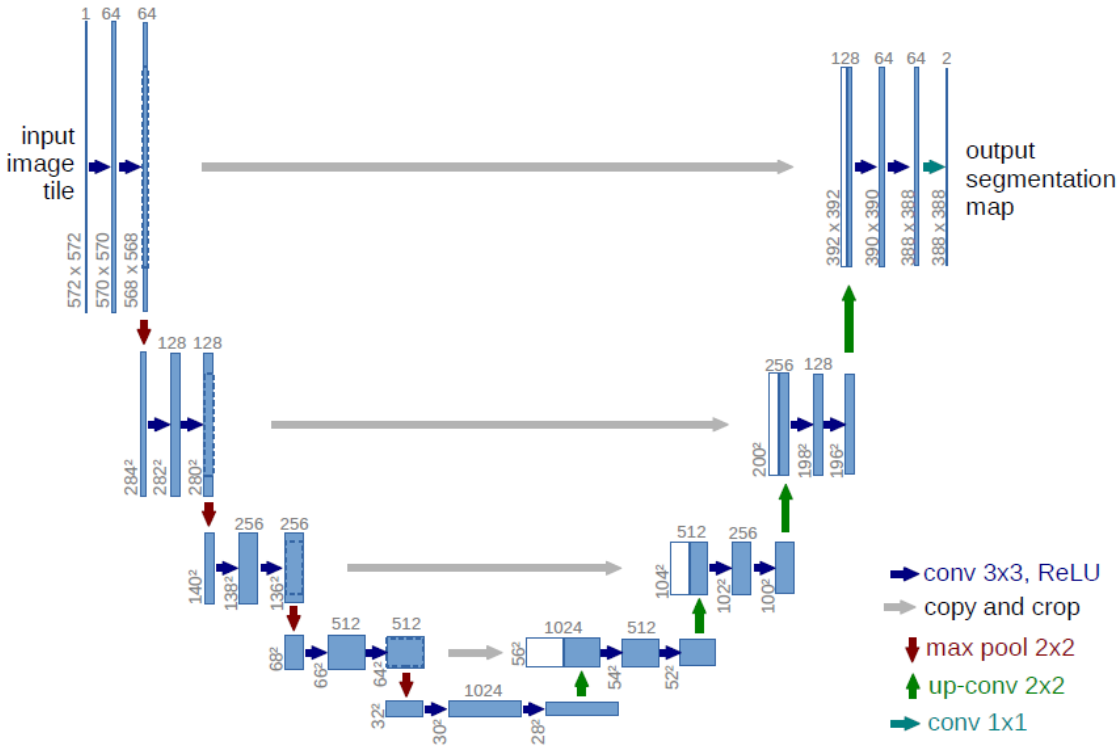


Figura 5.11: CNN de segmentación basada en encoder-decoder.

U-Net toma prestados conceptos como las *skip connections* de FCN y los mejora para obtener mejores resultados. *U-Net* es una CNN que se diseñó inicialmente para tareas de segmentación semántica en el ámbito de las imágenes biomédicas donde se requiere una alta precisión en la delimitación de diferentes estructuras (segmentación fina).

Estructura U-Net

Los dos bloques principales de U-Net son: el *encoder* o parte contractiva (*downsampling*) y el *decoder* o parte expansiva (*upsampling*).

- **Encoder**: captura el contexto de la imagen mediante una serie de capas convolucionales y de submuestreo (*pooling*). Cada bloque en esta parte reduce la resolución espacial pero aumenta el número de canales (profundidad de características). Aplica de forma repetida dos convoluciones sin padding de 3x3, con función de activación *Rectified Linear Unit* (ReLU) y una operación *max-pooling* de 2x2 con *stride* 2 para el submestreo. En cada paso de submuestreo se duplica el número de canales de características.

- **Decoder**: recupera la resolución espacial mediante capas de sobremuestreo (*upsampling*) y convoluciones. Cada bloque en esta parte aumenta la resolución espacial y reduce el número de canales. Cada paso realiza *upsampling* del mapa de características seguido de una convolución 2x2 que reduce a la mitad el número de canales. Se concatena (*skip connection*) el mapa de canales reducido con dos convoluciones 3x3, cada una seguida de una ReLU. En la capa final, se utiliza una convolución 1x1 para mapear cada vector de características de 64 componentes al número deseado de clases. En total, la red tiene 23 capas convolucionales.

En el *decoder*, se utilizan convoluciónes transpuestas con *skip connections* para aumentar la resolución espacial, las cuales son una característica clave de U-Net, ya que, ayudan a preservar los detalles espaciales finos donde hay estructuras pequeñas y complejas. La arquitectura original FCN-32 no las utiliza, en cambio, *U-Net* siempre utiliza convoluciones transpuestas con *skip connections*, que conectan las capas del codificador con las correspondientes del decodificador. Esto permite combinar información de alta resolución (detalles espaciales) con información semántica de bajo nivel. *U-Net* permite una segmentación precisa y detallada, incluso en imágenes con estructuras complejas y pequeñas.

5.2.3. PSPNet

La red *Pyramid Scene Parsing Network* (PSPNet) [24] es una red diseñada para mejorar la segmentación semántica al abordar el problema de la falta de contexto global en las CNNs tradicionales. La idea principal es que, para una segmentación precisa, es crucial comprender no solo los detalles locales, sino también el contexto global de la escena.

PSPNet se basa, al igual que las redes anteriores, en una arquitectura *encoder-decoder*. Su innovación clave es la incorporación de un módulo de agrupamiento piramidal (*Pyramid Pooling Module*) en el *encoder*, que captura información contextual a múltiples escalas. Este enfoque de los autores [24], viene dado por los problemas de segmentación que encontraron en las redes FCN y que compararon con su red propuesta.

Los principales problemas que encontraron, se describen a continuación:

- **Relación de etiquetas no coincidentes**: en una escena pueden existir patrones visuales en diferentes contextos. Por ejemplo, un avión puede estar en el cielo o en la pista, pero no en un lago. FCN al no interpretar la información contextual global, puede cometer errores de clasificación, como el mostrado en la primera fila de la *Figura 5.12*. El barco en el recuadro amarillo, FCN lo clasifica como un *automóvil* basándose en su patrón visual, pero no se tiene en cuenta la información contextual, de que un coche escasas veces puede estar sobre un río. Añadir información contextual global, podría ayudar a resolver este problema.

- **Etiquetas confusas**: existen pares o grupos de etiquetas similares que pueden ser confusos en su clasificación, por ejemplo, una montaña o monte, campo o tierra, césped o vegetación baja. Al tener una apariencia similar, incluso el mismo anotador puede incurrir en errores al etiquetar el *ground truth*, incluyendo píxeles que pertenezcan a una clase errónea. Como ejemplo de este error, en la segunda fila de la *Figura 5.12*, FCN clasifica el elemento del recuadro amarillo, parte como *edificio* y parte como *rascacielos*. Como el elemento de la imagen no puede pertenecer a dos etiquetas a la vez, debería clasificarse todo el elemento con una sola etiqueta. Esto podría solucionarse incluyendo relación entre categorías.

- **Elementos de imagen en diferentes escalas**: en una imagen podríamos encontrar un *gato* en primer plano (gran tamaño) y un *gato* en segundo plano (pequeño tamaño), pero ambos son *gato*. Esta diferencia de escala para una misma categoría de objeto, puede incurrir en errores. Los elementos pequeños (pocos píxeles) pueden ser difíciles de encontrar y que sean importantes para una segmentación fina. En cambio, los elementos grandes (muchos píxeles), pueden superar el tamaño del campo receptivo de la red (tamaños de parche) y produzcan una clasificación discontinua o errónea. En el recuadro amarillo de la tercera fila de la *Figura 5.12*, hay una *almohada* con apareciencia similar a la *sábana*. FCN clasifica la sábana y la almohada de forma conjunta. Por lo tanto, si se analizaran subregiones de la imagen, podria solucionarse el problema con elementos de imagen grandes y pequeños.

De forma resumida se aprecia que los posibles errores de clasificación de píxeles viene dado por la ausencia de información contextual global, la relación entre categorías y el uso de campos receptivos de tamaño fijo.

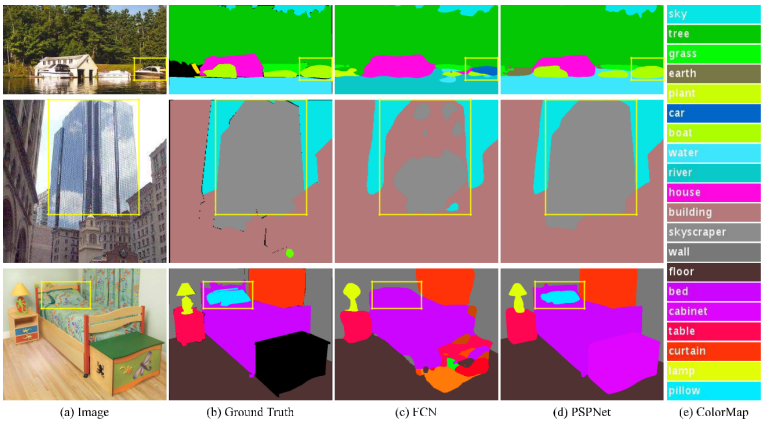


Figura 5.12: PSP-Net. CNN de segmentación basada en *encoder-decoder*.

Estructura PSPNet

Como puede apreciarse en la *Figura 2.3*, PSPNet dispone de un *encoder* basado en CNN para extraer características que pasan por un Módulo de Agrupamiento Piramidal (*Pyramid Pooling Module*), después realiza *upsampling* por interpolación para finalizar con una capa totalmente conectada para clasificación de píxeles.

Pyramid Pooling Module (PPM)

Este módulo realiza un agrupamiento promedio de las características extraídas del encoder (información contextual global), pero esto no es suficiente para la resolución de todos los problemas que se detectaron. Para ello, el módulo divide el mapa de características agrupado en cuatro subregiones a diferentes escalas (por ejemplo, 1x1, 2x2, 3x3, 6x6) y aplica pooling (por ejemplo, average pooling) a cada subregión.

El nivel más grueso es la agrupación global para generar una salida de un solo *bin*. El siguiente nivel de pirámide separa el mapa de características en diferentes subregiones y forma una representación agrupada para diferentes ubicaciones. La salida de los diferentes niveles en el módulo de agrupación de pirámides contiene el mapa de características con distintos tamaños. Para mantener el peso de la característica global, se utilizan capas convolucionales de 1x1 después de cada nivel de pirámide para reducir la dimensión de la representación del contexto a 1 = N de la original si el tamaño del nivel de la pirámide es N. Después, las características resultantes de cada nivel de la pirámide se interpolan (*upsampling*) para que tengan el mismo tamaño que el mapa de características original. Finalmente, estas características se concatenan con el mapa de características original, lo que permite que la red combine información de contexto local y global.

5.2.4. Familia DeepLab

La red DeepLab en el estudio [27] propone el uso de convoluciones dilatadas (*atrous convolutions*) para aumentar el campo receptivo de la red sin reducir la resolución espacial. Introduce el modelo estocástico Campo Aleatorio Condicional (*Conditional Random Field* o CRF) como post-procesamiento para refinar los bordes de la segmentación. Además, utiliza una red base (como VGG-16) modificada con convoluciones dilatadas.

Convolución dilatada (*Atrous convolution*)

La convolución dilatada **(Figura 2.4)** es una forma de realizar la propia convolución pero sobremuestreando los filtros. Esto equivale a expandir los filtros añadiendo espacios con ceros para aumentar el campo receptivo del filtro y aumentar la resolución contextual, pero sin aumentar el número de parámetros y el número de operaciones. Esto añade un hiperparámetro más a la convolución, el factor de dilatación *d*. Si el factor de dilatación es igual a uno, se produce una convolución normal. El dejar esos espacios "vacíos" puede provocar la pérdida de cierta información de la vecindad.

Atrous Spatial Pyramid Pooling (ASPP)

La segunda versión de DeepLab introduce el *Atrous Spatial Pyramid Pooling* (ASPP) **(Figura 5.13)**, un módulo que aplica convoluciones dilatadas para capturar información contextual a múltiples escalas, combinando características de diferentes niveles de resolución (pirámides) y utilizando también CRF como post-procesamiento.
Este módulo combina las características del módulo de agrupamiento piramidal visto en *PSPNet* con las convoluciones dilatadas. Utiliza capas de convolucionales dilatadas paralelas con diferentes factores de dilatación (campos receptivos) **(Figura 5.13)**. Las características extraídas para cada factor *d*, se procesan posteriormente en ramas separadas y se fusionan para generar el resultado final.

La tercera versión DeepLabV3 [28], elimina CRF como post-procesamiento, ya que la red es capaz de aprender características más robustas sin necesidad de refinamiento adicional y posible pérdida de información de la vecindad. Mejora el módulo ASPP al incluir en las *atrous convolutions* de diferentes tasas de dilatación, una capa de agrupamiento global (*global average pooling*) para capturar información contextual a nivel de toda la imagen. Este ASPP captura de manera efectiva información multiescala, pero en [28] descubrieron que a medida que el factor de dilatación se hacía más grande, la cantidad de pesos de filtro válidos (es decir, los pesos que se aplican a la región de características válidas, en lugar de rellenar con ceros) se hacía

más pequeña. Por ejemplo, al aplicar un filtro 3x3 a un mapa de características de 65x65 con diferentes factores de dilatación, en el caso extremo en el que el valor del factor de dilatación es cercano al tamaño del mapa de características, el filtro 3x3, en lugar de capturar todo el contexto de la imagen, se degenera en un simple filtro 1x1, ya que sólo el valor del filtro central es efectivo.

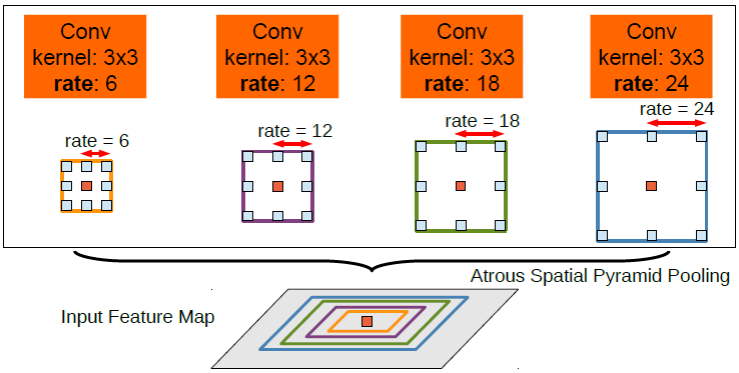


Figura 5.13: ASPP. Para clasificar el píxel central (naranja), ASPP aprovecha las características de múltiples escalas usando múltiples filtros paralelos con diferentes factores de dilatación (rate). Los campos receptivos se muestran en diferentes colores.

Para superar este problema incorporan información de contexto global, añadiendo agrupamiento promedio global (*global average pooling*). Las características extraídas se convolucionan con 256 filtros de 1x1. Después realizan *upsampling* por interpolación bilineal para obtener el número de etiquetas a clasificar.

El ASPP mejorado **(Figura 2.5)** consta de una convolución 1x1 y tres convoluciones 3x3 con factores de dilatación (6, 12, 18), todas con 256 filtros y normalización por lotes. Las características resultantes de todas las ramas se concatenan y pasan por otra convolución 1x1 (también con 256 filtros y normalización por lotes) antes de la convolución 1x1 final que genera las salidas finales.

Por último, la red *DeepLabV3+* extiende el modelo anterior *DeepLabV3* añadiéndole un *decoder* que combina características de bajo nivel (alta resolución espacial) con características de alto nivel (alto nivel semántico). Para ello utiliza una arquitectura *encoder-decoder*, donde el *encoder* es *DeepLabV3* y el *decoder* refina la segmentación **(Figura 5.14)**. El *decoder* toma las características de salida del *encoder* y las combina con características de capas intermedias de la red base de forma similar a las *skip connections* vistas en *U-Net*. Esto mejora la precisión en los bordes y detalles finos. Mantiene el módulo ASPP en el *encoder* para capturar información contextual a múltiples escalas.

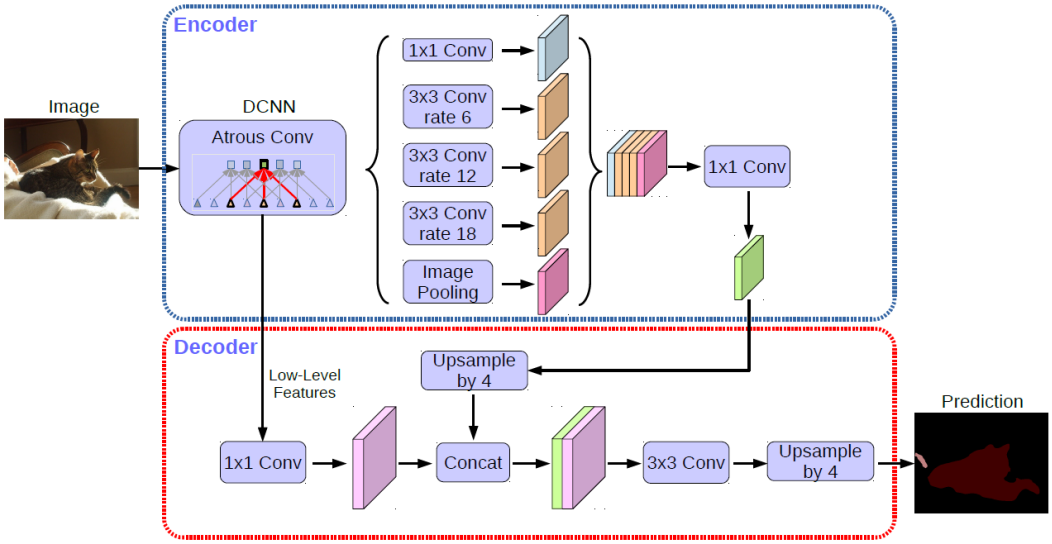


Figura 5.14: DeepLabV3+. DeepLabV3 con módulo encoder que codifica información contextual multiescala aplicando convolución dilatada en múltiples escalas. El módulo decodificador refina los resultados de segmentación a lo largo de los límites de los objetos.

DeepLab fue pionero en el uso de convoluciones dilatadas y ASPP para capturar información contextual. *DeepLabV3* mejoró la precisión al eliminar la dependencia del CRF y mejorar el ASPP. Y por último *DeepLabV3+* introdujo un *decoder* para combinar características de diferentes niveles de resolución, logrando una segmentación más precisa y detallada.

5.2.5. DANet

La *Dual Attention Network* (DANet) [30], es una red diseñada para mejorar la segmentación semántica de escenas, al capturar dependencias contextuales tanto a nivel de posición como de canal.

Estructura DANet

DANet se basa en una arquitectura encoder-decoder, con las mismas funciones que las redes anteriores. La innovación clave de *DANet* es la incorporación de dos módulos de atención: *Position Attention Module* (PAM) y *Channel Attention Module* (CAM).

Módulo de Atención de Posición (PAM)

PAM **(Figura 5.15)** captura las dependencias espaciales entre diferentes posiciones de la imagen, utilizando una matriz de atención para calcular las relaciones entre cada par de posiciones en el mapa de características. Las características resultantes se ponderan según estas relaciones, lo que permite que la red capture contextos globales y mejore la coherencia espacial.

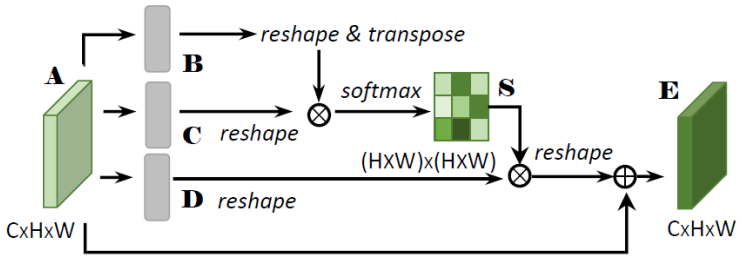


Figura 5.15: DANet. Módulo de atención de posición

Módulo de Atención de Canal (CAM)

El módulo CAM **(Figura 5.16)** captura las dependencias entre diferentes canales de características con una matriz de atención para calcular las relaciones entre cada par de canales en el mapa de características. Las características resultantes se ponderan según estas relaciones, lo que permite que la red refuerce las características más relevantes para la segmentación.

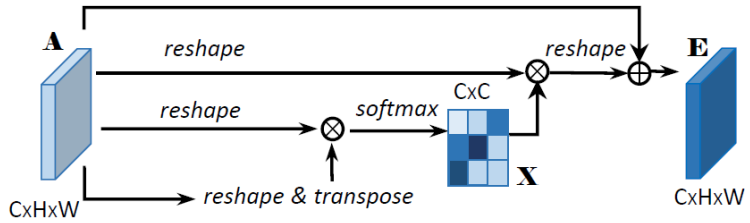


Figura 5.16: DANet. Módulo de atención de canal

Combinación de Módulos de Atención

Los módulos PAM y CAM se aplican de manera paralela **(Figura 5.17)** a las características extraídas por el encoder. Las características resultantes de ambos módulos se suman para producir un mapa de características enriquecido, que pasa al *decoder* para generar el mapa de segmentación final.

La capacidad de capturar dependencias contextuales a nivel de posición y canal permite que *DANet* supere a otras arquitecturas en tareas que requieren una comprensión global de la escena.

5.2.6. CFNet

La red *CFNet* [32] captura características co-ocurrentes, es decir, relaciones entre diferentes regiones de la imagen que suelen aparecer juntas. Se basa en utilizar las relaciones contextuales entre diferentes partes de la imagen.

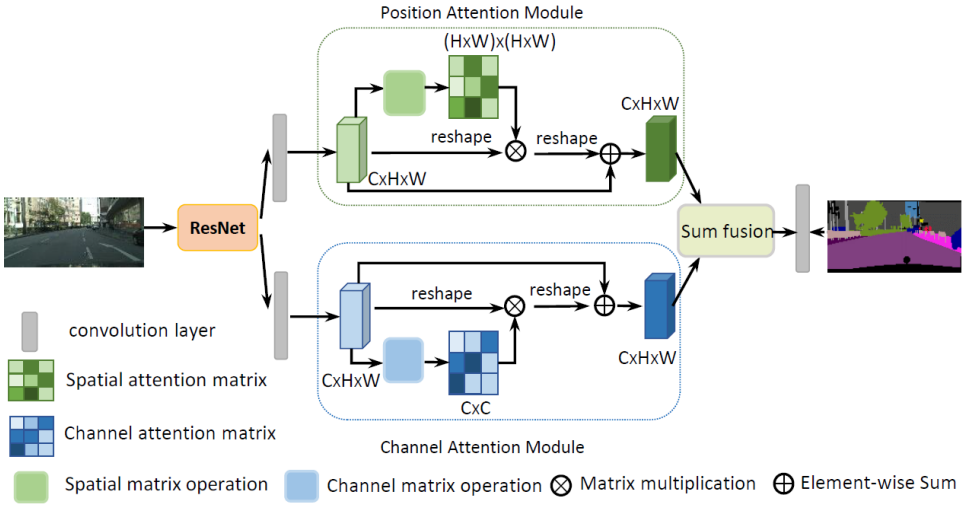


Figura 5.17: DANet. Combinación de PAM (verde) y CAM (azul)

Estructura CFNet

Al igual que la mayoría de modelos anteriores, *CFNet* se basa en una arquitectura encoder-decoder. Su innovación clave es la incorporación de un módulo de extracción de características Co-Ocurrentes (*Co-Occurrent Feature Extraction*).

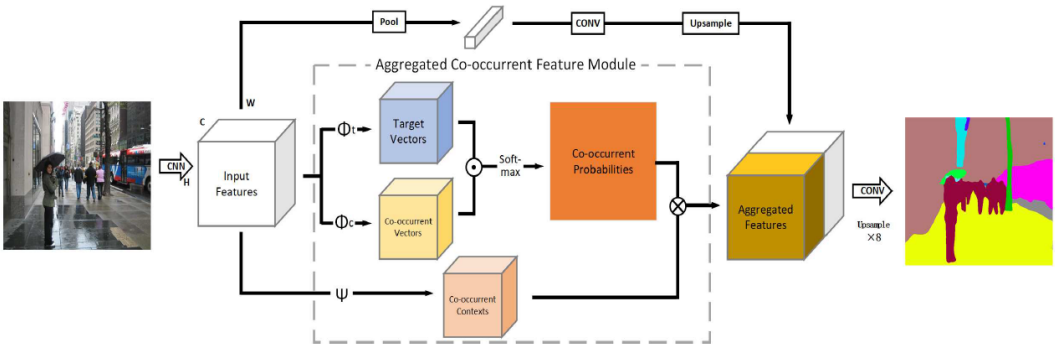


Figura 5.18: CFNet con modulo de extracción de características Co-ocurrentes

Módulo de Extracción de Características Co-Ocurrentes

Este módulo captura las relaciones entre diferentes regiones de la imagen mediante la construcción de un grafo de co-ocurrencia, donde cada nodo representa una región de la imagen, y las aristas representan las relaciones entre estas regiones. Además, utiliza una red de atención para calcular las relaciones entre los nodos del grafo, lo que permite que la red capture dependencias contextuales a largo alcance.

El *decoder* toma las características enriquecidas por el módulo de extracción de características co-ocurrentes y genera el mapa de segmentación final, utilizando convoluciones y operaciones de *upsampling* para recuperar la resolución espacial y producir una salida del mismo tamaño que

la imagen de entrada.

La capacidad de capturar características co-ocurrentes permite que *CFNet* supere a otras arquitecturas en tareas que requieren una comprensión global de la escena.

5.2.7. ASPOCRNet y SpatialOCRNet

Una arquitectura avanzada para segmentación que combina representaciones contextuales de objetos con mecanismos de atención espacial, es la red *SpatialOCRNet* (*Object-Contextual Representations with Spatial Attention*) y su variante *ASPOCR-Net* [34]. Esta red fue propuesta como una mejora sobre la arquitectura *OCRNet* (*Object-Contextual Representations Network*), incorporando un módulo de atención espacial para refinar aún más las características y mejorar la precisión en la segmentación.

ASPOCRNet y *SpatialOCRNet* se basan en la idea de que los objetos en una imagen tienen un contexto específico que puede ser explotado para mejorar la segmentación semántica, combinando representaciones contextuales de objetos (OCR) con un módulo de atención espacial para capturar dependencias contextuales y espaciales de manera más efectiva.

Estructura SpatialOCRNet

La arquitectura sigue un esquema *encoder-decoder* con la innovación de incorporar dos módulos principales: el OCR y el Módulo de Atención Espacial (*Spatial Attention Module*), similar al PAM visto en la red DANet.

Estructura ASPOCRNet

ASPOCRNet se basa en la estructura *SpatialOCRNet* pero añade al módulo OCR atencional, un módulo ASP que extrae características contextuales multiescala mediante convoluciones dilatadas.

Object-Contextual Representations (OCR)

Este módulo **(Figura 2.6)**, captura el contexto de los objetos en la imagen. La idea es que cada píxel no solo debe ser clasificado en función de sus características locales, sino también en función del contexto de los objetos a los que pertenece. Primero se calcula una región de interés (RoI) para cada objeto en la imagen y se agregan las características de los píxeles dentro de cada RoI para obtener una representación contextual del objeto. Después estas representaciones contextuales se fusionan con las características originales para enriquecer la información semántica.

Spatial Attention Module (SAM)

SAM refina las características espaciales mediante un mecanismo de atención que permite a la red enfocarse en las regiones más relevantes de la imagen. En este caso, se calcula un mapa de atención espacial que pondera la importancia de cada región en la imagen y luego las características se multiplican por este mapa de atención para resaltar las regiones más importantes y suprimir el ruido o las regiones irrelevantes. El *decoder* toma las características enriquecidas por los módulos OCR y SAM para generar el mapa de segmentación final. Esta combinación permite una segmentación más precisa y detallada, especialmente en escenas complejas con múltiples objetos y relaciones contextuales.

5.2.8. Attentional Class Feature Network (ACFNet)

La *Figura 5.19* muestra la red *Attentional Class Feature Network* (ACFNet) comentada en el estudio [31]. En este estudio presentan el concepto de *centro de clase*, que representa el nivel de contexto de una clase para ayudar a clasificarlos dentro de las diferentes categorías. Para ello, proponen el módulo *Attentional Class Feature* (ACF), para hacer que los diferentes píxeles se enfoquen de forma adaptativa en diferentes centros de clase. La estrategia general se centra en una estructura que va de una segmentación gruesa a una más fina.

Se propone una estrategia diferente a las vistas anteriormente. Crean una red de segmentación basada en una red CNN base con módulo ASPP para obtener una primera segmentación gruesa. A partir de las segmentaciones gruesas extraen información de clases que concatenan con la segmentación anterior y generan una segmentación más fina.

Estructura ACFNet

La red inicia con un *backbone* para extraer características que pasan por un módulo ASPP, generando un mapa de características denso de grano grueso por un lado, y un mapa de características por otro. Esta primera segmentación y las características extraídas entran al módulo ACF **(Figura 5.19)**.

***Attentional Class Feature* (ACF)**

Consta de dos bloques, el bloque de centro de clase (CCB) y el bloque de atención de clase (CAB), que se utilizan para calcular el centro de clase y la característica de clase atencional respectivamente. El módulo ACF se basa en una estructura de segmentación de gruesa a fina. La entrada del módulo ACF es el resultado de la segmentación gruesa y el mapa de características en la red base y la salida es la característica de clase atencional.

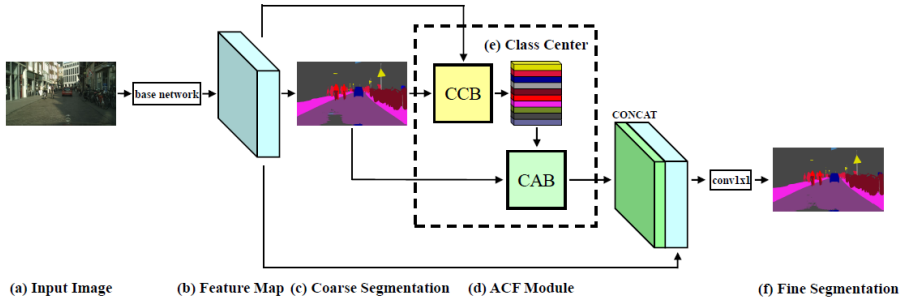


Figura 5.19: ACFNet. *Módulo Attentional Class Feature* (ACF)

Bloque de Centros de Clase (CCB)

Este bloque **(Figura 5.20)** calcula los centros de clase, entendiendo como centro de clase al promedio de las características de todos los píxeles que pertenecen a una misma clase. Como las imágenes *ground truth* no están disponibles durante esta fase, utilizan el resultado de la segmentación gruesa para evaluar la probabilidad de que un píxel pertenezca a una clase específica. Los píxeles que tengan mayor probabilidad de pertenencia a una determinada clase, contribuirán más en el cálculo del centro de clase. Al final se obtiene una matriz con los centros de clase para cada clase.

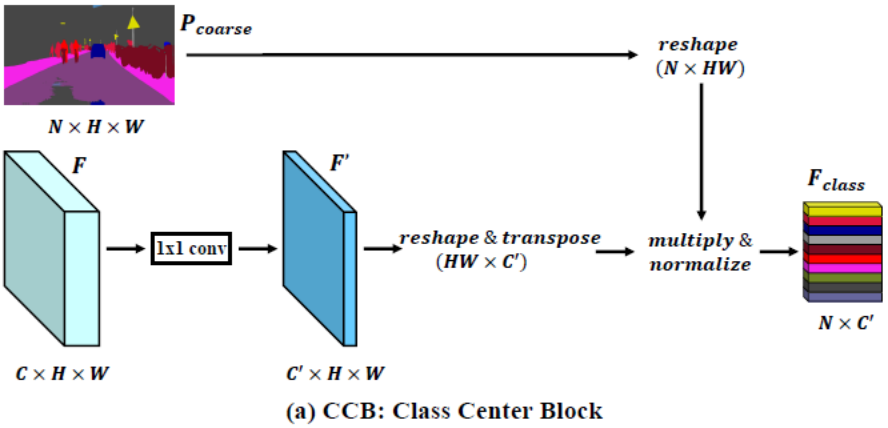


Figura 5.20: ACFNet. Bloque CCB para obtención de centros de clase a partir de la segmentación gruesa y las características extraídas.

Bloque de atención de clase (CAB)

Los centros de clase junto a los mapas de segmentación gruesa entran al bloque CAB **(Figura 5.21)**. La matriz de centros de clase se transpone y se modifica el tamaño de los mapas de segmentación gruesa obtenidos inicialmente, para después concatenarlas por medio de la multiplicación de ambas matrices. Tras la modificación de tamaño del producto obtenido, se obtiene la matriz características de clase atencional, que pasa por una convolución 1x1 que genera los

mapas de segmentación fina.

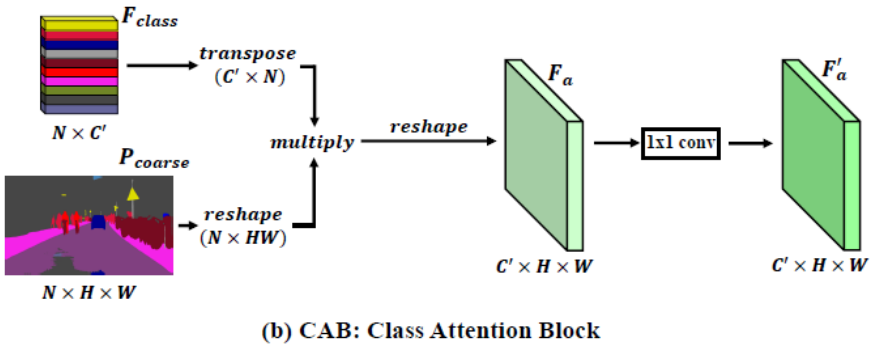


Figura 5.21: ACFNet. Bloque de atención de clase (CAB)

5.2.9. ResUNet-a

En el modelo *ResUnet-a* [41] se combinan diferentes bloques y estrategias vistas anteriormente. De forma general, se basa en una red *U-Net* modificada con bloques residuales.

Estructura ResUNet-a

La estructura de codificador-decodificador *ResUNet-a* modifica los bloques constructivos de *U-Net* por bloques residuales ResBlock-a **(Figura 5.22)**. Las imágenes de entrada al codificador pasan por una capa de convolución con *kernel* (1x1) normalizada (*BatchNorm*) para aumentar el número de características hasta alcanzar el tamaño de filtro inicial deseado. Después la información pasa por los bloques residuales con hasta tres convoluciones dilatadas en paralelo, además de las dos convoluciones de la arquitectura de red residual. Se suman (en lugar de concatenar) las diversas ramas de las convoluciones dilatadas. En la parte del codificador de la red, la salida de cada bloque residual se submuestrea mediante una convolución con un tamaño de filtro 1x1 y *stride* 2. Al final de las partes codificador-decodificador, existe un bloque *Pyramid scene Pooling Parsing* (PSPPooling) que las une. PSPPooling divide la entrada inicial (características) en cuatro particiones iguales, que se agrupan por una capa *max-pooling*. En cada partición se utilizan convoluciones dilatadas con diferentes coeficientes de dilatación.

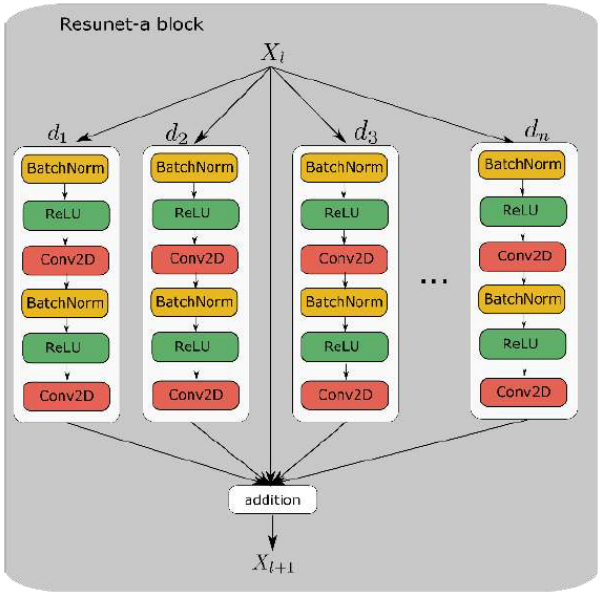


Figura 5.22: ResUNet-a. Bloque residuales

En la parte del decodificador, el sobremuestreo se realiza mediante interpolación de vecinos más cercanos, seguida de una convolución normalizada con filtro 1x1. La combinación de capas de las partes del codificador-decodificador se realiza con la capa *Combine*. Este módulo concatena las dos entradas y las somete a una convolución normalizada que lleva el número de características del número de etiquetadas deseado.

5.2.10. SwinTUNet

En el trabajo [39] proponen una metodología más novedosa, la red *SwinTUNet*, que en lugar de utilizar una red troncal basada en CNNs, utiliza un transformador visual (ViT) como extractor de características. De forma similar a la red *ResUNet-a*, la red SwinTUNet **(Figura 5.23)** parte de la red U-Net con sus *skips connections*. También modifica los bloques constructivos de U-Net, pero en lugar de usar los bloques *ResUNet-a*, utiliza los bloques *SwinU-Net*. Los bloques residuales eliminan en parte el problema de los gradientes que desaparecen o explotan, presentes en las arquitecturas profundas.

Estructura SwinT-UNet

La arquitectura general de *SwinT-Unet* propuesta en [39] consta de un codificador, un cuello de botella (*bottleneck*), un decodificador y conexiones de salto. La unidad básica de *SwinT-Unet* es el bloque *Swin Transformer*.

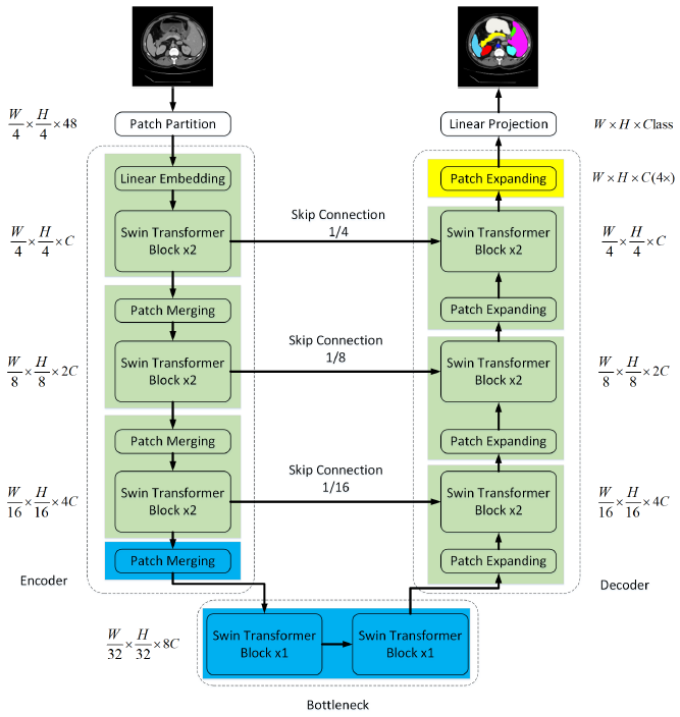


Figura 5.23: SwinT-UNet. Codificador y decodificador con bloques *Swin Transformer* en lugar de convoluciones y unidos por una capa *Bottleneck*.

Codificador

Las imágenes de entrada se dividen en parches no superpuestos de 4x4, obteniendo una salida de 4x4x3=48 características. Después, se aplica una capa *Linear Embedding* que proyecta las características extraídas a una dimensión arbitraria **(Figura 5.23 C)**. Los parches transformados pasan por varios bloques *Swin Transformer* y capas de fusión de parches (*Patch Merging*) para generar las representaciones jerárquicas de características. Las capas Patch Merging producen el *downsampling*, mientras que los bloque *Swin Transformer* son responsables del aprendizaje de la representación de características.

Bloque Swin Transformer

Cada bloque *Swin Transformer* **(Figura 5.25)** se compone de una capa *LayerNorm* (LN), un módulo de autoatención multicabezal, una conexión residual y un MLP de 2 capas con no linealidad GELU. El módulo de autoatención multicabezal basado en ventanas (W-MSA) y el módulo de autoatención multicabezal basado en ventanas desplazadas (SW-MSA) se aplican en los dos bloques sucesivos del *Bottleneck* y en cada bloque *Swin Transformer* del codificador y del decodificador.

En la *Figura 5.24* se ilustra cómo los bloques *Swin Transformer* se basan en ventanas de imagen desplazadas para calcular la autoatención. En la capa l (izquierda), se muestra una partición de ventanas normal, donde la autoatención se calcula dentro de cada ventana. En la siguiente capa $l+1$ (derecha), la partición de la ventana se desplaza, lo que da como resultado nuevas ventanas. El cálculo de la autoatención en las nuevas ventanas cruza los límites de las ventanas de la capa l, proporcionando conexiones entre ellos.

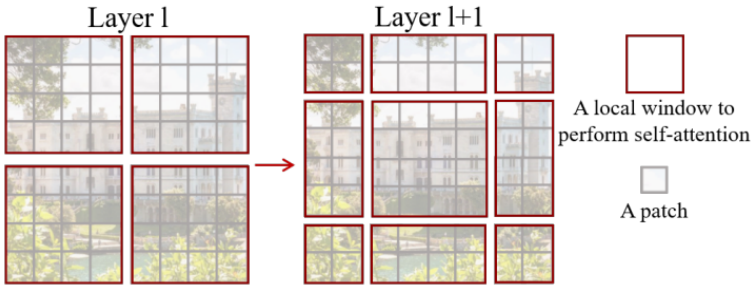


Figura 5.24: SwinT-UNet. *Swin Transformer block*.

Skip connections y Bottleneck

Como se vió en U-Net y otros modelos, las características superficiales y las profundas extraídas del codificador-decodificador se concatenan para reducir la pérdida de información espacial del codificador, seguido de una capa lineal. La dimensión de las características concatenadas se mantiene igual que la dimensión de las características de salida del *upsampling*. Como se aprecia en la *Figura 5.25*, sólo se utilizan dos bloques *Swin Transformer* sucesivos para construir el cuello de botella (*Bottleneck*) y aprender la representación de características profundas, ya que si se utilizaran más, podría no converger. En el *Bottleneck* la dimensión y la resolución de las características se mantienen sin cambios.

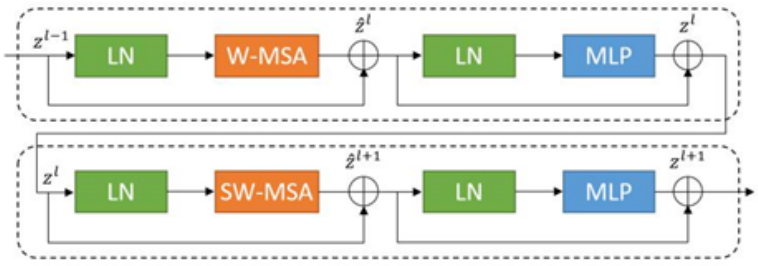


Figura 5.25: SwinT-UNet. *Bottleneck* compuesto por dos bloque consecutivos *Swin Transformer*.

Decodificador

El decodificador está compuesto por bloques *Swin Transformer* cada uno seguido de una capa de expansión de parches (*Path expanding*). Las características de contexto extraídas se fusionan con características multiescala del codificador a través de las *skip connections* para complementar la pérdida de información espacial causada por el codificador. Cada capa de expansión de parches realiza el *upsampling* a 2 veces la resolución. Al final, la última capa de *Path expanding* realiza un aumento de resolución de 4 veces, para restaurar la resolución de las imágenes de entrada.

5.2.11. SUIM-Net

En el estudio de la red *SUIM-Net* [37], se centraron en el reto de conseguir un modelo equilibrado entre rendimiento y eficiencia computacional para conseguir una inferencia rápida en el contexto de un robot submarino dirigido remotamente por percepción visual. Proponen dos arquitecturas *SUIM-Net* de tipología codificador-decodificador. Una basada en bloques de CNNs residuales y *skip connections* en el codificador y otra con codificador basado en la red *VGG16* y *skip connections*. Como en el estudio se basan en esta última y es la que se utilizará en este estudio, se describe ésta a continuación.

Estructura SUIM-Net VGG16

La red *SUIM-Net* con *backbone VGG16* **(Figura 5.26)**, tiene como entrada imágenes RGB de 320x256 píxeles, que pasan por la etapa del codificador.

La red del codificador extrae 512 mapas de características de las imágenes RGB de entrada, que son explotados por los cuatro bloques del codificador secuenciales. Cada bloque consta de dos o tres capas convolucionales y cada está enlazada por conexiones de salto a su respectiva capa del decodificador conjugado. La salida de cada bloque del codificador pasa por una capa de agrupamiento.

El decodificador tiene tres bloques iguales, formados por una capa de "deconvolución", seguida de otra de convolución con no linealidad ReLU y normalización por lotes, *Batch Normalization* (BN).

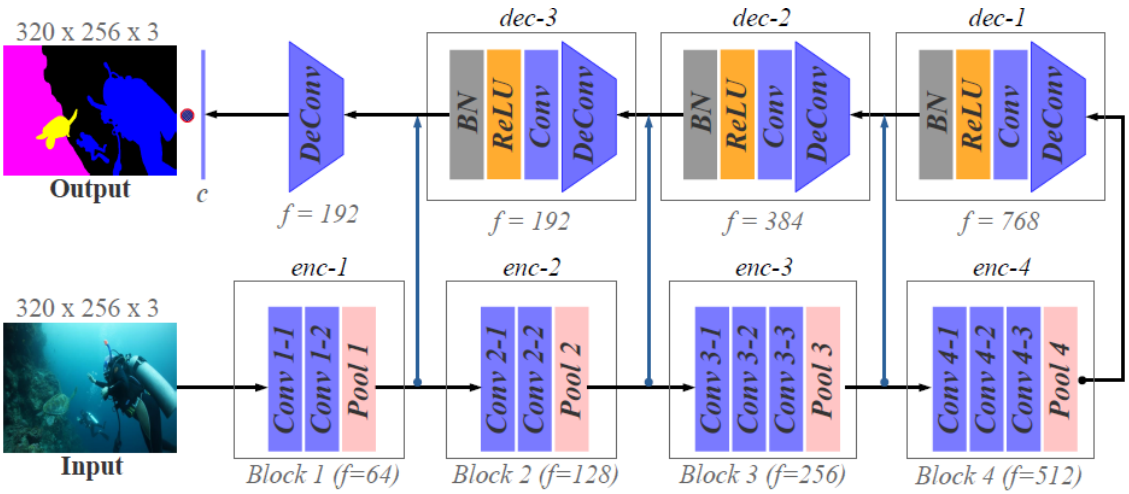


Figura 5.26: SUIM-UNet con red base VGG16 pre-entrenada como codificador y *skip connections* que la conectan con los bloques del decodificador.

La capa de convolución final genera posteriormente las etiquetas de píxel binarias por canal (*binary crossentropy*) para cada etiqueta de objeto, obteniendo un mapa de segmentación del tamaño de las imágenes de entrada y segmentando el número de etiquetas especificado.

Capítulo 6

Metodología

Una vez seleccionados los modelos a entrenar, las redes base y los conjuntos de imágenes representativos, en esta sección, inicialmente se describen las métricas para entrenamiento y evaluación, los detalles de entrenamiento de los modelos y el algoritmo para entrenamiento de los modelos en los diferentes experimentos.

6.1. Métricas de segmentación semántica visual

En esta sección se describen las métricas utilizadas en el presente estudio, teniendo en cuenta las siguientes anotaciones:

- TP: Verdaderos positivos *(True Positive)*.
- TN: Verdaderos negativos *(True Negative)*.
- FP: Falsos positivos *(False Positive)*.
- FN: Falsos negativos *(False Negative)*.

La métrica que se suele utilizar para la evaluación de modelos de segmentación semántica es la Intersección sobre la Unión (IoU, descrita en esta sección). Sin embargo, es importante tener en cuenta que aunque el IoU puede proporcionar una medida útil de rendimiento, no es la única métrica que se debe considerar al evaluar el rendimiento de un modelo de segmentación, ya que otros factores como la precisión, la sensibilidad y la puntuación F1 también pueden proporcionar información valiosa sobre el rendimiento del modelo.

6.1.1. Exactitud (*Accuracy*)

Para la evaluación del rendimiento en los entrenamientos de los modelos y combinaciones, se utiliza la métrica Exactitud **(Figura 6.1)**. Como en segmentación semántica se clasifican píxeles

y se comparan con los de la máscara verdadera en el proceso de entrenamiento, los valores de exactitud obtenidos corresponden al porcentaje de píxeles que se clasificaron correctamente para cada modelo y subconjunto de imágenes (Exactitud promedio).

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Figura 6.1: Cálculo de exactitud (*accuracy*) para una etiqueta.

6.1.2. Precisión (*PPV o Precision*)

Mide la precisión de las predicciones positivas **(Figura 6.2)**. Responde a la pregunta: "De todos los píxeles que el modelo etiquetó como positivos, ¿cuántos eran realmente positivos?".

$$Precision = \frac{TP}{TP + FP}$$

Figura 6.2: Cálculo de precisión (PPV) para una etiqueta.

6.1.3. Sensibilidad (*TPR o Recall*)

Mide la capacidad del modelo para encontrar todas los píxeles positivos **(Figura 6.3)**. Responde a la pregunta: "De todos los positivos reales, ¿cuántos identificó correctamente el modelo?".

$$Recall = \frac{TP}{TP + FN}$$

Figura 6.3: Cálculo de la sensibilidad (TPR) para una etiqueta.

6.1.4. Puntuación F1 (*F1-score*)

La media armónica de precisión y sensibilidad. Es una métrica que mide el compromiso entre precisión y sensibilidad **(Figura 6.4)**.

6.1.5. Intersección sobre Unión (*IoU*)

La Intersección sobre Unión (IoU) es la métrica de referencia utilizada para evaluar el rendimiento de algoritmos de segmentación semántica (entre otros), Cuantifica la precisión con la que un límite predicho (como el área segmentada de un objeto) coincide con el límite real y verdadero de dicho objeto. Una puntuación de 0 indica que no hay superposición ni concordancia

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

Figura 6.4: Cálculo de la puntuación F1 (*F1-score*) para una etiqueta.

entre las regiones predichas y las regiones verdaderas, lo que significa que no hay un área común entre ellas. Una puntuación de 1 indica una superposición perfecta, donde la región predicha es idéntica a la verdadera.

$$IoU = 2 (TP / (TP + FP + FN))$$

Figura 6.5: Cálculo de la Intersección sobre la Unión (*IoU*) para una etiqueta.

6.1.6. Coeficiente de Sørensen-Dice (*Dice*)

El coeficiente de Dice es una medida de la similitud entre dos conjuntos. El coeficiente oscila entre 0 y 1, donde 1 indica que los dos conjuntos son idénticos y 0 indica que los dos conjuntos no se superponen.

$$Dice = 2 ((precision * recall) / (precision + recall))$$

Figura 6.6: Cálculo de la puntuación Sørensen-Dice (*Dice*) para una etiqueta.

La Figura 6.7 muestra las diferencias entre las métricas IoU y Dice. Mientras que IoU representa la proporción de solapamiento entre dos regiones entre la unión de las mismas, el coeficiente Dice representa dos veces el solapamiento entre la suma de las dos regiones por separado.

6.2. Entrenamiento

6.2.1. Ajuste fino (*Fine Tuning*)

Siguiendo la bibliografía, todos los modelos utilizados en este estudio que tienen red troncal o *backbones* para extracción de características, son modelos pre-entrenados con el conjunto de imágenes Imagenet. Por lo tanto, todos los *backbones* se inicializan con dichos *pesos*, técnica llamada Ajuste Fino (*Fine Tuning*). Los modelos que no utilizan red base se entrenan de extremo a extremo.

6.2.2. Optimización

La selección de la técnica de optimización en el proceso de entrenamiento es crucial a la hora de obtener buenos resultados. La optimización consiste en algoritmos que minimizan o

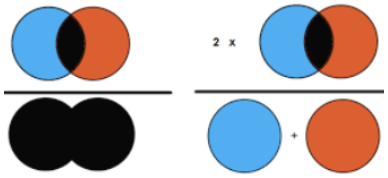


Figura 6.7: Representación de conceptos IoU y Dice. Izqda.: IoU. Dcha.: Dice.

maximizan una función objetivo (función de pérdida), ajustando parámetros internos como los *pesos* y el sesgo del modelo. Estos parámetros se actualizan iterativamente para acercarse a una solución óptima que minimice la función de pérdida. Los métodos de optimización se clasifican en dos categorías:

- **Algoritmos de primer orden**: Utilizan la derivada de primer orden para determinar si la función de pérdida está disminuyendo. El más común es el Descenso de Gradiente (GD por sus siglas en inglés).
- **Algoritmos de segundo orden**: Emplean la derivada de segundo orden (hessiano) para evaluar la minimización de la función. Sin embargo, su cálculo es complejo y se utiliza menos.

Los optimizadores más utilizados en la bibliografía corresponden a variantes del **Descenso de Gradiente**: **Descenso de Gradiente Estocástico** (SGD) y **Adam**. A continuación se resume el funcionamiento de estos algoritmos de minimización.

- **GD**: calcula el gradiente en todo el conjunto de datos disponible hasta encontrar el valor mínimo, para luego, actualizar los parámetros internos del modelo. Como realiza el cálculo sobre todo el conjunto de datos, puede provocar que si éste es demasiado grande, el proceso pueda ser lento y difícil de controlar por producirse una única actualización de los parámetros hasta la convergencia de la función.
- **SGD**: Al igual que el algoritmo GD, calcula el gradiente pero en este caso a partir de grupos pequeños de datos del conjunto de entrenamiento, en lugar de con todo el conjunto. Estos grupos pequeños se definen en el tamaño de lote o *batch* en los que se divide una época de entrenamiento. Este provoca que la actualización de parámetros no se realice una vez, sino que se haga por cada grupo de datos o *batch*, reduciendo el coste computacional.
- **Adam**: se considera una adaptación de SGD, pero en este caso, se realiza una variación de la tasa de aprendizaje o *learning rate* a lo largo del entrenamiento. Esto permite ir adaptando su comportamiento hasta encontrar o acercarse al mínimo.

Como en este estudio los modelos que se están utilizando son pre-entrenados, es decir, no se entrenan desde cero, es de interés que la tasa de aprendizaje sea adaptativa, y se evite dar saltos excesivos a la hora de localizar el mínimo. Por este motivo se selecciona el optimizador **Adam**. Además se utiliza la función *ReduceLROnPlateau* para reducir aún más la tasa de aprendizaje para cuando el algoritmo se estanca y deja de aprender un determinado número de épocas. También se incluye la función *EarlyStopping* que termina el entrenamiento cuando el algoritmo deja de aprender. En este caso, se cargan los pesos obtenidos en la mejor época. Esto evita tener que estar guardando pesos y seleccionar el que mejor rendimiento obtuvo, además de que reduce el tiempo de entrenamiento.

6.2.3. Función de pérdida

Una función de pérdida (o función de costo) en un modelo CNN, es una métrica que mide qué tan bien el modelo está realizando sus predicciones en comparación con los valores reales (etiquetas). Su objetivo es cuantificar el error entre las predicciones del modelo y los datos verdaderos, proporcionando una orientación, que guía el proceso de optimización (p. e. el descenso de gradiente) para ajustar los pesos de la red y mejorar su rendimiento.

En una CNN, la función de pérdida se calcula durante el entrenamiento y se minimiza iterativamente. Para problemas de segmentación semántica, las funciones más comunes son *categorical crossentropy* y *sparse categorical crossentropy*. La diferencia entre una y otra es la forma en la que se representan las etiquetas y las predicciones:

- **categorical crossentropy**: las etiquetas están en formato *one-hot*. Esto significa que si hay 3 clases, las etiquetas se representan como vectores binarios. Por ejemplo:
  - Clase 0: [1, 0, 0]
  - Clase 1: [0, 1, 0]
  - Clase 2: [0, 0, 1]
- **sparse categorical crossentropy**: Las etiquetas son números enteros que representan directamente la clase. Por ejemplo:
  - Clase 0: 0
  - Clase 1: 1
  - Clase 2: 2

Las máscaras del conjunto de imágenes no están en formato *one-hot*, es decir cada píxel corresponde a un número que es el identificador de clase, además la última capa de la CNN utiliza la función de activación *softmax* porque no es una segmentación binaria, en todos los casos, se segmentan más de dos etiquetas. Por lo tanto, se utilizan la función de pérdida *sparse categorical crossentropy*.

6.3. Implementación de algoritmo de entrenamiento

Los modelos utilizados en este estudio se encuentran en carpetas, por lo tanto, deben importarse al inicio del algoritmo junto con los requisitos. A continuación se describen los pasos generales que contiene el algoritmo que servirá como base para todos los entrenamientos. Para cada modelo y entrenamiento se utiliza dicho algoritmo almacenado en forma de *Jupyter Notebook* .

1. Se definen las rutas donde se almacena el conjunto de imágenes.

```
TRAIN_DIR = "./rellis20/train"
VALID_DIR = "./rellis20/val"
TEST_DIR = "./rellis20/test"

train_images = sorted(glob(os.path.join(TRAIN_DIR, "images/*")))
train_masks = sorted(glob(os.path.join(TRAIN_DIR, "labels/*")))
val_images = sorted(glob(os.path.join(VALID_DIR, "images/*")))
val_masks = sorted(glob(os.path.join(VALID_DIR, "labels/*")))
test_images = sorted(glob(os.path.join(TEST_DIR, "images/*")))
test_masks = sorted(glob(os.path.join(TEST_DIR, "labels/*")))

print(len(train_images))
print(len(train_masks))
print(len(val_images))
print(len(val_masks))
print(len(test_images))
print(len(test_masks))
```

2. Se define el tamaño de entrada necesario para la red que se entrena, el número de clases, el tamaño de lote (8 en todos los modelos), el *backbone* utilizado, los pesos para inicializar la red *Imagenet* y los nombres categóricos de las etiquetas.

```
IMAGE_SIZE = 320
BATCH_SIZE = 8
NUM_CLASSES = 20
BACKBONE_NAME = "vgg16"
WEIGHTS = "imagenet"

CLASSES = ["void", "dirt", "grass", "tree", "pole", "water", "sky", "vehicle",
           "object", "asphalt", "building", "log", "person", "fence", "bush",
           "concrete", "barrier", "puddle", "mud", "rubble"]
```

3. Creación de los subconjuntos de imágenes *train*, *val* y *test* de imágenes RGB y sus máscaras por separado, cambiando su tamaño al definido como entrada al modelo que se entrena.

```
def read_image(image_path, mask=False):
    image = tf_io.read_file(image_path)
    if mask:
        image = tf_image.decode_png(image, channels=1)
        image.set_shape([None, None, 1])
        image = tf_image.resize(images=image, size=[IMAGE_SIZE, IMAGE_SIZE])
        image = tf.round(image)
    else:
        image = tf_image.decode_png(image, channels=3)
        image.set_shape([None, None, 3])
        image = tf_image.resize(images=image, size=[IMAGE_SIZE, IMAGE_SIZE])
    return image


def load_data(image_list, mask_list):
    image = read_image(image_list)
    mask = read_image(mask_list, mask=True)
    return image, mask


def data_generator(image_list, mask_list, batch_size):
    dataset = tf_data.Dataset.from_tensor_slices((image_list, mask_list))
    dataset = dataset.map(load_data, num_parallel_calls=tf_data.AUTOTUNE)
    dataset = dataset.batch(batch_size, drop_remainder=True)
    return dataset

train_dataset = data_generator(train_images, train_masks,BATCH_SIZE)
val_dataset = data_generator(val_images, val_masks,BATCH_SIZE)
test_dataset = data_generator(test_images, test_masks, 1)     # Batchsize=1

print("Train Dataset:", train_dataset)
print("Val Dataset:", val_dataset)
print("Test Dataset:", test_dataset)
```

4. Se crea el modelo, conectando la CNN base al modelo de segmentación e inicializando con los pesos de *Imagenet*, si el modelo utiliza *backbone* como extractor de características.

```
base_model, layers, layer_names = tasm.create_base_model(
    name=BACKBONE_NAME,
    weights=WEIGHTS,
    height=IMAGE_SIZE,
    width=IMAGE_SIZE,
    include_top=False,
    pooling=None)

BACKBONE_TRAINABLE = False

...
class UNet(tf.keras.Model):
    def __init__(self, n_classes, base_model, output_layers, height=None, width=None, filters=128,
                 final_activation="softmax", backbone_trainable=False,
                 up_filters=[32, 64, 128, 256, 512], include_top_conv=True, **kwargs)
...
model = tasm.UNet(n_classes=NUM_CLASSES,
                  base_model=base_model,
                  output_layers=layers,
                  backbone_trainable=BACKBONE_TRAINABLE,
                  include_top_conv=True)
```

5. Compilación. Se selecciona el optimizador, la función de coste y los hiperparámetros de entrenamiento, como se definió en la **sección 3.1**, para posteriormente compilar el modelo.

```
opt=tf.keras.optimizers.Adam(learning_rate=0.001, epsilon=1e-7)
model.compile(optimizer=opt,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor = 'val_accuracy',
                                                 mode = 'max',
                                                 min_delta = 0.01,
                                                 patience = 3,
                                                 factor = 0.25,
                                                 verbose = 1,
                                                 cooldown = 0,
                                                 min_lr = 0.00000001)

early_stopper = tf.keras.callbacks.EarlyStopping(monitor = 'val_accuracy',
                                                 mode = 'max',
                                                 min_delta = 0.005,
                                                 patience = 10,
                                                 verbose = 1,
                                                 restore_best_weights = True)
```

6. Evaluación base. Antes de entrenar el modelo se evalúa la CNN para tener una línea base de *Accuracy* para cada modelo sin entrenar.

Model evaluation: only inference

```
# No trained model
# train subset
train_loss, train_acc = model.evaluate(train_dataset)
print(f"Train accuracy: {train_acc*100:.3f}")

#val subset
val_loss, val_acc = model.evaluate(val_dataset)
print(f"Val accuracy: {val_acc*100:.3f}")

#test subset
test_loss, test_acc = model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc*100:.3f}")
```

7. **Entrenamiento 1**. En este primer entrenamiento se entrena el modelo durante 5 épocas. Se utiliza inicialmente para verificar que el modelo "aprende" y no hay ningún error. Además antes del entrenamiento se congelan las capas que son del tipo "funcional". Este tipo de capas se usa en modelos no secuenciales complejos, es decir, permiten que se use un modelo completo como si fuera una capa para enlazarla a otra red.

En el caso de que no exista capa funcional, el modelo se entrena 5 épocas, para ajustar los pesos de la parte de la red semántica al nuevo conjunto de datos, en el caso de modelos que se inicializan con *Imagenet*. Por ejemplo, en la red U-Net con *backbone* VGG16, la capa funcional se define con valor "functional_X=false". En el primer entrenamiento se guardan los pesos y se vuelve a evaluar el modelo para cada subconjunto.

1.A. Training with Backbone not trainable ¶

Set Backbone trainable to False.

```
for layer in model.layers:
    if "functional" in layer.name:
        layer.trainable = False

    print(layer.name + ": " + str(layer.trainable))
```

```
up_sampling2d: True
functional_1: False
convolution_bn_activation: True
convolution_bn_activation_1: True
pam__module: True
cam__module: True
convolution_bn_activation_5: True
convolution_bn_activation_6: True
dropout: True
dropout_1: True
dropout_2: True
convolution_bn_activation_7: True
convolution_bn_activation_8: True
convolution_bn_activation_9: True
concatenate_1: True
concatenate_2: True
convolution_bn_activation_10: True
```

```
EPOCHS = 5
history = model.fit(train_dataset,
                    epochs=EPOCHS,
                    validation_data=val_dataset,
                    callbacks=[early_stopper, reduce_lr]
                    )
```

```
Epoch 1/5
825/825 ━━━━━━━━━━━━━━━━━━━━ 2363s 3s/step - accuracy: 0.8911 - loss: 1.1864 - val_accuracy: 0.6701 - val_loss: 1.2993 - learning_rate: 0.0010
```

8. **Entrenamiento 2**. Se descongelan todas las capas del modelo (excepto el *backbone*) y se entrena durante 30 épocas, se guardan los pesos y se vuelve a evaluar el modelo para cada subconjunto.

1.B. Train completly unfreezed model with train and test data ¶

Make whole model trainable and use validation set.

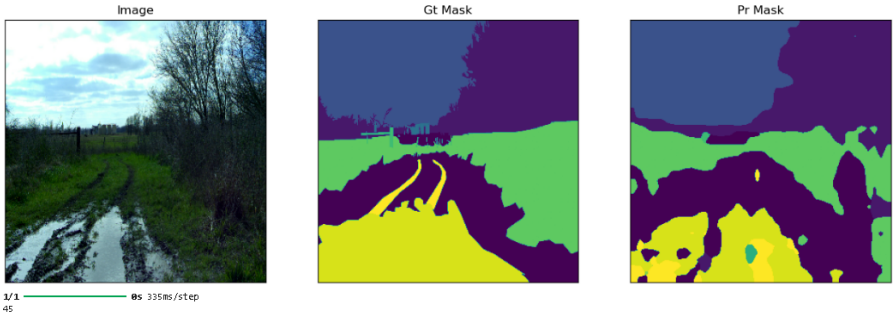
```
for layer in model.layers:
    layer.trainable = True

    print(layer.name + ": " + str(layer.trainable))
```

```
up_sampling2d: True
functional_1: True
convolution_bn_activation: True
convolution_bn_activation_1: True
pam__module: True
cam__module: True
convolution_bn_activation_5: True
convolution_bn_activation_6: True
dropout: True
dropout_1: True
dropout_2: True
convolution_bn_activation_7: True
convolution_bn_activation_8: True
convolution_bn_activation_9: True
concatenate_1: True
concatenate_2: True
convolution_bn_activation_10: True
```

```
EPOCHS = 30
history = model.fit(train_dataset,
                    epochs=EPOCHS,
                    validation_data=val_dataset,
                    callbacks=[early_stopper, reduce_lr]
                    )
```

```
Epoch 1/30
825/825 ━━━━━━━━━━━━━━━━━━━━ 2246s 3s/step - accuracy: 0.9403 - loss: 0.1978 - val_accuracy: 0.7894 - val_loss: 0.5710 - learning_rate: 0.0010
```

9. Visualización de inferencia. Se selecciona un conjunto de imágenes de muestra del conjunto de test, se realiza inferencia con el modelo con los pesos cargados durante 5+30 épocas y se muestran las imágenes comparando la imagen RGB, la máscara real y la predicción.



10. Inferencia en conjunto de test completo. Se realiza la inferencia en el subconjunto de test y de las máscaras *ground truth* para compararlas con las predicciones.

11. Extracción de métricas. Se extraen las métricas de inferencia para el conjunto de test y el *ground truth* guardando en ficheros ".csv".

All Test dataset inference

```
def infer(model, image_tensor):
    predictions = model.predict(np.expand_dims((image_tensor), axis=0), verbose=0)
    predictions = np.squeeze(predictions)
    predictions = np.argmax(predictions, axis=2)
    return predictions

def inference_maskID(RGB_image_list, model):
    aux=0
    pred_masks_id=[]
    for image_file in RGB_image_list:
        image_tensor = read_image(image_file)                          # Read RGB test image
        pred_mask_id = infer(image_tensor=image_tensor, model=model)   # Predicted MaskID
        pred_masks_id.append(pred_mask_id)
        aux+=1
    return pred_masks_id

pred_masksID = inference_maskID(test_images, model)
```

```
df = pd.concat([df, isPresent],axis=1)
folder='./inference_metrics/rellis5/'
df.to_csv(folder + 'SEM_DANET_EFFNET_Rellis5_Metrics.csv',index=False)

mPPV=pd.DataFrame(PPV_list.mean(axis=0))
mPPV.to_csv(folder + 'SEM_DANET_EFFNET_Rellis5_mPPV.csv', index=False)

mTPR=pd.DataFrame(TPR_list.mean(axis=0))
mTPR.to_csv(folder + 'SEM_DANET_EFFNET_Rellis5_mTPR.csv', index=False)

mF1=pd.DataFrame(F1_list.mean(axis=0))
mF1.to_csv(folder + 'SEM_DANET_EFFNET_Rellis5_mF1.csv', index=False)

mIoU=pd.DataFrame(IoU_list.mean(axis=0))
mIoU.to_csv(folder + 'SEM_DANET_EFFNET_Rellis5_mIoU.csv', index=False)

mDice=pd.DataFrame(Dice_list.mean(axis=0))
mDice.to_csv(folder + 'SEM_DANET_EFFNET_Rellis5_mDice.csv', index=False)
```

Todos los pasos descritos en esta sección se aplicarán a todos los modelos entrenados en este estudio, modificando en cada caso el modelo de segmentación, el *backbone* (si tiene), el conjunto de imágenes de entrenamiento, y las rutas para guardar los pesos y las métricas.

En todos los entrenamientos se utilizan como métrica el *Accuracy* obtenido en el conjunto de validación. Esto es debido a que, como hay muchos modelos para evaluar, interesa acelerar el tiempo de entrenamiento, puesto que el uso de otras métricas requiere mayor tiempo de procesamiento. Posteriormente se realiza la inferencia del conjunto de test completo y las imágenes de verdad (*ground truth*) para obtener las métricas: mDice, mF1, mIoU, mPPV y mTPR y su posterior evaluación.

Capítulo 7

Resultados Experimentales

En este capítulo se abordan varias estrategias aplicadas a la segmentación semántica de imágenes en entornos no estructurados. El objetivo es plantear diferentes condiciones para ver como afectan a los resultados de segmentación de imágenes. En primer lugar, a partir de los modelos del Estado del Arte y analizados en el *Capítulo 5* se entrena el conjunto de imágenes Rellis-3D con 20 etiquetas, lo cual permite evaluar el rendimiento de los modelos en las mismas condiciones **(sección 7.1)**. En segundo lugar, con los mejores modelos evaluados se entrenan los tres conjuntos de imágenes seleccionados y vistos en el *Capítulo 4*. Esto permitirá evaluar el rendimiento de un mismo modelo en conjuntos de datos diferentes y cómo las propias condiciones de adquisición de las imágenes y los diferentes entornos afecta a la calidad de la segmentación (**sección 7.2**). Por último, con el fin de tener un conjunto de imágenes mayor, con entornos más variados y plataformas de adquisición diferentes, se unen los tres conjuntos de imágenes Rellis-3D, RUGD, Goose con Goose-Ex para formar un único conjunto de imágenes denominado OFFROAD. Este conjunto de imágenes ampliado se entrena con los modelos seleccionados para evaluar si los modelos son capaces de generalizar bien cuando hay diferentes plataformas de adquisición de imágenes, entornos diferentes y si se mejora el problema mencionado de etiquetas escasamente representadas (**sección 7.3**).

7.1. Experimento 1: comparativa de modelos entrenados con Rellis-3D de 20 etiquetas

Para esta prueba se selecciona el conjunto de datos Rellis-3D original con 20 etiquetas para entrenar los 15 modelos con las redes base seleccionadas **(sección 5.1.2)**. Se descarta RUGD porque está obtenido por el mismo equipo que Rellis-3D y este último contiene mejoras. A fecha de inicio de este estudio las imágenes de GOOSE todavía no estaban disponibles.

7.1.1. Pre-procesado de Rellis-3D

Rellis-3D original contiene 5957 imágenes RGB y 5957 máscaras con los siguientes valores de píxel (IDs) para cada etiqueta (CLASSES):

- IDs = [0, 1, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 18, 19, 23, 27, 31, 33, 34]
- CLASSES = [*void, dirt, grass, tree, pole, water, sky, vehicle,object, asphalt, building, log, person, fence, bush,concrete, barrier, puddle, mud, rubble*]

Para que los IDs de las máscaras sean correlativos, se generan nuevas máscaras con valores de ID de 0 a 19 manteniendo el orden de las clases. Posteriormente se divide el conjunto de imágenes en entrenamiento (*train*), validación (*val*) y test (*test*), como se describe en la *Figura 7.1.*

	Nº de Imágenes RELLIS-3D (1900 x 1200 pxs)		
Subconjunto	**Train**	**Val**	**Test**
RGB	3302	983	1672
Máscaras	3302	983	1672

Figura 7.1: RELLIS-3D original. División del conjunto de imágenes.

Siguiendo el algoritmo descrito anteriormente, se entrenan los 15 modelos con Rellis-3D de 20 etiquetas pre-procesado. Una vez obtenidos y guardados los pesos y métricas de los modelos, se realiza el análisis de resultados que se detalla a continuación.

7.1.2. Entrenamiento

Después del entrenamiento de los 15 modelos seleccionados con y sin red troncal, se obtienen 37 combinaciones. La *Tabla 7.2* muestra los *Accuracy* obtenidos en los 37 entrenamientos con 5 y 30 épocas para los conjuntos *train* y *val*. Se puede apreciar que todos los modelos (excepto U-Net) con sólo 5 épocas superan el 60 % *Accuracy* en los subconjuntos *train* y *val*, obteniendo un máximo de 89,9 % y 84,1 % *Accuracy* en los subconjuntos *train* y *val* respectivamente.

Después de entrenar 30 épocas más, el rendimiento mejora y se obtiene un mínimo de 78,5 % y 78,0 % *Accuracy* y un máximo de 94,4 % y 87,3 % *Accuracy* en los subconjuntos *train* y *val* respectivamente. Todos los modelos mejoran cuanto mayor es el número de épocas de entrenamiento, excepto FP-Net+ResNet50 que empeora (78,8 % a 78,0 % Accuracy en val) y una ligera mejora en CF-Net+EfficientNetb3 (84,1 % a 84,2 % *Accuracy* en *val*) y ASPOCR-Net+ResNet50 (79,7 % a 81,3 % *Accuracy* en *val*). Estos modelos pueden ser interesantes si se necesita un tiempo de entrenamiento corto, pues con sólo 5 épocas se alcanza un rendimiento de entrenamiento óptimo. Cabe destacar que se ha utilizado un criterio de parada de entrenamiento, para que si

el algoritmo deja de aprender (no mejora *Accuracy* en validación) el algoritmo se detenga, por lo que no todos los modelos han agotado las 30 épocas de entrenamiento.

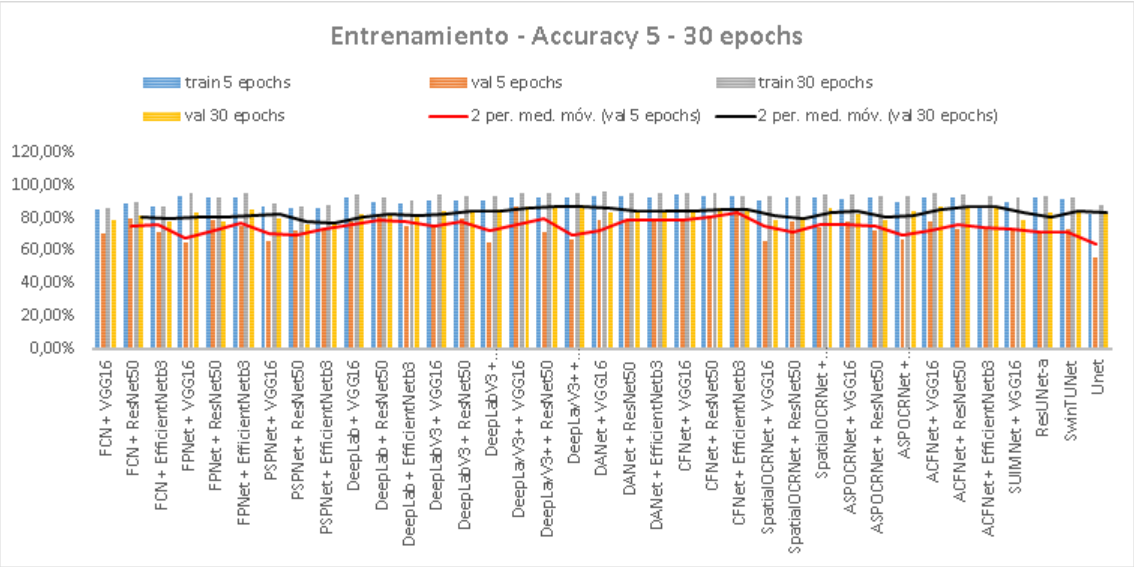


Figura 7.2: Rellis-3D 20. *Accuracy* obtenido en el entrenamiento de modelos con y sin backbones para *train* y *val*.

La *Figura 7.3* muestra la línea base (verde) que corresponde al *Accuracy* obtenido cuando se han cargado los pesos en el backbone con Imagenet si el modelo utiliza red CNN base y sin entrenar se evalúa en el conjunto de *test*. Con 5 épocas de entrenamiento (azul) se aprecia una mejora significativa respecto a la línea base, lo cuál indica que el modelo aprende de los datos. Tras las 30 épocas de entrenamiento (amarillo) se comprueba que de forma general el rendimiento mejora. De forma similar a la evaluación de los subconjuntos *train* y *val*, en *test* se aprecia que 4 modelos (FC-Net+ResNet50, FP-Net+ResNet50, DeepLab+VGG16 y CF-Net+EfficientNetb3) consiguen un *Accuracy* prácticamente igual en el entrenamiento de 5 y 30 épocas, por lo que pueden ser interesantes si es necesario entrenamiento rápido. Los tres mejores son FPNet+EfficientNetb3, DeepLabV3+ y CF-Net con ResNet50 con 90,6 %, 90,4 % y 90,2 % *Accuracy* en *test* respectivamente.

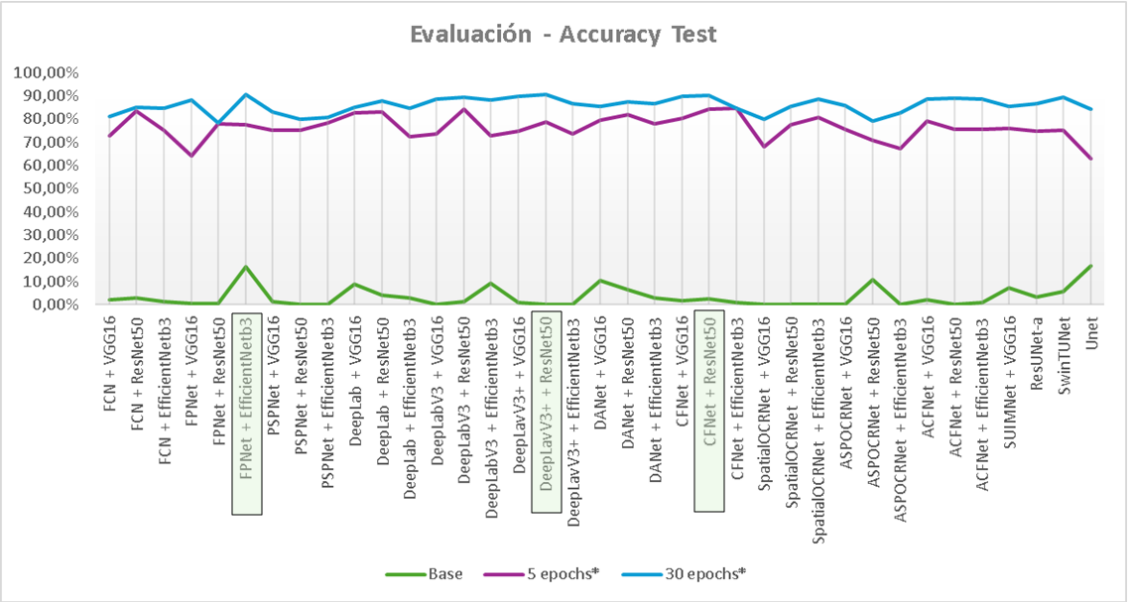


Figura 7.3: Rellis-3D 20. *Accuracy* obtenido en el entrenamiento de modelos con y sin backbones para el subconjunto de *test*.

Si se analiza el rendimiento de los modelos entrenados según si utiliza o no red base, se puede estudiar la contribución que aporta o no al modelo a la hora de extraer características, y cuál funciona mejor para el conjunto de datos entrenado. En la *Figura 7.4*, se muestran las gráficas de las líneas base de la métrica *Accuracy* obtenidas en los 15 modelos con y sin *backbones*. La gráfica de la izquierda muestra 15 modelos, tres de ellos (U-Net, ResU-Net y SwinTU-Net) se han incluído en la gráfica pero no utilizan el *backbone* VGG16, pero el resto sí. La gráfica central y la derecha, muestran 11 modelos con redes troncales EfficientNetb3 y ResNet50 respectivamente.

La mayor parte de los modelos presentan una línea base entorno a cero y en algunos casos alcanza entorno al 10 % y el 15 %. Los modelos que superan el 5,0 % Accuracy en VGG16 son: DA-Net, DeepLab y SUIM-Net, con EfficientNetb3: DeepLabV3 y FP-Net, con ResNet50: DA-Net y DeepLab y sin uso de backbone: SwinT-UNet.

Después de 5 épocas de entrenamiento, se obtienen las gráficas mostradas en la *Figura 7.5*. Respecto a las líneas base anteriores, la gráfica izquierda (*backbone* VGG16) muestra entre el 65 % y 85 % *Accuracy*. Las redes U-Net, ResU-Net y SwinTU-Net (sin VGG16) alcanzan entorno al 70 %-80 % *Accuracy*. Con sólo 5 épocas de entrenamiento DeepLab obtiene los mejores resultados con un 82,9 % *Accuracy* en el conjunto de *test*, seguido de CF-Net 80,3 % *Accuracy*.

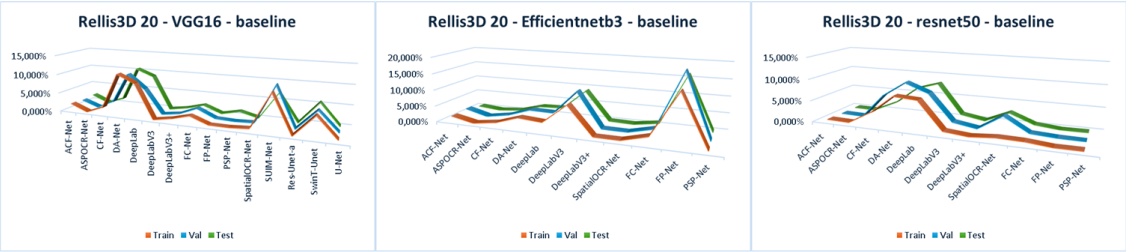


Figura 7.4: Rellis-3D 20. Lineas base (*accuracy*) de modelos sin entrenamiento. Izqda. con y sin backbone VGG16. Centro: con backbone EfficientNetb3. Dcha.: con backbone ResNet50.

De forma similar, el entrenamiento con 5 épocas de los 11 modelos con red troncal Efficient-Netb3 **(Figura 7.5. Centro)**, se alcanzan entre el 65 % y el 85 % *Accuracy*. Siendo en este caso los mejores resultados en CF-Net 84,6 % y SpatialOCR-Net con 80,7 % *Accuracy*, ambos en el conjunto de *test*. En la gráfica **(Figura 7.5. Derecha)**, se observa un mejor rendimiento entre 70 % y 85 % *Accuracy* para los modelos entrenados con ResNet50 con 5 épocas. CF-Net obtiene de nuevo los mejores resultados con 84,4 % *Accuracy* en el conjunto de *test*. Le sigue DeepLab con 84,3 % *Accuracy* también en *test*.

El entrenamiento durante 30 épocas más de los modelos anteriores **(Figura 7.6)**, mejora los resultados obtenidos. Con y sin backbone VGG16 pasa de un 65 %-85 % a un 75 %-90 % *Accuracy* para los tres subconjuntos (*train, val* y *test*). Con red troncal EfficientNetb3 la mejora es aún mayor, pasando de 65 %-85 % a un 80 %-95 % *Accuracy* en los tres subconjuntos.

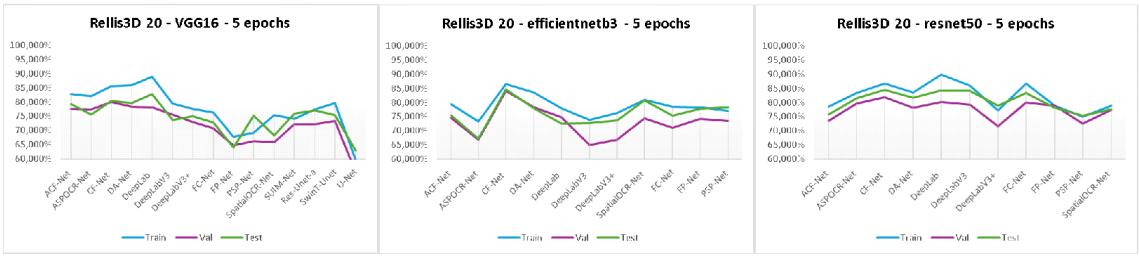


Figura 7.5: Rellis-3D 20. (*Accuracy*) de modelos con y sin backbones después de entrenar 5 épocas.

Por último, con ResNet50 se pasa de un 70 %-85 % a un 80 %-95 % *Accuracy* también en los tres subconjuntos.

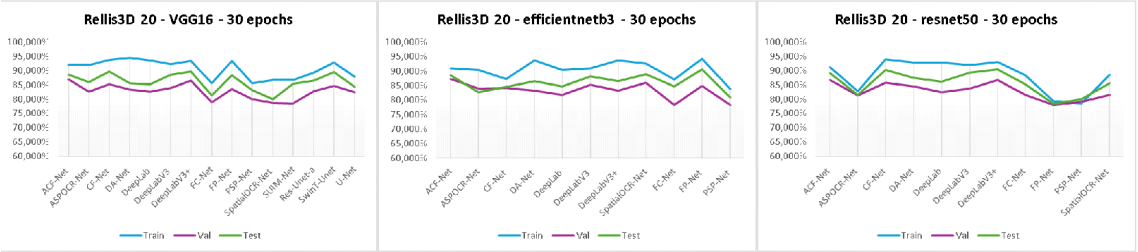


Figura 7.6: Rellis-3D 20. (*Accuracy*) de modelos con y sin backbones después de entrenar 30 épocas.

7.1.3. Inferencia

Para comparar las variantes de los modelos de segmentación semántica en el conjunto de *test*, la *Figura 7.8* recoge la métrica Intersección sobre la Unión de píxeles para cada una de las 20 clases (IoU) de Rellis-3D. Si se analiza el valor promedio IoU para cada clase de todas las variantes de los modelos, se aprecia que las clases: *void, dirt, pole, water, vehicle, object, asphalt, building, log, fence y rubble*, obtienen un IoU por clase inferior al 1,5 %. Las clases *person, barrier, puddle y mud,* mejoran ligeramente con IoU promedio entre 3-6 %. Sólo hay cinco con mIoU >30 % de todas las variantes **(Figura 7.7)** que si las comparamos con los resultados reportados en la bibliografía de referencia [52], se aprecia que sólo las clases *sky, grass, tree* y *bush* tienen resultados similares, con el resto de las 20 etiquetas se obtienen deficientes resultados. En [52] obtienen (> 30 % mIoU) en sus modelos frente al 19,90 % mIoU del mejor modelo de este experimento.

	IoU por clase - RELLIS 20					
	sky	grass	tree	bush	concrete	mIoU
HRNET+OCR	96,94%	90,20%	80,53%	76,76%	84,22%	51,55%
GSCNN	97,02%	84,95%	78,52%	70,33%	83,82%	52,92%
Experimento 1	95,54%	85,61%	68,34%	63,02%	44,25%	19,90%

Figura 7.7: Rellis-3D 20. mIoU por clase y promedio reportados en la bibliografía.

Si comparamos los valores IoU promediados de todas las variantes obtenidos **(Figura 7.8)** con la cantidad de píxeles por clase del *ground truth* de Rellis-3D, se puede apreciar que los resultados obtenidos están directamente relacionados con la cantidad de píxeles por clase. La *Figura 7.9*, recoge la cantidad de píxeles por clase del *ground truth* ordenados de mayor a menor (TP), respecto al IoU promediado de todos los modelos (IoU) para imágenes de 128x128 píxeles. En verde las 5 clases con mayor número de píxeles coincide con los mejores resultados IoU

promediados, las 5 siguientes clases con mayor número de píxeles (azúl) coincide también con los siguientes mejores resultados IoU (excepto para la clase *asphalt*) y teniendo en cuenta con los resultados son entre 3-6 % IoU. El resto de clases (gris) con valores inferiores a 1,5 % IoU coinciden con los valores más bajos de cantidad de píxeles por clase. Esto claramente indica que la mayoría de clases están escasamente representadas en el conjunto de datos Rellis-3D, y los modelos no tienen suficiente información para aprender dichas clases.

Metric IoU Rellis 20																				
Model	void	dirt	grass	tree	pole	water	sky	vehicle	object	asphalt	building	log	person	fence	bush	concrete	barrier	puddle	mud	rubble
ACFNet + EfficientNet	0,00%	0,00%	**78,94%**	**52,03%**	0,03%	0,00%	**94,23%**	2,86%	1,79%	0,23%	0,00%	0,09%	3,27%	0,05%	**33,86%**	**33,14%**	3,05%	8,25%	9,93%	2,41%
ACFNet + ResNet	0,00%	0,00%	**83,86%**	**62,64%**	0,25%	0,09%	**93,64%**	1,23%	0,27%	0,01%	0,00%	0,01%	5,36%	2,05%	**61,82%**	**39,47%**	4,42%	8,12%	10,07%	2,88%
ACFNet + VGG	0,00%	0,00%	**83,48%**	**61,66%**	0,00%	0,00%	**93,78%**	0,43%	0,84%	0,00%	0,00%	0,00%	5,54%	2,17%	**58,86%**	**35,31%**	4,33%	7,77%	8,13%	3,69%
ASPOCRNet + EfficientNet	0,00%	0,00%	**58,25%**	**30,54%**	0,02%	0,01%	**63,98%**	2,06%	0,81%	0,15%	0,01%	0,03%	1,53%	0,03%	**22,03%**	**21,66%**	2,07%	5,59%	1,95%	0,01%
ASPOCRNet + ResNet	0,00%	0,00%	**69,34%**	**31,94%**	0,02%	0,00%	**89,72%**	2,07%	3,76%	0,00%	0,00%	0,00%	1,40%	0,29%	**46,11%**	**29,73%**	3,72%	1,49%	2,09%	0,00%
ASPOCRNet + VGG	0,00%	0,00%	**80,25%**	**52,41%**	0,01%	0,00%	**92,88%**	0,44%	3,03%	0,00%	0,00%	0,00%	5,28%	1,34%	**51,95%**	**34,55%**	3,02%	5,74%	4,96%	1,28%
CFNet + EfficientNet	0,00%	0,00%	**77,64%**	**56,56%**	0,00%	0,00%	**91,90%**	3,36%	0,08%	0,00%	0,02%	0,00%	2,85%	0,04%	**54,44%**	**31,61%**	0,95%	3,71%	0,03%	0,33%
CFNet + ResNet	0,00%	0,00%	**84,59%**	**66,28%**	0,00%	0,00%	**94,62%**	0,51%	1,74%	0,00%	0,00%	0,00%	5,54%	1,54%	**61,76%**	**36,37%**	3,33%	8,84%	7,05%	0,84%
CFNet + VGG	0,00%	0,00%	**84,53%**	**64,32%**	0,00%	0,00%	**94,41%**	0,72%	1,59%	0,00%	0,00%	0,00%	5,44%	1,74%	**61,96%**	**35,18%**	1,95%	8,11%	4,91%	0,39%
DANet + EfficientNet	0,00%	0,00%	**76,19%**	**42,40%**	0,00%	0,08%	**93,44%**	4,82%	0,82%	0,08%	0,00%	0,06%	4,78%	0,60%	**9,89%**	**23,88%**	2,01%	5,70%	2,77%	1,28%
DANet + ResNet	0,00%	0,00%	**80,28%**	**60,38%**	0,00%	0,02%	**93,77%**	0,87%	0,51%	0,00%	0,00%	0,00%	5,58%	1,33%	**56,66%**	**39,20%**	2,90%	7,04%	6,47%	0,48%
DANet + VGG	0,00%	0,00%	**76,66%**	**58,37%**	0,01%	0,00%	**93,24%**	0,73%	1,24%	0,00%	0,02%	0,00%	5,48%	1,03%	**51,03%**	**35,38%**	3,09%	6,55%	5,76%	0,35%
DeepLab + EfficientNet	0,00%	0,00%	**74,38%**	**47,38%**	0,00%	0,05%	**91,13%**	1,69%	0,29%	0,24%	0,00%	0,11%	4,12%	0,41%	**24,31%**	**18,48%**	1,60%	6,49%	5,33%	1,18%
DeepLab + ResNet	0,00%	0,00%	**82,51%**	**57,35%**	0,00%	0,00%	**93,48%**	0,10%	1,80%	0,00%	0,00%	0,00%	4,61%	0,31%	**56,76%**	**33,12%**	3,79%	7,64%	6,05%	1,24%
DeepLab + VGG	0,00%	0,00%	**79,07%**	**53,92%**	0,00%	0,00%	**91,90%**	0,00%	0,33%	0,00%	0,00%	0,00%	4,62%	0,33%	**52,56%**	**26,19%**	0,96%	5,16%	2,67%	0,03%
DeepLabV3 + EfficientNet	0,00%	0,00%	**74,30%**	**57,29%**	0,00%	0,00%	**93,06%**	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	**36,41%**	**26,68%**	1,51%	6,59%	4,46%	2,20%
DeepLabV3 + ResNet	0,00%	0,00%	**84,40%**	**64,47%**	0,00%	0,10%	**93,85%**	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	**59,79%**	**36,92%**	2,13%	8,06%	8,40%	2,98%
DeepLabV3 + VGG	0,00%	0,00%	**82,83%**	**58,36%**	0,00%	0,00%	**93,40%**	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	**58,58%**	**35,83%**	1,59%	7,94%	7,72%	3,39%
DeepLabV3+ + EfficientNet	0,00%	0,00%	**84,14%**	**65,54%**	0,01%	0,02%	**95,54%**	4,93%	5,27%	0,55%	0,03%	0,00%	6,02%	1,18%	**61,04%**	**44,25%**	8,58%	8,94%	10,23%	1,43%
DeepLabV3+ + ResNet	0,00%	0,00%	**84,14%**	**65,54%**	0,01%	0,02%	**95,54%**	4,93%	5,27%	0,55%	0,03%	0,00%	6,02%	1,18%	**61,04%**	**44,25%**	8,58%	8,94%	10,23%	1,43%
DeepLabV3+ + VGG	0,00%	0,00%	**85,61%**	**64,63%**	0,01%	0,02%	**95,42%**	3,56%	2,01%	0,47%	0,03%	0,00%	5,93%	1,41%	**62,16%**	**42,73%**	8,94%	9,00%	10,23%	3,19%
FCNet + EfficientNet	0,00%	0,00%	**78,50%**	**49,29%**	0,00%	0,00%	**85,25%**	0,00%	0,00%	0,00%	0,00%	0,00%	0,24%	0,00%	**53,13%**	**25,54%**	0,35%	5,80%	0,17%	0,37%
FCNet + ResNet	0,00%	0,00%	**78,70%**	**49,70%**	0,00%	0,00%	**88,71%**	0,00%	0,00%	0,00%	0,00%	0,00%	0,02%	0,00%	**55,80%**	**15,89%**	0,39%	5,31%	1,69%	0,00%
FCNet + VGG	0,00%	0,00%	**74,87%**	**43,50%**	0,00%	0,02%	**86,36%**	0,00%	0,00%	0,01%	0,00%	0,00%	1,21%	0,00%	**46,58%**	**18,42%**	0,38%	3,37%	0,86%	0,08%
FPNet + EfficientNet	0,00%	0,00%	**84,49%**	**68,04%**	0,00%	0,08%	**95,20%**	2,93%	2,30%	2,30%	0,13%	0,00%	5,63%	1,55%	**63,02%**	**41,26%**	7,87%	10,02%	8,85%	4,35%
FPNet + ResNet	0,00%	0,00%	**63,71%**	**45,18%**	0,00%	0,00%	**90,02%**	0,09%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	**46,91%**	**36,96%**	0,28%	1,76%	2,62%	0,00%
FPNet + VGG	0,00%	0,00%	**80,93%**	**64,80%**	0,00%	0,04%	**95,36%**	3,49%	6,71%	0,43%	0,05%	0,00%	6,90%	2,97%	**57,15%**	**42,44%**	10,18%	9,10%	9,69%	3,67%
PSPNet + EfficientNet	0,00%	0,00%	**0,81%**	**0,79%**	0,00%	0,00%	**31,17%**	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	**0,00%**	**0,00%**	0,00%	0,00%	0,00%	0,00%
PSPNet + ResNet	0,00%	0,00%	**72,41%**	**35,47%**	0,00%	0,00%	**84,49%**	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	**46,02%**	**1,84%**	0,00%	0,01%	0,00%	0,00%
PSPNet + VGG	0,00%	0,00%	**77,14%**	**38,79%**	0,00%	0,04%	**85,50%**	0,00%	0,21%	0,00%	0,00%	0,00%	0,11%	0,00%	**52,43%**	**21,25%**	0,05%	3,98%	0,27%	0,00%
SpOCRNet + EfficientNet	0,00%	0,00%	**78,95%**	**62,60%**	0,00%	0,00%	**93,62%**	0,50%	1,55%	0,00%	0,00%	0,00%	4,92%	0,60%	**55,94%**	**34,36%**	2,32%	7,75%	6,13%	2,48%
SpOCRNet + ResNet	0,00%	0,00%	**78,37%**	**56,91%**	0,00%	0,00%	**92,15%**	0,00%	0,00%	0,00%	0,00%	0,00%	0,09%	0,00%	**56,60%**	**33,94%**	0,50%	2,31%	1,80%	0,00%
SpOCRNet + VGG	0,00%	0,00%	**72,39%**	**45,07%**	0,00%	0,01%	**88,22%**	0,10%	0,03%	0,00%	0,00%	0,00%	3,33%	0,01%	**40,92%**	**25,97%**	0,93%	2,18%	2,31%	0,32%
SUIMet + VGG	0,00%	0,00%	**78,98%**	**43,67%**	0,04%	0,01%	**92,02%**	0,77%	0,20%	0,20%	0,13%	0,00%	2,92%	0,61%	**51,61%**	**34,89%**	4,87%	6,76%	3,16%	0,42%
Unet	0,00%	0,00%	**75,29%**	**57,74%**	0,00%	0,09%	**94,91%**	0,51%	0,95%	5,60%	0,16%	0,01%	4,95%	0,70%	**46,10%**	**36,84%**	12,73%	5,42%	4,61%	2,08%
ResUNet	0,00%	0,00%	**79,92%**	**62,63%**	0,00%	0,13%	**94,42%**	0,52%	0,48%	0,64%	0,00%	0,00%	4,95%	0,13%	**54,53%**	**21,66%**	3,38%	8,67%	5,05%	0,01%
SwinT-Unet	0,00%	0,00%	**83,20%**	**68,34%**	0,00%	0,29%	**94,83%**	1,27%	0,19%	6,24%	0,20%	0,02%	4,85%	0,95%	**61,83%**	**38,88%**	5,61%	8,40%	9,70%	1,41%
Promedio	0,00%	0,00%	**76,33%**	**53,15%**	0,01%	0,03%	**89,87%**	1,23%	1,19%	0,48%	0,02%	0,01%	3,34%	0,66%	**49,50%**	**30,65%**	3,31%	6,12%	5,04%	1,25%

Figura 7.8: Rellis-3D 20. Número de píxeles por clase del ground truth (TP), respecto al IoU promediados de todas las variantes de los modelos obtenidos en la inferencia del conjunto de *test*.

Class	*grass*	*sky*	*bush*	*tree*	*concrete*	*puddle*	*mud*	*barrier*	*asphalt*	*person*	*object*	*vehicle*	*fence*	*rubble*	*pole*	*water*	*building*	*log*	*dirt*	*void*
TP	60649507	51971721	29456928	18729472	5632074	2382791	1324984	204507	171454	142753	120276	96913	91556	84780	81114	42605	27567	1348	427	23
IoU	76,33%	89,87%	49,50%	53,15%	30,65%	6,12%	5,04%	3,31%	0,48%	3,34%	1,19%	1,23%	0,66%	1,25%	0,01%	0,03%	0,02%	0,00%	0,00%	0,00%

Figura 7.9: Rellis-3D 20. (IoU) de todas las variantes de los modelos obtenidos en la inferencia del conjunto de *test*.

Para valorar si el uso o no de red troncal para extraer características es relevante, se analizan los IoU por clase separados por las redes base VGG16, ResNet50 y EfficientNetb3 **(Figura 7.10)**. De nuevo, puede observarse que sólo 5 clases son segmentables de forma destacable (*grass, tree,*

sky, bush y *concrete*). Los mejores resultados IoU de los modelos para cada *backbone* obtienen resultados similares.

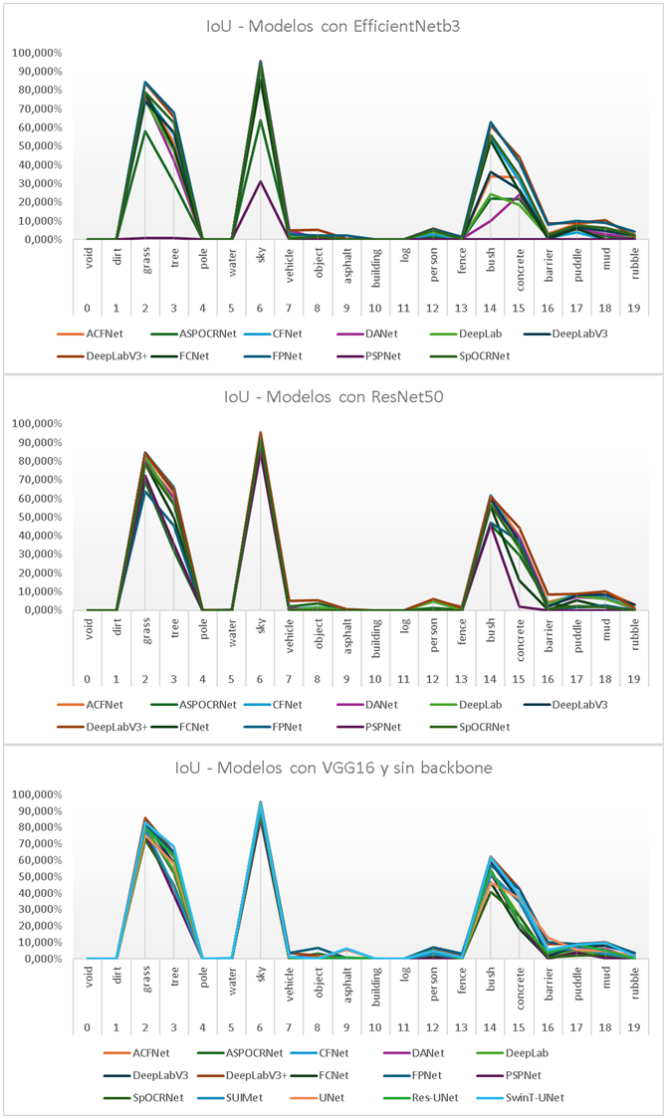


Figura 7.10: Rellis-3D 20. (*mIoU* por clase) según *backbone* utilizado en los modelos en la inferencia del conjunto de *test*.

Para la clase *sky*, todos los modelos sin red troncal, con ResNet50 y VGG16 obtienen un IoU superior al 80 % IoU, pero con EfficientNetb3 los modelos ASPOCR-Net y PSP-Net no lo alcanzan.

En la clase *grass*, ASPOCR-Net y PSP-Net con EfficientNetb3 no superan el 70 % IoU al igual que ASPOCR-Net y FP-Net con ResNet50. Estos modelos con VGG16 y el resto de modelos con y sin *bakcbone* superan el 70 % IoU. De forma similar, los modelos ASPOCR-Net y PSP-Net con EfficientNetb3 y ResNet50, y PSP-Net con VGG16, no superan el 40 % IoU para la clase *tree*, sí lo superan el resto de variantes. Para el resto de clases, existe una variabilidad similar.

7.1.4. Resumen Resultados

A partir de los resultados cuantitativos obtenidos en los entrenamientos de las 37 variantes de los 15 modelos elegidos y la inferencia de estos en el conjunto de test, se seleccionan 5 modelos para utilizarse en el resto de experimentos.

Modelo	Acc. 30 e
FPNet + EfficientNetb3	90,55%
DeepLavV3+ + ResNet50	90,44%
CFNet + ResNet50	90,15%
DeepLavV3+ + VGG16	89,73%
CFNet + VGG16	89,67%
SwinTUNet	89,46%
DeepLabV3 + ResNet50	89,25%
ACFNet + ResNet50	89,19%
SpOCRNet + EfficientNetb3	88,81%
ACFNet + VGG16	88,62%
ACFNet + EfficientNetb3	88,57%
DeepLabV3 + VGG16	88,47%
FPNet + VGG16	88,42%
DeepLabV3 + EfficientNetb3	88,22%
DeepLab + ResNet50	87,88%
DANet + ResNet50	87,39%
ResUNet-a	86,52%
DeepLavV3+ + EfficientNetb3	86,47%
DANet + EfficientNetb3	86,47%
ASPOCRNet + VGG16	85,91%
SpOCRNet + ResNet50	85,59%
DANet + VGG16	85,55%
SUIMNet + VGG16	85,38%
DeepLab + VGG16	85,22%
FCN + ResNet50	85,18%
FCN + EfficientNetb3	84,67%
DeepLab + EfficientNetb3	84,63%
CFNet + EfficientNetb3	84,52%
Unet	84,32%
PSPNet + VGG16	83,10%
ASPOCRNet + EfficientNetb3	82,76%
FCN + VGG16	81,22%
PSPNet + EfficientNetb3	80,89%
SpOCRNet + VGG16	80,09%
PSPNet + ResNet50	79,94%
ASPOCRNet + ResNet50	79,10%
FPNet + ResNet50	78,32%

Modelo	mIoU
FPNet + EfficientNetb3	19,90%
DeepLavV3+ + ResNet50	19,89%
DeepLabV3+ + EfficientNetb3	19,89%
DeepLabV3+ + VGG16	19,77%
FPNet + VGG16	19,70%
SwinT-Unet	19,31%
ACFNet + ResNet50	18,81%
CFNet + ResNet50	18,65%
ACFNet + VGG16	18,30%
CFNet + VGG16	18,26%
DeepLabV3 + ResNet50	18,06%
DANet + ResNet50	17,77%
SpOCRNet + EfficientNetb3	17,59%
DeepLabV3 + VGG16	17,48%
DeepLab + ResNet50	17,44%
Unet	17,43%
DANet + VGG16	16,95%
ASPOCRNet + VGG16	16,86%
ResUNet	16,86%
ACFNet + EfficientNetb3	16,21%
CFNet + EfficientNetb3	16,18%
SpOCRNet + ResNet50	16,13%
SUIMet + VGG16	16,06%
DeepLab + VGG16	15,89%
DeepLabV3 + EfficientNetb3	15,13%
FCNet + EfficientNetb3	14,93%
FCNet + ResNet50	14,81%
FPNet + ResNet50	14,38%
SpOCRNet + VGG16	14,09%
ASPOCRNet + ResNet50	14,08%
PSPNet + VGG16	13,99%
DeepLab + EfficientNetb3	13,86%
FCNet + VGG16	13,78%
DANet + EfficientNetb3	13,44%
PSPNet + ResNet50	12,01%
ASPOCRNet + EfficientNetb3	10,54%
PSPNet + EfficientNetb3	1,64%

Figura 7.11: Rellis-3D 20. (*IoU* por clase) según *backbone* utilizado en los modelos en la inferencia del conjunto de *test*.

La tabla izquierda de la *Figura 7.11*, muestra los *Accuracy* obtenidos en la evaluación del entrenamiento en el conjunto de *test* de mayor a menor. La tabla derecha muestra los IoU promedio (mIoU) de cada variante del modelo de mayor a menor. Se aprecia que para los 8 mejores resultados obtenidos de cada tabla, prácticamente coinciden los mejores modelos (mayor Accuracy, mejor mIoU), pero no en el mismo orden. También se puede apreciar que no sólo es

importante la elección del modelo, sino también el *backbone* utilizado, ya que el rendimiento de un mismo modelo se ve afectado por la red base utilizada. Esto puede apreciarse por ejemplo, con el modelo FP-Net, que con red base EfficientNetb3 obtiene los mejores resultados en Accuracy y mIoU, pero si se utiliza con ResNet50 obtiene el peor valor de Accuracy y un mIoU promedio.

Como se vió anteriormente, no parece haber un *backbone* mejor que otro, ya que se ha visto que lo importante es la combinación *modelo-backbone*. Por lo tanto, se eligen los 5 mejores modelos, en base al *Accuracy* y mIoU mayor. Los modelos seleccionados son: ACF-Net, CF-Net, DeepLabV3+, FP-Net y SwinT-Unet. Cabe destacar que SwinT-Unet no utiliza red base, la extracción de características se realiza con un Transformador Visual (ViT). Estos modelos se utilizarán en los posteriores experimentos.

Respecto al conjunto de datos Rellis-3D con 20 etiquetas, se ha podido comprobar que existen clases escasamente representadas, y que únicamente 5 clases:*grass, sky, bush, tree*y *concrete*, se pueden segmentar con variable seguridad. Por ello, es necesario utilizar otra estrategia para utilizar el conjunto de datos para entrenar modelos. Alguna de ellas podría ser el aumentado de datos, procesar el conjunto de imágenes para agrupar las clases en categorías con mayor cantidad de píxeles o bien descartar clases. Esto no implica un problema para la tarea de percepción visual de un robot, ya que no es necesario que el robot identifique tantas clases para la navegación. Realmente necesita saber qué zonas son transitables, cuáles no y qué zonas debe evitar o esquivar.

7.2. Experimento 2: la importancia de los datos de entrenamiento - conjuntos de imágenes de 5 etiquetas

A partir de los resultados del Experimento 1 **(sección: 7.1)**, se establecen las bases para este segundo experimento. Se entrenan los 5 modelos seleccionados: SwinT-UNet sin *backbone* y ACF-Net, CF-Net, DeepLabV3+ y FP-Net con las redes base: VGG16, ResNet50 y EfficientNetb3. Estas 13 combinaciones se entrenan por separado con los conjuntos de datos: Rellis-3D, RUGD y GOOSE procesados como se detalla a continuación.

7.2.1. Pre-procesado de Conjuntos de Imágenes

Para evitar el problema de clases poco representadas en los conjuntos de imágenes y ya que, la tarea de percepción visual de un robot permite una segmentación más burda, se decide agrupar las clases originales de los tres conjuntos de datos, en 5 clases para que tengan un mayor número de píxeles y que representen zonas interesantes de navegabilidad para un robot terrestre. Como las clases entre los conjuntos de datos no son equivalentes, se procura agruparlas de forma

que representen 5 nuevas etiquetas: *background* (fondo no transitable), *obstable* (obstáculo), *withwater* (con agua), *unstable* (navegabilidad inestable) y *stable* (navegabilidad estable).

Al igual que se hizo en el pre-procesado de Rellis-3D del Experimento 1, en esta ocasión se generan nuevas máscaras con los valores de píxeles de 0 a 4 que corresponden a las nuevas etiquetas. La agrupación de las clases originales a las nuevas etiquetas se recogen en la *Figura 7.12* para cada conjunto de imágenes. Los tamaños de las imágenes de Rellis-3D y RUGD se mantienen (1920x1200 y 688x550 píxeles respectivamente), pero las de GOOSE de 2048x1000 se reducen a 655x320 píxeles. En GOOSE se incluye el conjunto de imágenes extendido GOOSE-Ex que contiene la misma anotación de etiquetas que GOOSE.

	ID	Nueva clase	Clases originales agrupadas
RELLIS-3D 20 a 5 etiquetas	**0**	**background**	void, sky
	1	**obstacle**	tree, bush, person, rubble, barrier, log, fence, vehicle, object, pole, building
	2	**withwater**	mud, puddle, water
	3	**unstable**	grass, dirt
	4	**stable**	concrete, asphalt
RUGD 24 a 5 etiquetas	**0**	**background**	void, sky
	1	**obstacle**	tree, bush, person, log, fence, vehicle, pole, building, bicycle, container/generic-object, rock-bed, sign, rock, bridge, picnic-table
	2	**withwater**	water
	3	**unstable**	dirt, sand, grass, gravel, mulch
	4	**stable**	concrete, asphalt
GOOSE 64 a 5 etiquetas	**0**	**background**	undefined, sky
	1	**obstacle**	resto de clases
	2	**withwater**	water
	3	**unstable**	snow, high_grass, bush, debris, crops, tree_root
	4	**stable**	bikeway, pedestrian_crossing, road_marking, sidewalk, asphalt, low_grass, cobble, leaves, moss, gravel, soil

Figura 7.12: Agrupación de etiquetas originales de los conjuntos de imágenes Rellis-3D, RUGD y GOOSE en 5 nuevas etiquetas.

Para el entrenamiento se ha dividido cada conjunto de datos en tres subconjuntos (*train, val, test*), con el mismo número de imágenes RGB y de máscaras que se especifican en la **Figura 7.13**. Siguiendo la misma estrategia de entrenamiento del Experimento 1 y el algoritmo descrito, se entrenan los 5 modelos (13 combinaciones) con Rellis-3D, RUGD y Goose de 5 etiquetas por separado.

Una vez obtenidos y guardados los pesos y métricas de los modelos, se realiza el análisis de resultados que se detalla en la siguiente sección.

	5 Etiquetas		
	train	val	test
Rellis-3D	3302	983	1672
RUGD	4779	1924	733
GOOSE	9467	2367	1369

Figura 7.13: Número de imágenes y máscaras para los tres subconjuntos *train, val* y *test* de Rellis-3D, RUGD y GOOSE de 5 etiquetas..

7.2.2. Entrenamiento

La *Figura 7.14* muestra los *Accuracy* de evaluación del conjunto de *test* obtenidos después del entrenamiento de 30 épocas más para los tres conjuntos de imágenes.

Los dos mejores resultados en Rellis-3D corresponden a FP-Net+EfficientNetb3 (94,2 % Accuracy) y FP-Net+ResNet50 (93,4 % Accuracy) en *test*. Con RUGD los mejores modelos son Swin-Transf-Unet (con ViT) y ACF-Net+ResNet50 con 93,5 % y 92,8 % Accuracy en *test* respectivamente. Por último, CF-Net+EfficientNetb3 y FP-Net+EfficientNetb3 con 88,1 % y 87,5 % Accuracy en test respectivamente son los mejores resultados obtenidos para el conjunto de datos GOOSE de 5 etiquetas.

De forma intuitiva cabría esperar que a modelos iguales, deberían dar resultados similares tanto si ha sido entrenado con GOOSE, con Rellis-3D o con RUGD, pero esto no ha sido así. Esto pone de manifiesto que la diferencia de resultados entre modelos está en el conjunto de imágenes y no en los modelos en sí. Esto puede ser debido a que los entornos de adquisición de los tres conjuntos de imágenes son diferentes, sobretodo en GOOSE y además las clases no se han anotado por los mismos métodos ni personas. Los conjuntos de imágenes RUGD y Rellis-3D son más similares debido a que se han obtenido con robot semejante, no siendo así con GOOSE, que se ha obtenido desde un vehículo, una excavadora y un robot. Esto queda reflejado en los resultados más similares entre Rellis-3D y RUGD. Por ejemplo, con el modelo SwinT-Unet el *Accuracy* es el más alto obtenido en RUGD pero es el peor con GOOSE en las mismas condiciones de entrenamiento.

7.2.3. Inferencia

La inferencia en el conjunto de prueba de cada conjunto de imágenes para las 13 combinaciones de modelos entrenados corresponden a SwinT-Unet sin red base y los modelos ACF-Net, CF-Net, DeepLabV3+ y FP-Net con los backbones EfficientNetb3, ResNet50 y VGG16.

	Accuracy Test		
Modelo	**Rellis-3D**	**RUGD**	**GOOSE**
ACF-Net + EfficientNetb3	91,246%	90,051%	85,098%
ACF-Net + ResNet50	92,868%	**92,846%**	86,093%
ACF-Net + VGG16	91,872%	91,125%	86,192%
CF-Net + EfficientNetb3	91,284%	90,394%	**88,119%**
CF-Net + ResNet50	92,390%	91,272%	87,700%
CF-Net + VGG16	92,990%	90,640%	87,161%
DeepLabV3+ + EfficientNetb3	92,993%	90,684%	82,204%
DeepLabV3+ + ResNet50	92,717%	90,597%	84,830%
DeepLabV3+ + VGG16	92,795%	90,533%	86,423%
FP-Net + EfficientNetb3	**94,236%**	88,991%	**87,949%**
FP-Net + ResNet50	**93,365%**	87,259%	86,952%
FP-Net + VGG16	92,743%	90,730%	86,010%
Swin-Transf-Unet	92,591%	**93,471%**	79,516%

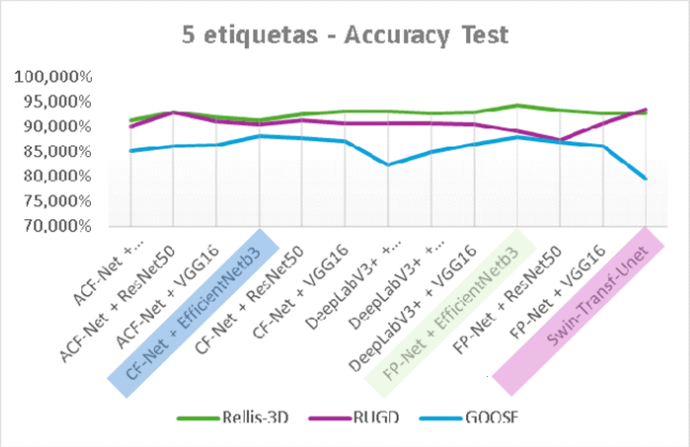


Figura 7.14: Izqda. Accuracy en la evaluación del conjunto de *test* después de entrenamiento con Rellis-3D, RUGD y GOOSE de 5 etiquetas. Dcha. Gráfica de Accuracy en *test*.

Rellis-3D de 5 etiquetas

Los modelos DeepLabV3+ y FP-Net ambos con EfficientNetb3 como red base obtienen los mejores resultados de inferencia del conjunto de *test* en Rellis-3D de 5 etiquetas, con 64,7 % y 65,7 % mIoU respectivamente **(Figura 7.15)**.

	RELLIS-3D 5 Etiquetas - mIoU por etiqueta					mIoU
Modelo	**background**	**obstacle**	**withwater**	**unstable**	**stable**	**Modelo**
ACFNet + EfficientNet	94,10%	77,07%	15,57%	81,19%	39,70%	61,53%
ACFNet + ResNet	94,22%	81,70%	15,51%	84,78%	41,11%	63,47%
ACFNet + VGG	93,01%	79,12%	13,64%	83,28%	37,21%	61,25%
CFNet + EfficientNet	93,57%	76,73%	10,58%	82,01%	40,35%	60,65%
CFNet + ResNet	93,80%	80,28%	11,88%	83,52%	35,42%	60,98%
CFNet + VGG	94,60%	82,20%	11,65%	85,17%	38,41%	62,41%
DeepLabV3+ + EfficientNet	**95,68%**	81,48%	16,62%	83,91%	45,77%	**64,69%**
DeepLabV3+ + ResNet	95,27%	80,62%	**16,63%**	83,40%	**46,29%**	64,44%
DeepLabV3+ + VGG	95,23%	80,58%	**16,67%**	83,69%	**46,02%**	64,44%
FPNet + EfficientNet	95,65%	**84,87%**	15,91%	**86,77%**	45,15%	**65,67%**
FPNet + ResNet	**95,68%**	**82,89%**	13,83%	**84,93%**	44,01%	64,27%
FPNet + VGG	95,45%	81,04%	15,43%	83,39%	44,16%	63,90%
SwinT-Unet	94,94%	81,68%	14,99%	84,21%	40,36%	63,23%
Píxeles Ground Truth	8314494	7845625	600063	9705441	928425	

Figura 7.15: Número de píxeles verdaderos por clase, mIoU por clase y mIoU de cada modelo en la inferencia del conjunto de *test* en Rellis-3D de 5 etiquetas.

Si se analizan los resultados por etiqueta, las clases mejor representadas de mayor a menor corresponden a *background* (>90 % mIoU), *unstable* (>80 % mIoU) y *obstacle* (>75 % mIoU). Las etiquetas *stable* (>35 % mIoU) y *withwater* (>10 % mIoU) obtienen resultados significativamente inferiores al resto y está relacionado con la cantidad de número de píxeles verdaderos para estas etiquetas, teniendo *stable* (928425 píxeles) y withwater (600063 píxeles) una cantidad de píxeles muy inferior al resto de clases.

RELLIS-3D 5 etiquetas	FP-Net + EfficientNetb3											
		background		obstacle		withwater		unstable		stable		
	ID	TP	IoU	TP	IoU	TP	IoU	TP	IoU	TP	IoU	Etiquetas
GT	31	22338	1	42832	1	18603	1	18627	1	0	1	background, obstacle, withwater, unstable
Predicción	31	22162	0,974	40782	0,934	16881	0,876	17371	0,792	0	1	background, obstacle, withwater, unstable
GT	42	38400	1	22307	1	4154	1	37539	1	0	1	background, obstacle, withwater, unstable
Predicción	42	38079	0,988	21686	0,925	212	0,051	36766	0,877	0	0	background, obstacle, withwater, unstable
GT	119	16113	1	33388	1	34212	1	18687	1	0	1	background, obstacle, withwater, unstable
Predicción	119	15798	0,965	30438	0,893	27585	0,786	17472	0,625	0	1	background, obstacle, withwater, unstable
GT	683	46455	1	21710	1	898	1	32504	1	833	1	background, obstacle, withwater, unstable, stable
Predicción	683	45354	0,967	20100	0,787	474	0,488	29963	0,901	673	0,435	background, obstacle, withwater, unstable, stable
GT	997	12244	1	48874	1	478	1	26867	1	13937	1	background, obstacle, withwater, unstable, stable
Predicción	997	11824	0,863	42795	0,796	2	0,001	24896	0,761	9337	0,654	background, obstacle, withwater, unstable, stable

Figura 7.16: Rellis-3D. Cantidad de píxeles (TP) y (mIoU) por etiqueta verdadera y predicha.

A continuación se seleccionan ejemplos de imágenes representativas **(Figura 7.17)** para comparar la imagen RGB, con la máscara verdadera y la predicha por el modelo con mejores resultados (FP-Net+EfficientNetb3).

El modelo para las 5 imágenes de ejemplo *(Figura 7.17)* predice que existen las mismas etiquetas que en la máscara verdadera *(Figura 7.16)*. Para la imagen con identificador 31 (ID 31), existe un camino de hierba (*unstable* en amarillo) con agua y lodo (*withwater* en azul) rodeado de arbustos y árboles (*obstacle* en rojo) y el cielo en el fondo (*background* en negro). No existe zona asfaltada (*stable* en verde). La predicción obtiene muy buenos resultados.

En la imagen 42 hay zonas con agua que no se han segmentado bien, si el robot no fuera apto para el agua, podría ser peligroso. En la imagen 119 hay una persona detectada bien como *obstacle* pero el agua está sub-segmentada. En la imagen 683 se observan las cinco etiquetas, y la predicción mejora la segmentación de la zona inestable (amarillo). Por último, en la imagen 997 aparece un camino de hierva en bajada (*unstable* en amarillo), rodeado de arbustos (*obstacle* en rojo). El camino termina en una zona plana (*stable* en verde). En la predicción hay algunos errores de segmentación, en el camino se segmenta una pequeña zona como *withwater* que no existe y otra como *background* siendo *obstacle*.

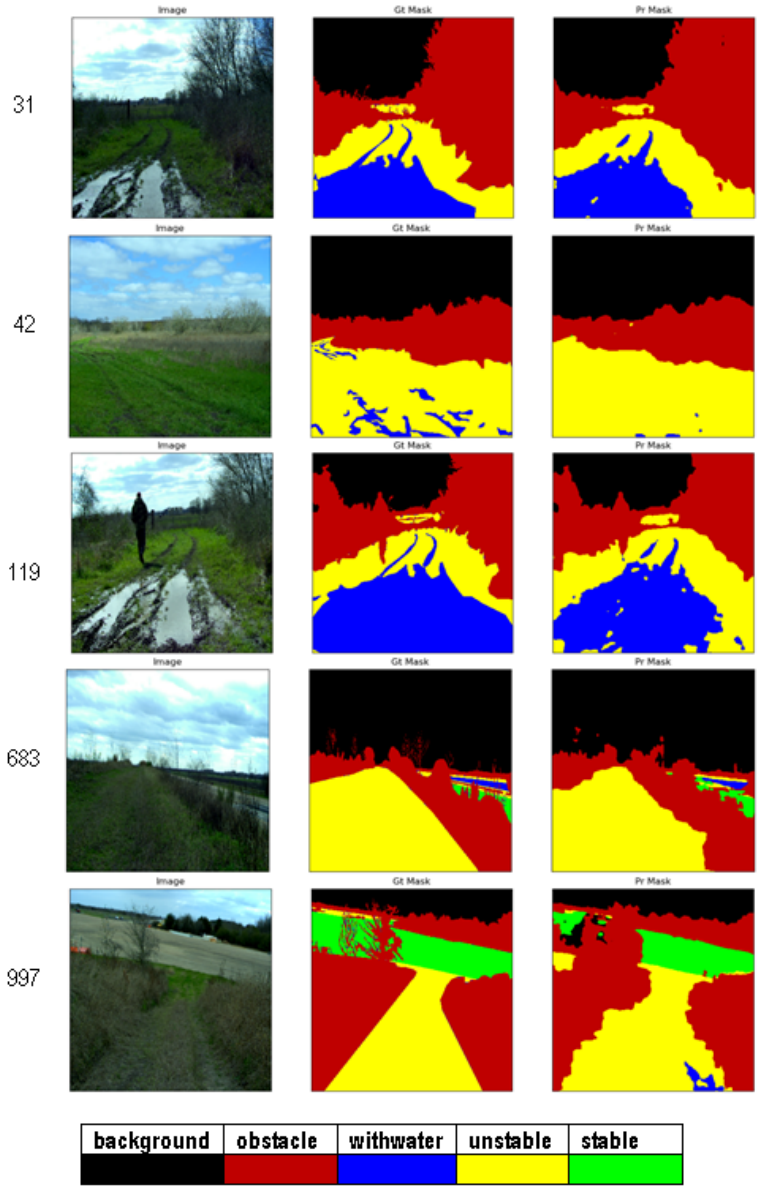


Figura 7.17: Rellis-3D. Izqda.: Imagen RGB. Centro: Máscara verdadera. Dcha.: Predicción con FP-Net+EfficientNetb3 en Rellis-3D de 5 etiquetas.

RUGD de 5 etiquetas

Los mejores resultados en la inferencia del conjunto de test en RUGD de 5 etiquetas corresponden también a DeepLabV3+ con EfficientNetb3 y FP-Net+VGG16 con 46,8 % y 47,0 % mIoU respectivamente *(Figura 7.18)*. Al igual que en Rellis-3D, dichos modelos no son los mejores a la hora de inferir por etiqueta, es decir, unos modelos predicen mejor unas etiquetas que otras (mIoU por clase).

Las etiquetas con mejores resultados son *obstacle* (>83 % mIoU), *unstable* (>81 % mIoU) y *background* (>30 % mIoU) y con deficientes resultados *withwater* (0 % mIoU) y *stable* (<10 % mIoU). Si observamos el número de píxeles verdaderos, se aprecia que *withwater* está poco

representada en el conjunto de datos (7631 píxeles) y *stable* con 469932 píxeles tampoco parece tener suficiente representación.

	RUGD 5 Etiquetas - mIoU por etiqueta					mIoU
Modelo	**background**	**obstacle**	**withwater**	**unstable**	**stable**	**Modelo**
ACFNet + EfficientNet	49,28%	88,34%	0,00%	86,17%	0,86%	44,93%
ACFNet + ResNet	46,91%	88,17%	0,00%	**87,82%**	**7,13%**	46,01%
ACFNet + VGG	39,30%	88,15%	0,00%	86,59%	4,44%	43,70%
CFNet + EfficientNet	35,01%	87,90%	0,00%	86,39%	1,57%	42,17%
CFNet + ResNet	34,39%	87,48%	0,00%	**86,60%**	4,18%	42,53%
CFNet + VGG	32,64%	87,65%	**0,05%**	86,58%	3,27%	42,04%
DeepLabV3+ + EfficientNet	**57,83%**	**89,63%**	0,00%	86,18%	0,37%	**46,80%**
DeepLabV3+ + ResNet	55,63%	89,38%	0,00%	86,13%	1,88%	46,60%
DeepLabV3+ + VGG	50,47%	88,88%	0,00%	86,17%	1,26%	45,36%
FPNet + EfficientNet	50,44%	86,08%	0,00%	83,39%	0,16%	44,01%
FPNet + ResNet	41,53%	83,14%	0,00%	81,00%	0,00%	41,13%
FPNet + VGG	56,20%	**89,39%**	0,00%	86,18%	3,53%	**47,06%**
SwinT-Unet	**65,60%**	85,81%	**0,03%**	75,57%	**4,92%**	46,38%
Píxeles Ground Truth	765926	5632860	7631	5133123	469932	

Figura 7.18: Número de píxeles verdaderos por clase, mIoU por clase y mIoU de cada modelo en la inferencia del conjunto de *test* en RUGD de 5 etiquetas.

Los ejemplos de imágenes representativas se muestran en la *Figura 7.19* para comparar la imagen RGB, con la máscara verdadera y la predicha por el modelo con mejores resultados (FP-Net+VGG16).

Como se observa en la *Figura 7.20*, el modelo no ha podido predecir en ninguna de las imágenes de ejemplo la etiqueta withwater (azul). Esto es debido a que en la máscara verdadera hay muy pocos píxeles (menos de 200 en cada imagen) para que el modelo pueda aprender a segmentar. De forma similar ocurre con la etiqueta *stable* (verde) que o está infra-representada o no está. En la imagen 520 no hay píxeles de dicha clase, en la imagen 139 y 490 hay 1 y 2 píxeles respectivamente, que son insuficientes para poder segmentar correctamente. Incluso las imágenes 59 y 324 con 24962 y 2868 píxeles para la etiqueta *stable*, no se han podido segmentar correctamente.

Se puede destacar que en la imagen 59 se ha clasificado la zona estable (verde) como inestable (amarillo), en cambio la pequeña zona estable de la imagen 324, sí se ha segmentado parcialmente. Los obstáculos (rojo), como los coches de la imagen 324, los árboles y la mesa de picnic de la imagen 139, se han segmentado correctamente.

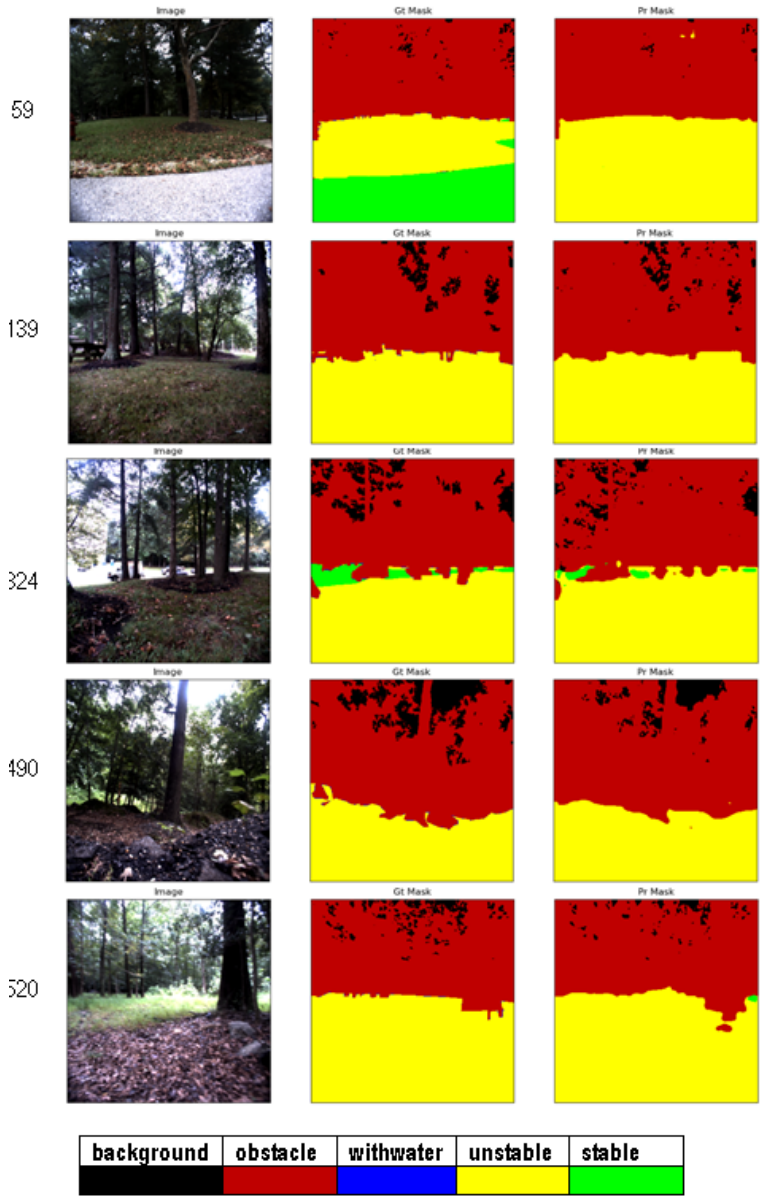


Figura 7.19: RUGD. Izqda.: Imagen RGB. Centro: Máscara verdadera. Dcha.: Predicción con FP-Net+VGG16 en RUGD de 5 etiquetas.

RUGD 5 etiquetas	FP-Net + VGG16											
		background		obstacle		withwater		unstable		stable		
	ID	TP	IoU	TP	IoU	TP	IoU	TP	IoU	TP	IoU	Labels
GT	59	1048	1	49200	1	107	1	27083	1	24962	1	background, obstacle, withwater, unstable, stable
Predicción	59	330	0,31	48929	0,972	0	0	26719	0,51	0	0	background, obstacle, unstable
GT	139	3801	1	53926	1	158	1	44514	1	1	1	background, obstacle, withwater, unstable, stable
Predicción	139	2571	0,599	53076	0,951	0	0	43963	0,978	0	0	background, obstacle, unstable
GT	324	5734	1	51317	1	133	1	42348	1	2868	1	background, obstacle, withwater, unstable, stable
Predicción	324	3182	0,496	49889	0,911	0	0	42222	0,954	1002	0,346	background, obstacle, unstable, stable
GT	490	9464	1	57985	1	187	1	34762	1	2	1	background, obstacle, withwater, unstable, stable
Predicción	490	4986	0,526	56212	0,889	0	0	34087	0,931	0	0	background, obstacle, unstable
GT	520	3735	1	46584	1	105	1	51976	1	0	1	background, obstacle, withwater, unstable
Predicción	520	1726	0,455	45500	0,924	0	0	51375	0,97	0	0	background, obstacle, unstable

Figura 7.20: Rellis-3D. Cantidad de píxeles (TP) y (mIoU) por etiqueta verdadera y predicha.

GOOSE de 5 etiquetas

Al igual que en Rellis-3D los mejores resultados en la inferencia del conjunto de *test* en GOOSE de 5 etiquetas corresponden también a DeepLabV3+ y FP-Net ambos con con EfficientNetb3 con 49,8 % y 49,7 % mIoU respectivamente **(Figura 7.21)**. Las etiquetas con mejores resultados son *stable* (>68 % mIoU), *obstacle* (>67 % mIoU) y *background* (>65 % mIoU) y con deficientes resultados *withwater* (<4 % mIoU) y *stable* (<25 % mIoU). Si observamos el número de píxeles verdaderos, *withwater* está poco representada (46806 píxeles) y *unstable* con 2295167 píxeles que parece tener suficientes píxeles de verdad en el conjunto de datos, no ha podido obtener buenos resultados.

	GOOSE 5 Etiquetas - mIoU por etiqueta					mIoU
Modelo	**background**	**obstacle**	**withwater**	**unstable**	**stable**	**Modelo**
ACFNet + EfficientNet	67,00%	74,43%	0,31%	18,35%	75,30%	47,08%
ACFNet + ResNet	65,81%	75,10%	0,02%	22,02%	77,08%	48,00%
ACFNet + VGG	66,39%	75,04%	0,03%	21,11%	76,96%	47,91%
CFNet + EfficientNet	66,63%	79,00%	0,54%	**23,54%**	**79,44%**	**49,83%**
CFNet + ResNet	66,72%	77,52%	**0,67%**	**23,80%**	**79,39%**	49,62%
CFNet + VGG	66,85%	77,52%	0,39%	21,93%	78,11%	48,96%
DeepLabV3+ + EfficientNet	67,39%	70,24%	0,00%	7,41%	73,22%	43,65%
DeepLabV3+ + ResNet	67,69%	74,63%	0,03%	13,27%	75,08%	46,14%
DeepLabV3+ + VGG	69,18%	76,75%	0,03%	17,94%	77,40%	48,26%
FPNet + EfficientNet	**69,90%**	**80,69%**	0,21%	19,40%	78,13%	**49,67%**
FPNet + ResNet	68,68%	**78,32%**	0,08%	20,96%	77,55%	49,12%
FPNet + VGG	**70,68%**	77,03%	0,20%	16,97%	76,23%	48,22%
SwinT-Unet	66,40%	67,20%	**3,45%**	1,89%	68,19%	41,43%
Píxeles Ground Truth	2975979	8141643	46806	2295167	8970101	

Figura 7.21: Número de píxeles verdaderos por clase, mIoU por clase y mIoU de cada modelo en la inferencia del conjunto de *test* en GOOSE de 5 etiquetas.

Si se analizan algunas imágenes de muestra (**Figura 7.22)**, en primer lugar se observa en las imágenes RGB, que el ángulo de visión de la cámara es más elevado que en Rellis-3D y RUGD debido a que está montada sobre un automóvil. Además, en las máscaras verdaderas se observa la parte delantera del vehículo que se anotó como *obstacle*. Al no tener en cuenta esta particularidad a la hora de montar la cámara, se produce una segmentación que podría dar lugar a errores en la navegación del robot. También se encuentran algunos errores de anotación a la hora de crear las máscaras verdaderas, por ejemplo, en la imagen 59 a la derecha del camino, se ha segmentado como verdad un terraplén o zona alta no navegable como zona estable (verde), cuando debería ser como mínimo no estable o un obstáculo. De forma similar ocurre a la derecha del camino en la imagen 490.

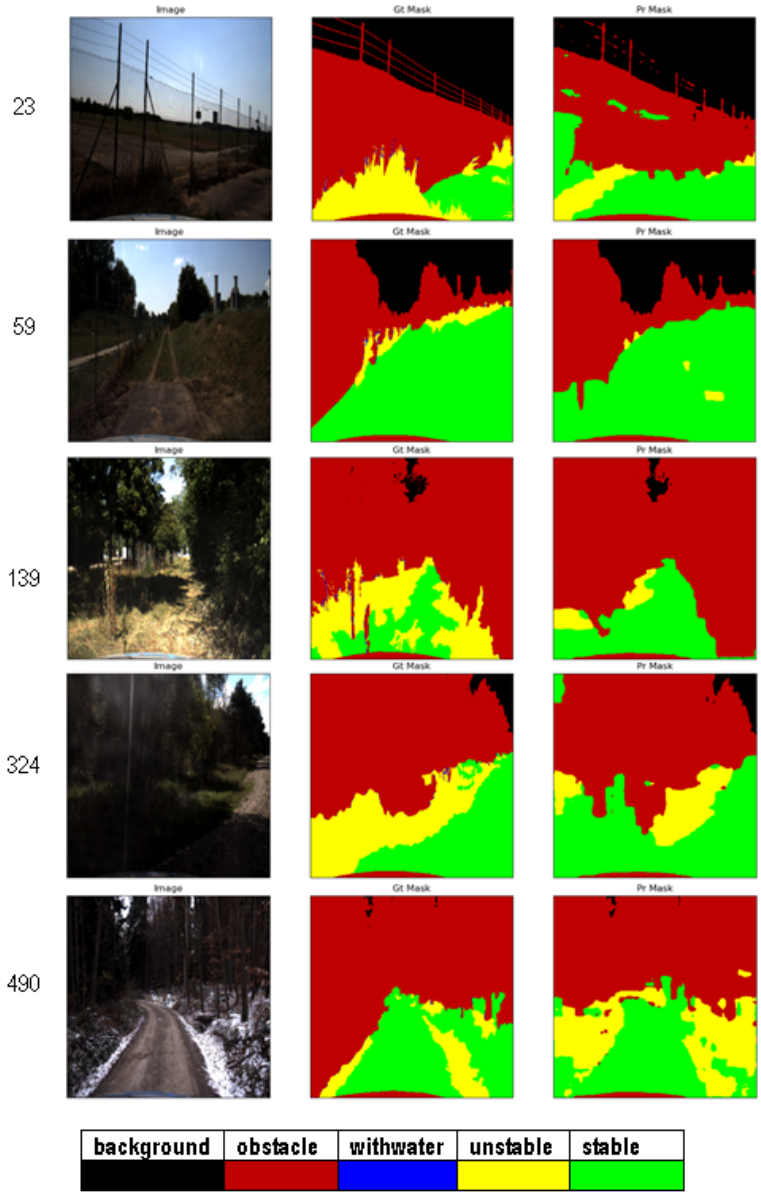


Figura 7.22: GOOSE. Izqda.: Imagen RGB. Centro: Máscara verdadera. Dcha.: Predicción con FP-Net+EfficientNetb3 en RUGD de 5 etiquetas.

En las imágenes seleccionadas se observa que al igual que en RUGD no se han podido segmentar las zonas con agua, debido a que están infra-representadas **(Figura 7.23)**. Aunque las zonas inestables se han podido segmentar, los resultados son mejorables. El resto de etiquetas obtiene buenos resultados.

7.2.4. Resumen Resultados

En este experimento se ha podido comprobar que entrenando con los mismos modelos y combinaciones, el rendimiento de entrenamiento no es equivalente entre los distintos conjuntos de imágenes. Si se comparan los mejores resultados de segmentación del primer experimento

(Rellis-3D de 20 etiquetas) con los obtenidos con Rellis-3D de 5 etiquetas, ambos con FP-Net EfficientNetb3, se observa que se pasa de 19,9 % a 65,7 % mIoU en *test*. Siendo una mejora considerable, se acepta la hipótesis inicial, de que, agrupando las clases menos representadas se obtienen mejores resultados.

GOOSE 5 etiquetas	GOOSE 5 - FP-Net+EfficientNetb3											
		background		obstacle		withwater		unstable		stable		
	ID	TP	IoU	TP	IoU	TP	IoU	TP	IoU	TP	IoU	Labels
GT	23	31952	1	46918	1	313	1	14522	1	8695	1	background, obstacle, withwater, unstable, stable
Pred	23	31442	0,942	37777	0,757	0	0	4520	0,297	8540	0,36	background, obstacle, unstable, stable
GT	59	19173	1	33673	1	198	1	3727	1	45629	1	background, obstacle, withwater, unstable, stable
Pred	59	19023	0,976	31177	0,864	0	0	210	0,049	44735	0,902	background, obstacle, unstable, stable
GT	139	1434	1	70147	1	388	1	19552	1	10879	1	background, obstacle, withwater, unstable, stable
Pred	139	1152	0,752	69251	0,883	0	0	3109	0,158	10690	0,52	background, obstacle, unstable, stable
GT	324	2637	1	57141	1	156	1	20589	1	21877	1	background, obstacle, withwater, unstable, stable
Pred	324	2596	0,944	51510	0,851	0	0	5634	0,229	19723	0,529	background, obstacle, unstable, stable
GT	490	240	1	66534	1	112	1	7855	1	27659	1	background, obstacle, withwater, unstable, stable
Pred	490	167	0,605	52624	0,779	0	0	4977	0,2	19725	0,571	background, obstacle, unstable, stable

Figura 7.23: GOOSE. Cantidad de píxeles (TP) y (mIoU) por etiqueta verdadera y predicha.

En el estudio realizado de los modelos seleccionados para comparar el comportamiento por separado de los conjuntos de imágenes de 5 etiquetas, en Rellis-3D se supera el 65 % mIoU en *test*, mientras que en RUGD y GOOSE no se alcanza el 50 % mIoU en *test*. Se toman como referencia los resultados aportados por estudios similares: RELLIS-3D y RUGD de 5 etiquetas [56] y GOOSE [47] aunque las etiquetas no sean equivalentes y GOOSE no incluya GOOSE-EX, como sí se incluye en este estudio. En Rellis-3D (65,67 % mIoU) se ha podido superar al modelo base de GANAV [56] (63,01 % mIoU) pero no a la variante lineal (68,91 %). En RUGD (47,06 %) no se han podido mejorar los resultados de [56] (>70 % mIoU). En GOOSE (49,83 % mIoU) tampoco se alcanzan los reportados en [47] (>60 % mIoU) pero en este caso aunque agrupan a 10 categorías de imágenes obtenidas sólo con el automóvil, en este estudio se incluyen las imágenes extendidas obtenidas con la excavadora y el robot.

	mIoU		
	RELLIS-3D	RUGD	GOOSE
GANAV	63,01%	74,32%	-
GANAV Linear	68,91%	77,64%	-
GOOSE PP-LiteSeg	-	-	67,21%
GOOSE DDRNet	-	-	70,23%
GOOSE Mask2Former	-	-	64,26%
EXPERIMENTO 2	**65,67%**	**47,06%**	**49,83%**

Figura 7.24: mIoU reportados en la bibliografía de referencia para RELLIS-3D y RUGD de 5 etiquetas y GOOSE con 10.

En estos conjuntos de imágenes se observan deficiencias en las anotaciones de las máscaras verdaderas y en GOOSE además del ángulo de visión diferente a Rellis-3D y RUGD, en las imágenes se visualiza la parte delantera del automóvil donde se encuentra la cámara, provocando que en primer plano se segmente un obstáculo.

La propia variabilidad en las condiciones de adquisición de las imágenes, así como los diferentes entornos de los conjuntos de imágenes, quedan reflejados en la distribución de las predicciones de las etiquetas *(Figura 7.25)*. En Rellis-3D el fondo *background* se consigue segmentar de forma óptima y en GOOSE de forma aceptable, pero en RUGD la segmentación es de peor calidad. Y viene evidenciado en que las imágenes adquiridas suelen estar en zonas más boscosas y con árboles altos que ocultan el cielo. Los obstáculos se consiguen segmentar de forma óptima en los tres conjuntos de imágenes, ya que al agrupar etiquetas, la cantidad de píxeles es representativa.

Las zonas con agua han sido las más difíciles de segmentar en los tres conjuntos de imágenes, debido a que hay pocas imágenes en las que aparezca, por lo tanto, al estar infra-representada, se refleja en los resultados. Una posible solución sería agrupar las zonas de agua y lodo como obstáculos. Las mayores diferencias en los tres conjuntos de datos se evidencia en las etiquetas *unstable* y *stable*. En Rellis-3D y RUGD, las zonas inestables se segmentan con resultados muy aceptables, sin embargo, en Goose no ocurre lo mismo. Esto podría ser debido (entre otras causas) al hecho detectado de que hay anotaciones incorrectas en las máscaras verdaderas. En cambio, las zonas estables en GOOSE se segmentan muy bien, y puede ser debido a que el conjunto de imágenes incluye escenas más estructuradas con zonas estables (carreteras, aceras). Lo contrario ocurre en RUGD con calidad de segmentación de las zonas estables muy baja (no hay imágenes estructuradas) y de calidad aceptable en Rellis-3D.

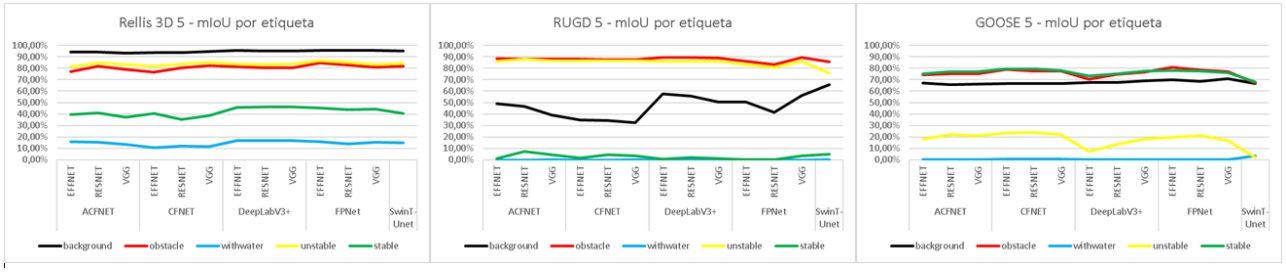


Figura 7.25: mIoU por predicción de etiqueta para los tres conjuntos de imágenes de 5 etiquetas: Rellis-3D, RUGD y GOOSE.

Si se tiene en cuenta el importante análisis que se realizó **(ver Sección 4.5)** respecto a las diferencias entre conjuntos de imágenes, los resultados obtenidos en este experimento han sido coherentes. Con Rellis-3D se han obtenido los mejores resultados debido a que todas las etiquetas en los conjunto de entrenamiento, validación y prueba estaban presentes y la adquisición de las

imágenes proceden del mismo robot. Por su parte, con RUGD se esperaba un menor rendimiento que con Rellis-3D, debido a qué algunas etiquetas inferidas no estaban presentes en el entrenamiento. El menor rendimiento con GOOSE que incluye GOOSE-Ex, está totalmente justificado aunque todas las etiquetas estén presentes en los tres subconjuntos, sin embargo, las imágenes adquiridas proceden de tres plataformas diferentes: el automóvil, la excavadora y el robot. Esta característica permite evaluar la generalización del modelo frente a distintas plataformas, pero la dificultad de segmentación es más desafiante.

7.3. Experimento 3: imágenes combinadas (OFFROAD)

Este experimento se planteó antes del análisis del segundo experimento. Parte de la hipótesis de que uniendo los tres conjuntos de imágenes: Rellis-3D, RUGD y Goose de 5 etiquetas, se podría resolver el problema de las etiquetas poco representadas.

7.3.1. Conjunto de imágenes

Respetando los subconjuntos de imágenes *train, val* y *test* de Rellis-3D, RUGD y GOOSE con GOOSE-Ex de 5 etiquetas del Experimento 2, se unen en un único conjunto de imágenes denominado OFFROAD, con 17548 imágenes para *train*, 5274 para *val* y 3774 para *test*. Los tamaños de imagen en Rellis-3D y RUGD se reducen (512x320 y 400x320 píxeles respectivamente) y se mantienen (655x320 píxeles en GOOSE y GOOSE-Ex).

7.3.2. Entrenamiento

Se utilizan los mismos modelos y combinaciones utilizados en el experimento 2, pero con el conjunto de datos OFFROAD (5 etiquetas). Las condiciones de entrenamiento se mantienen también. Los dos mejores resultados obtenidos son CF-Net+EfficientNetb3 y FP-Net+VGG16 con 84,1 % y 82,0 % Accuracy en *test* respectivamente (Figura 7.26. Lo que indica que el rendimiento es inferior respecto a los entrenamientos de los conjuntos de datos por separado, en los que se obtuvieron un rendimiento superior al 90 % en Rellis-3D y RUGD, y superior al 85 % de Accuracy en *test*. El inferior rendimiento de GOOSE y las propias diferencias en la adquisición y anotación de las imágenes entre los diferentes conjuntos de imágenes que se comentó en el experimento 2, es lo que ha provocado que los resultados finales con COMBINADO se hayan promediado. A continuación, se analizan los resultados en la inferencia del conjunto de test de los modelos y las etiquetas por separado.

Experimento 3	Accuracy Test
Modelo	OFFROAD
ACF-Net + EfficientNetb3	68,550%
ACF-Net + ResNet50	74,374%
ACF-Net + VGG16	68,906%
CF-Net + EfficientNetb3	**84,053%**
CF-Net + ResNet50	79,896%
CF-Net + VGG16	74,777%
DeepLabV3+ + EfficientNetb3	77,693%
DeepLabV3+ + ResNet50	72,923%
DeepLabV3+ + VGG16	73,827%
FP-Net + EfficientNetb3	79,493%
FP-Net + ResNet50	78,418%
FP-Net + VGG16	82,010%

Figura 7.26: OFFROAD. Accuracy en la evaluación del conjunto de *test* después de entrenamiento con OFFROAD de 5 etiquetas.

7.3.3. Inferencia

La Figura 7.27 recoge los resultados de inferencia en el conjunto de *test* de OFFROAD, para cada clase y por modelo. La etiqueta *withwater* sigue teniendo deficientes resultados, ya que como se comentó en el Experimento 2, está subrepresentada en los conjuntos de imágenes Rellis-3D, RUGD y GOOSE, por lo tanto, en OFFROAD también.

Si en Rellis-3D, RUGD y GOOSE se obtenían aproximadamente en los mejores modelos para la clase *background* 95 %, 65 % y 70 % mIoU en test respectivamente, en OFFROAD se obtiene 80 %. En la clase *obstacle* 84 %, 89 % y 80 % mIoU en *test*, mientras que en OFFROAD 83 % mIoU en *test*. Si en Rellis-3D y RUGD por separado se alcanzaba 85 % y en GOOSE 23 % mIoU aprox. en *test*, OFFROAD obtiene 65 % mIoU en test para la clase *unstable*.

Por último, para la etiqueta *stable*, si Rellis-3D no alcanzaba el 50 %, RUGD obtenía resultados deficientes, y GOOSE alcanzaba 79 % mIoU en *test* (contiene imágenes estructuradas), con OFFROAD se obtiene 40 % mIoU.

De forma general, OFFROAD obtiene resultados de segmentación promediados respecto a los tres conjuntos de imágenes analizadas por separado. Se procede a evaluar algunas imágenes de ejemplo, para valorar la calidad de la segmentación en OFFROAD.

Si se comparan las imágenes de ejemplo en la predicción de GOOSE obtenidas con FP-Net+EfficientNetb3, con las mismas obtenidas en la inferencia de OFFROAD con CF-Net+EfficientNetb3, se observa una calidad de segmentación inferior en OFFROAD. Las predicciones con GOOSE se asemejan mejor a las máscaras verdaderas que con OFFROAD.

	OFFROAD 5 Etiquetas - mIoU por etiqueta					mIoU
Modelo	background	obstacle	withwater	unstable	stable	Modelo
ACFNet + EfficientNet	71,79%	66,29%	**5,49%**	42,32%	13,28%	39,83%
ACFNet + ResNet	73,28%	67,22%	0,31%	53,71%	11,37%	41,18%
ACFNet + VGG	69,87%	64,48%	**6,36%**	43,75%	17,76%	40,44%
CFNet + EfficientNet	**81,19%**	**83,34%**	2,09%	**62,32%**	**41,62%**	**54,11%**
CFNet + ResNet	**79,22%**	**81,15%**	0,07%	**64,80%**	**27,88%**	**50,63%**
CFNet + VGG	74,13%	67,77%	0,02%	53,70%	9,51%	41,03%
DeepLabV3+ + EfficientNet	77,01%	70,36%	0,05%	56,07%	20,80%	44,86%
DeepLabV3+ + ResNet	75,26%	64,03%	0,01%	51,95%	11,56%	40,56%
DeepLabV3+ + VGG	76,24%	70,56%	0,10%	53,60%	6,65%	41,43%
FPNet + EfficientNet	78,74%	78,93%	1,02%	58,78%	23,78%	48,25%
FPNet + ResNet	76,92%	74,75%	0,29%	56,73%	14,70%	44,68%
FPNet + VGG	76,24%	70,56%	0,10%	53,60%	6,65%	41,43%
SwinT-Unet	77,24%	72,35%	0,93%	54,10%	28,33%	46,59%
Píxeles Ground Truth	75355839	135147726	4090133	107063361	64800541	

Figura 7.27: OFFROAD. mIoU y número de píxeles verdaderos por etiqueta y mIoU modelo obtenidos en la inferencia del conjunto de test.

7.3.4. Resumen Resultados

Después de evaluar la información cuantitativa y cualitativa obtenida en el conjunto de imágenes OFFROAD, se puede decir que no se cumple la hipótesis inicial de que uniendo los tres conjuntos de imágenes la segmentación mejore, y solucione el problema de las etiquetas poco representadas. No se tienen métricas de referencia de estudios similares para realizar comparaciones pero la calidad de segmentación es inferior respecto a los resultados obtenidos en los conjuntos de imágenes por separado. Como se comentó en el Experimento 2, al incluir en OFFROAD imágenes obtenidas en 5 plataformas diferentes, el desafío es mayor. La unión de diferentes conjuntos de imágenes puede ser interesante si nos interesa un modelo que sea capaz de generalizar con suficiente calidad en circunstancias en las que se necesite el uso de diferentes plataformas en las que va montada la cámara. Si se requiere de una sola plataforma, es mejor utilizar el conjunto de imágenes que represente mejor las condiciones que tendrá el robot en su entorno real.

7.4. Experimento 4: tiempos de inferencia

Respecto a los tiempos de inferencia de los modelos y combinaciones, se toman como ejemplo de estudio, los tiempos obtenidos en el conjunto de imágenes Rellis-3D de 5 etiquetas y los modelos del Experimento 2. La Figura 7.29 muestra los tiempos de evaluación por imagen y por conjunto de *test* de menor a mayor a la hora de evaluar Rellis-3D de 5 etiquetas.

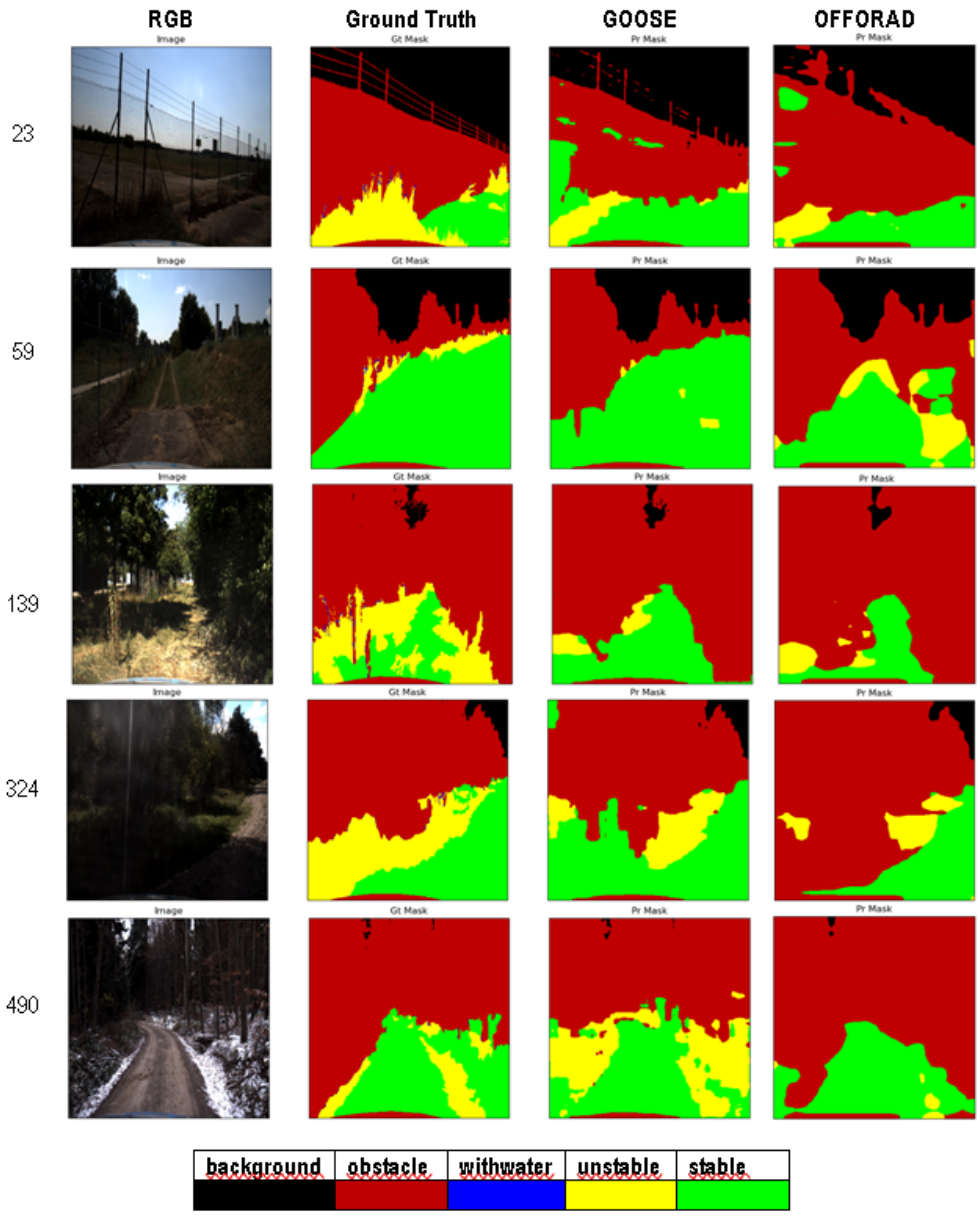


Figura 7.28: OFFROAD. De izquierda a derecha: Imagen RGB. Máscara verdadera, Predicción GOOSE FP-Net+EfficientNetb3, Predicción OFFROAD.

Rellis-3D 5 etiquetas	Evaluación test (1672 imágenes)		
Modelo	ms/imagen	Total en test	Tamaño
ACF-Net + EfficientNetb3	54	91	320x320
CF-Net + EfficientNetb3	78	130	320x320
ACF-Net + ResNet50	168	281	320x320
DeepLabV3+ + ResNet50	183	307	320x320
DeepLabV3+ + VGG16	187	312	320x320
Swin-Transf-Unet	190	318	128x128
CF-Net + ResNet50	192	321	320x320
ACF-Net + VGG16	256	428	320x320
CF-Net + VGG16	258	431	320x320
DeepLabV3+ + EfficientNetb3	453	757	320x320
FP-Net + EfficientNetb3	461	771	320x320
FP-Net + ResNet50	536	900	320x320
FP-Net + VGG16	856	1431	320x320

Figura 7.29: Rellis-3D 5 etiquetas. Tiempos de inferencia por imagen y conjunto de *test*.

7.5. Discusión

En el primer experimento se ha utilizado un conjunto de datos no estructurado desafiante para la segmentación semántica (Rellis-3D de 20 etiquetas), en el que se han podido evaluar diferentes modelos y combinaciones para seleccionar aquellos mejores. Cabe destacar que en pruebas iniciales se ha evaluado el balanceo de etiquetas, la ponderación de pesos, y el aumentado de datos (giro respecto al eje y) con resultados no satisfactorios y descartados en los experimentos.

En el segundo experimento se evalúan los mejores modelos y combinaciones seleccionados en el experimento 1 para entrenarlos con los tres conjuntos de imágenes por separado. Se pudo observar que las diferencias obtenidas vienen dadas por la metodología de adquisición y anotación de las imágenes de cada conjunto de imágenes, así como la calidad de anotaciones de las máscaras verdaderas. La hipótesis de que agrupando etiquetas se pueden obtener mejores resultados ha sido corroborada. Agrupando las 20 etiquetas de Rellis-3D a 5 etiquetas se han obtenido resultados significativamente mejores.

Tras el experimento 3, se puede sugerir que la combinación de los tres conjuntos de imágenes de 5 etiquetas, en uno sólo (OFFROAD) no mejora la calidad de la segmentación, por lo tanto se rechaza la hipótesis inicial.

A partir de los resultados obtenidos, se concluye que, a la hora de modelar el sistema de percepción visual de un robot terrestre en entornos no estructurados, es de vital importancia utilizar un conjunto de imágenes de entrenamiento que sea lo más representativo posible a las condiciones en las que se situará el robot real. Por ejemplo, la altura de la cámara respecto al suelo: se vieron diferencias significativas entre Rellis-3D y RUGD respecto a GOOSE. Este

último tiene incorporada la cámara en un automóvil tipo SUV, mientras que los otros dos la tienen en un robot pequeño y con la cámara a una distancia al suelo menor. Esto hace que las imágenes adquiridas difieran en su perspectiva (pendientes, ocultaciones, sombras, etc.). Otra característica diferente en GOOSE, es que en primer plano aparece la parte delantera del vehículo que al segmentarse como obstáculo, puede afectar al movimiento del robot. También se observaron diferencias a la hora de elegir las escenas donde se capturan las imágenes, GOOSE incluye algunas imágenes estructuradas, pero por otro lado, tiene en cuenta las diferentes estaciones del año y condiciones meteorológicas: lluvia, nieve, soleado, etc. RUGD y Rellis-3D son más desafiantes (ocultaciones, sombras, pendientes) y sin imágenes estructuradas, sin embargo, con estos dos conjuntos de imágenes se han obtenido mejores resultados, sobretodo en Rellis-3D con 5 etiquetas.

Como se comentó anteriormente parecen existir algunos errores a la hora de realizar las anotaciones de etiquetas de las máscaras verdaderas, y además, al realizarse manualmente por personal y técnicas diferentes, puede hacer que un píxel de una determinada etiqueta, se etiquete diferente entre conjuntos de imágenes.

En ninguno de los experimentos se ha podido segmentar con suficiente calidad y fiabilidad las zonas húmedas o con agua, por lo que sería interesante equipar al robot real con protección para agua, al menos para pequeños charcos y lodo.

Como se vio en la *sección 7.4* respecto a los tiempos de inferencia, se recomienda seleccionar un modelo que tenga un compromiso entre velocidad de inferencia y calidad de segmentación de acuerdo a las características técnicas y de entorno al que vayan destinado.

Como conclusión, se puede afirmar que es mejor seleccionar un conjunto de imágenes lo más semejante posible a los entornos que se encontrará el robot real y que el método de adquisición de imágenes (plataforma de la cámara) sea también lo más semejante posible. Si esto es desconocido o se requiere generalirzar a varias plataformas, un conjunto de imágenes combinado como OFFROAD puede ser interesante. También podría evaluarse la combinación de Rellis-3D y RUGD sin incluir GOOSE, ya que este último tiene mayores diferencias.

Respecto a los modelos y combinaciones, se puede concluir que las redes de segmentación ACF-Net, CF-Net, DeepLabV3+ y FP-Net con las diferentes redes base, tienen una calidad aceptable para una segmentación gruesa y suficiente para la percepción de un robot. La red SwinT-Unet que utiliza ViT en lugar de red base basada en CNN, también es una opción muy interesante, ya que ha sido el mejor modelo para RUGD.

Capítulo 8

Conclusiones y líneas futuras

En esta sección se exponen las conclusiones extraídas al desarrollar este trabajo así como las posibles líneas futuras que se pueden seguir para continuar el proyecto.

8.1. Conclusiones

A lo largo de este TFM se han extraído las siguientes conclusiones:

- **Objetivo 1. Modelos**: se ha realizado un estudio del Estado del Arte de la segmentación semántica visual a nivel de píxel con técnicas de aprendizaje profundo **(Capítulo 2)**, en especial aquellos trabajos centrados en la percepción visual para conducción autónoma de robots en entornos no estructurados **(Capítulo 1)**. Sin ánimo de ser exhaustivos, se han analizado los modelos seleccionados para comprender su funcionamiento, destacando las propiedades y características relevantes de cada modelo y las estrategias que se emplean a la hora de poder resolver problemas desafiantes en la segmentación semántica visual **(Capítulo 5)**.

- **Objetivo 2. Conjuntos de imágenes**: se han podido seleccionar tres conjuntos de imágenes adquiridos en entornos no estructurados **(Capítulo 4)** para el entrenamiento de los modelos seleccionados. Se analizaron las propiedades generales por separado y las diferencias entre ellos que son de vital importancia a la hora de obtener buenos resultados.

- **Objetivo 3. Experimentos:** se han diseñado cuatro experimentos a modo de estrategia para poder evaluar su uso para percepción visual. El primer experimento se centra en los modelos, el segundo en los tres conjuntos de imágenes seleccionados por separado, el tercero en la creación de un nuevo conjunto de imágenes a partir de la combinación de los tres anteriores y el tercero centrado en los tiempos de inferencia. Esto ha permitido obte-

ner un conocimiento profundo en el funcionamiento y comportamiento de los modelos de segmentación semántica visual y sus combinaciones a la hora de entrenar con los diferentes conjuntos de imágenes y cómo las diferencias de éstos afectan a los resultados **(Capítulo 7)**. Se reportaron resultados cualitativos y cuantitativos para su posterior análisis. Las métricas **(Capítulo 6)** evalúan los rendimientos de entrenamiento e inferencia, la calidad de la segmentación y los tiempos de inferencia **(Capítulo 7)**. Por otro lado, se aportan imágenes de ejemplo, para valorar cualitativamente la calidad de las segmentaciones.

- **Objetivo 4. Análisis**: en la sección *7.5* **(Capítulo 7)** se analizó la viabilidad de las estrategias propuestas en base a los resultados cuantitativos y cualitativos obtenidos.

Finalmente, este trabajo sugiere que con pocos recursos, es posible dotar de percepción visual para conducción autónoma a un robot en entornos no estructurados con una calidad aceptable y teniendo en cuenta las consideraciones comentadas.

8.2. Líneas Futuras

Para concluir, se muestran las líneas futuras interesantes que se sugieren investigar:

- Repetir las condiciones del Experimento 3 para GOOSE excluyendo GOOSE-EX (varias plataformas) y agrupar las etiquetas en base a categorías **(Figura 4.12)**.
- Estudiar la agrupación de etiquetas de forma aún más eficiente de los conjuntos de imágenes por separado y combinadas. Por ejemplo, eliminar la etiqueta *withwater* o incluir imágenes que permitan que esta etiqueta no esté sub-representada. Con los nuevos conjuntos de imágenes valorar el uso de balanceo de clases y aumentado de datos. Se debe tener en cuenta que la modificación de imágenes no debe modificar el valor de los píxeles, ni el sentido semántico de las etiquetas. Por ejemplo, utilizar imágenes *espejo* o generar nuevas con Redes Generativas Adversarias (GAN).
- Hacer una validación cruzada con conjuntos de imágenes diferentes e incluyendo WildScenes *Tabla 2.1*. Por ejemplo, entrenar con la combinación de Rellis-3D y RUGD o GOOSE y evaluando en WildScenes. De esta forma se demostraría el grado de generalidad alcanzable por las redes de segmentación semántica de imágenes.
- Añadir información multimodal. Rellis-3D y GOOSE incluyen datos de un sensor LIDAR con nubes de puntos que combinada con la segmentación semántica propuesta, podría ofrecer mejores resultados.

Bibliografía

[1] Sikai Chen et al. "A taxonomy for autonomous vehicles considering ambient road infrastructure". En: *Sustainability* 15.14 (2023), pág. 11258.

[2] Tesla. *https://www.tesla.com/es_ES?redirect = no*. 6 de Mayo de 2025.

[3] Waymo. *https://waymo.com/intl/es/*. 6 de Mayo de 2025.

[4] Ross Girshick. "Fast r-cnn". En: *Proceedings of the IEEE international conference on computer vision*. 2015, págs. 1440-1448. URL: https://openaccess.thecvf.com/content_iccv_2015/papers/Girshick_Fast_R-CNN_ICCV_2015_paper.pdf.

[5] Joseph Redmon et al. "You only look once: Unified, real-time object detection". En: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, págs. 779-788. URL: https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Redmon_You_Only_Look_CVPR_2016_paper.pdf.

[6] Ross Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". En: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, págs. 580-587. URL: https://openaccess.thecvf.com/content_cvpr_2014/papers/Girshick_Rich_Feature_Hierarchies_2014_CVPR_paper.pdf.

[7] Wei Liu et al. "Ssd: Single shot multibox detector". En: *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*. Springer. 2016, págs. 21-37. URL: https://link.springer.com/chapter/10.1007/978-3-319-46448-0_2.

[8] Evan Shelhamer, Jonathan Long y Trevor Darrell. "Fully convolutional networks for semantic segmentation". En: *IEEE transactions on pattern analysis and machine intelligence* 39.4 (2016), págs. 640-651. URL: https://arxiv.org/pdf/1411.4038.pdf.

[9] Lucas Prado Osco et al. "Semantic segmentation of citrus-orchard using deep neural networks and multispectral UAV-based imagery". En: *Precision Agriculture* 22.4 (2021),

págs. 1171-1188. URL: https://link.springer.com/article/10.1007/s11119-020-09777-5.

[10] Rongxin Guo et al. "Jmlnet: Joint multi-label learning network for weakly supervised semantic segmentation in aerial images". En: *Remote Sensing* 12.19 (2020), pág. 3169. URL: https://www.mdpi.com/2072-4292/12/19/3169.

[11] Kaiming He et al. "Mask r-cnn". En: *Proceedings of the IEEE international conference on computer vision*. 2017, págs. 2961-2969. URL: https://arxiv.org/abs/1703.06870.

[12] José Augusto Correa Martins et al. "Semantic segmentation of tree-canopy in urban environment with pixel-wise deep learning". En: *Remote Sensing* 13.16 (2021), pág. 3054. URL: https://www.mdpi.com/2072-4292/13/16/3054.

[13] Sasha Targ, Diogo Almeida y Kevin Lyman. "Resnet in resnet: Generalizing residual architectures". En: *arXiv preprint arXiv:1603.08029* (2016). URL: https://arxiv.org/abs/1603.08029.

[14] Karen Simonyan y Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". En: *arXiv preprint arXiv:1409.1556* (2014). URL: https://arxiv.org/abs/1409.1556.

[15] Mingxing Tan y Quoc Le. "Efficientnet: Rethinking model scaling for convolutional neural networks". En: *International conference on machine learning*. PMLR. 2019, págs. 6105-6114. URL: https://arxiv.org/abs/1905.11946v5.

[16] Alex Krizhevsky, Ilya Sutskever y Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". En: *Advances in neural information processing systems* 25 (2012).

[17] Christian Szegedy et al. "Going deeper with convolutions". En: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, págs. 1-9. URL: https://arxiv.org/abs/1409.4842.

[18] Mark Everingham et al. "The pascal visual object classes (voc) challenge". En: *International journal of computer vision* 88 (2010), págs. 303-338. URL: https://link.springer.com/article/10.1007/S11263-009-0275-4.

[19] Laritza Pérez-Enríquez, Raquel Díaz-Hernández y Leopoldo Altamirano Robles. "Implementación de CNN basada en una arquitectura VGG16 para detección y clasificación de árboles mediante la segmentación semántica en imágenes áereas." En: *Res. Comput. Sci.* 151.7 (2022), págs. 157-170. URL: https://www.rcs.cic.ipn.mx/2022_151_

7/Implementacion%20de%20CNN%20basada%20en%20una%20arquitectura%20VGG16%20para%20deteccion%20y%20clasificacion.pdf.

[20] Jia Deng et al. "Imagenet: A large-scale hierarchical image database". En: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, págs. 248-255. URL: https://image-net.org/index.php.

[21] Olaf Ronneberger, Philipp Fischer y Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". En: *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*. Springer. 2015, págs. 234-241. URL: https://arxiv.org/abs/1505.04597.

[22] Martin Maška et al. "A benchmark for comparison of cell tracking algorithms". En: *Bioinformatics* 30.11 (2014), págs. 1609-1617. URL: OP-CBIO140081%201609..1617%20(silverchair.com).

[23] Assaf Arbelle y Tammy Riklin Raviv. "Microscopy cell segmentation via convolutional LSTM networks". En: *2019 IEEE 16th International Symposium on Biomedical Imaging (ISBI 2019)*. IEEE. 2019, págs. 1008-1012. URL: https://arxiv.org/pdf/1805.11247v2.

[24] Hengshuang Zhao et al. "Pyramid scene parsing network". En: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, págs. 2881-2890. URL: https://arxiv.org/abs/1612.01105.

[25] Marius Cordts et al. "The cityscapes dataset for semantic urban scene understanding". En: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, págs. 3213-3223. URL: http://arxiv.org/abs/1604.01685.

[26] Tsung-Yi Lin et al. "Microsoft coco: Common objects in context". En: *Computer vision–ECCV 2014: 13th European conference, zurich, Switzerland, September 6-12, 2014, proceedings, part v 13*. Springer. 2014, págs. 740-755. URL: https://link.springer.com/chapter/10.1007/978-3-319-10602-1_48.

[27] Liang-Chieh Chen et al. "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs". En: *IEEE transactions on pattern analysis and machine intelligence* 40.4 (2017), págs. 834-848. URL: https://arxiv.org/pdf/1606.00915.pdf.

[28] Liang-Chieh Chen et al. "Rethinking atrous convolution for semantic image segmentation". En: *arXiv preprint arXiv:1706.05587* (2017). URL: https://arxiv.org/pdf/1706.05587.pdf.

[29] Liang-Chieh Chen et al. "Encoder-decoder with atrous separable convolution for semantic image segmentation". En: *Proceedings of the European conference on computer vision (ECCV)*. 2018, págs. 801-818. URL: https://arxiv.org/pdf/1802.02611.pdf.

[30] Jun Fu et al. "Dual attention network for scene segmentation". En: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, págs. 3146-3154. URL: https://arxiv.org/pdf/1809.02983.pdf.

[31] Fan Zhang et al. "Acfnet: Attentional class feature network for semantic segmentation". En: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, págs. 6798-6807. URL: https://arxiv.org/pdf/1909.09408.pdf.

[32] Hang Zhang et al. "Co-occurrent features in semantic segmentation". En: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, págs. 548-557. URL: https://openaccess.thecvf.com/content_CVPR_2019/papers/Zhang_Co-Occurrent_Features_in_Semantic_Segmentation_CVPR_2019_paper.pdf.

[33] Bowen Cheng, Alex Schwing y Alexander Kirillov. "Per-pixel classification is not all you need for semantic segmentation". En: *Advances in neural information processing systems* 34 (2021), págs. 17864-17875. URL: https://proceedings.neurips.cc/paper/2021/hash/950a4152c2b4aa3ad78bdd6b366cc179-Abstract.html.

[34] Yuhui Yuan, Xilin Chen y Jingdong Wang. "Object-contextual representations for semantic segmentation". En: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VI 16*. Springer. 2020, págs. 173-190. URL: https://arxiv.org/pdf/1909.11065.pdf.

[35] Ke Sun et al. "High-resolution representations for labeling pixels and regions". En: *arXiv preprint arXiv:1904.04514* (2019). URL: https://arxiv.org/pdf/1904.04514.pdf.

[36] Lei Pi y Jin Wu. "FPNet: Fusion Attention Instance Segmentation Network Based On Pose Estimation". En: *2021 33rd Chinese Control and Decision Conference (CCDC)*. IEEE. 2021, págs. 2426-2431. URL: https://ieeexplore.ieee.org/document/9602451.

[37] Md Jahidul Islam et al. "Semantic segmentation of underwater imagery: Dataset and benchmark". En: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, págs. 1769-1776. URL: https://arxiv.org/pdf/2004.01241.

[38] Jieneng Chen et al. "Transunet: Transformers make strong encoders for medical image segmentation". En: *arXiv preprint arXiv:2102.04306* (2021). URL: https://arxiv.org/abs/2102.04306.

[39] Hu Cao et al. “Swin-unet: Unet-like pure transformer for medical image segmentation”. En: *European conference on computer vision*. Springer. 2022, págs. 205-218. URL: https://arxiv.org/abs/2105.05537.

[40] Z Xu. “Multi-atlas labeling beyond the cranial vault-workshop and challenge”. En: *Synapse website* (2016). URL: https://www.synapse.org/Synapse:syn3193805/wiki/89480.

[41] Foivos I Diakogiannis et al. “ResUNet-a: A deep learning framework for semantic segmentation of remotely sensed data”. En: *ISPRS Journal of Photogrammetry and Remote Sensing* 162 (2020), págs. 94-114. URL: https://www.sciencedirect.com/science/article/abs/pii/S0924271620300149.

[42] *ISPRS WG III/4. ISPRS 2D Semantic Labeling Contest. Available online*. http://www2.isprs.org/commissions/comm3/wg4/semantic-labeling.html. [Acceso 07-03-2025].

[43] Kai A Metzger, Peter Mortimer y Hans-Joachim Wuensche. “A fine-grained dataset and its efficient semantic segmentation for unstructured driving scenarios”. En: *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE. 2021, págs. 7892-7899. URL: https://ieeexplore.ieee.org/abstract/document/9411987.

[44] Rui Nunes, João Filipe Ferreira y Paulo Peixoto. “Procedural Generation of Synthetic Forest Environments to Train Machine Learning Algorithms”. En: *ICRA 2022 Workshop in Innovation in Forestry Robotics: Research and Industry Adoption*. 2022. URL: https://openreview.net/forum?id=rpzgjNCe4G9.

[45] Kavisha Vidanapathirana et al. “WildScenes: A benchmark for 2D and 3D semantic segmentation in large-scale natural environments”. En: *The International Journal of Robotics Research* (2024), pág. 02783649241278369. URL: https://arxiv.org/abs/2312.15364.

[46] Yuanzhi Liu et al. “Botanicgarden: A high-quality dataset for robot navigation in unstructured natural environments”. En: *IEEE Robotics and Automation Letters* 9.3 (2024), págs. 2798-2805. URL: https://ieeexplore.ieee.org/abstract/document/10415477.

[47] Peter Mortimer et al. “The goose dataset for perception in unstructured environments”. En: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2024, págs. 14838-14844. URL: https://arxiv.org/abs/2310.16788.

[48] Adam Paszke et al. “Enet: A deep neural network architecture for real-time semantic segmentation”. En: *arXiv preprint arXiv:1606.02147* (2016). URL: https://arxiv.org/abs/1606.02147.

[49] Abhinav Valada et al. “Deep multispectral semantic scene understanding of forested environments using multimodal fusion”. En: *2016 International Symposium on Experimental Robotics*. Springer. 2017, págs. 465-477. URL: https://link.springer.com/chapter/10.1007/978-3-319-50115-4_41.

[50] Daniel Maturana et al. “Real-time semantic mapping for autonomous off-road navigation”. En: *Field and Service Robotics: Results of the 11th International Conference*. Springer. 2018, págs. 335-350. URL: https://link.springer.com/chapter/10.1007/978-3-319-67361-5_22.

[51] Maggie Wigness et al. “A rugd dataset for autonomous navigation and visual perception in unstructured outdoor environments”. En: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019, págs. 5000-5007. URL: https://ieeexplore.ieee.org/abstract/document/8968283.

[52] Peng Jiang et al. “Rellis-3d dataset: Data, benchmarks and analysis”. En: *2021 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2021, págs. 1110-1116. URL: https://arxiv.org/abs/2011.12954.

[53] Peter Neigel, Jason Rambach y Didier Stricker. “OFFSED: Off-Road Semantic Segmentation Dataset”. En: *Proceedings of the 16th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications VISIGRAPP-(Volume 4)*. SciTePress. 2021, págs. 552-557. URL: https://www.dfki.de/fileadmin/user_upload/import/11376_OFFSEDD_VISAPP2021.pdf.

[54] Raphael Hagmanns et al. “Excavating in the Wild: The GOOSE-Ex Dataset for Semantic Segmentation”. En: *ArXiv* abs/2409.18788 (2024). URL: https://api.semanticscholar.org/CorpusID:272968814.

[55] Mingxing Tan y Quoc Le. “Efficientnet: Rethinking model scaling for convolutional neural networks”. En: *International conference on machine learning*. PMLR. 2019, págs. 6105-6114. URL: http://arxiv.org/abs/1805.04687.

[56] Tianrui Guan et al. “GA-Nav: Efficient Terrain Segmentation for Robot Navigation in Unstructured Outdoor Environments”. En: *IEEE Robotics and Automation Letters* 7.3 (2022), págs. 8138-8145. DOI: 10.1109/LRA.2022.3187278.