Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

MÁSTER UNIVERSITARIO EN INGENIERÍA INFORMÁTICA

SEGMENTACIÓN SEMÁNTICA DE IMÁGENES SATELITALES EN GRANDES AGLOMERACIONES URBANAS

TRABAJO FIN DE MÁSTER

Autor: Rubén Sesma Redín

Director: Mikel Galar Idoate

Pamplona, Abril 2019

SUMARIO

1 Introducción 1
1.1 Motivación 3
1.2 Objetivos 4
2 Preliminares 5
2.1 Programa *Copernicus* 5
2.1.1 Misiones Sentinel 6
2.1.2 CLMS: Servicio de Monitorización Terrestre 8
2.2 Sistemas de Información Geográfica 10
2.2.1 Herramientas 10
2.2.2 Georeferenciación y sistemas de coordenadas 11
2.2.3 Datos ráster 13
2.2.4 Datos vectoriales 15
2.3 Deep learning 19
2.3.1 Redes neuronales 19
2.3.2 Redes neuronales convolucionales 24
2.3.3 Segmentación semántica 29
2.4 Tecnologías 32
3 SIWUAC: Sentinel Images With Urban Atlas Classification 35
3.1 Propuesta 35
3.2 Implementación 37
3.3 Conjunto de imágenes sintéticas 48
4 SegNet 51
4.1 Arquitectura original 52
4.2 Modificaciones 55
5 Estudio experimental 57
5.1 Marco experimental 57
5.2 Pruebas realizadas 60
5.2.1 Establecer el tamaño de imagen 60
5.2.2 Reclasificación de Urban Atlas 62
5.3 Resultados 64
5.3.1 Resultados cuantitativos 65
5.3.2 Resultados cualitativos 67
6 Conclusiones 83
6.1 Líneas futuras 85

7 Bibliografía 87
Anexo I - Tabla de figuras 91

1 Introducción

En los últimos años se han producido importantes avances tecnológicos en el sector de la teledetección y el tratamiento de imágenes de satélite. En Europa, se ha impulsado la investigación en el ámbito de la observación de la Tierra gracias al programa *Copernicus* gestionado por la Agencia Espacial Europea (ESA). El programa se compone de las denominadas misiones Sentinel dedicadas al monitoreo de nuestro planeta. Estas misiones hacen posible que cualquier interesado tenga acceso a una amplia gama de productos y servicios satelitales, hecho que ha favorecido la aparición multitud de aplicaciones y servicios basados en el análisis de imágenes satelitales.

La visión artificial o visión por computador es una disciplina que se centra en el desarrollo de métodos para analizar y comprender imágenes del mundo real. Su objetivo general se basa en emular el comportamiento de la visión humana para obtener información relevante a partir de imágenes. Actualmente tiene numerosas aplicaciones como por ejemplo el reconocimiento de caras o gestos en sistemas de identificación personal, la clasificación de imágenes de piezas para el control de calidad, el reconocimiento de matrículas y la detección de objetos en vehículos autónomos, pero también puede ser aplicada al análisis de imágenes satelitales.

Recientemente, la visión por computador ha dado un salto cualitativo gracias al uso de técnicas de *machine learning,* traducido como aprendizaje automático. Este tipo de técnicas han destacado principalmente en la clasificación de imágenes. Sin embargo, los avances recientes en *deep learning* (un subcampo del *machine learning* que hace uso de modelos computacionales denominados redes neuronales profundas) han permitido abordar otros problemas de visión por computador más sofisticados como la detección de objetos o la segmentación de imágenes. Dentro de esta última, la segmentación semántica consiste en otorgar una etiqueta o categoría a cada píxel de una imagen, por lo que los límites de los objetos segmentados pueden tener cualquier forma arbitraria, en cambio los sistemas de reconocimiento y localización de dan como resultado la ventana rectangular donde se ha detectado un objeto.

A pesar de que se está comenzando a investigar la adaptación de redes neuronales de segmentación semántica aplicadas a imágenes aéreas [1], los principales estudios en éste ámbito [2] [3] [4] muestran sus resultados empleando conjuntos de imágenes de la vida real. Estas imágenes y su segmentación ideal, están publicadas gracias a los desafíos online propuestos por PASCAL-VOC *(Pattern Analysis, Statistical Modelling and Computational Learning – Visual Object Classes)* [5] o COCO (*Common Objects in Context*) [6] y son empleadas en las principales investigaciones dentro de este ámbito.

Para nuestro proyecto, dichos conjuntos de datos no han sido suficientes ya que no contienen imágenes de satélite. Este motivo, junto a los productos y servicios satelitales del programa *Copernicus*, nos han llevado a enfocar este proyecto de la siguiente forma: No solamente resolver la segmentación semántica en imágenes satelitales, sino también presentar una herramienta capaz de generar conjuntos de imágenes etiquetadas a nivel de pixel para el entrenamiento de las redes neuronales empleadas.

Por un lado la debemos destacar la facilidad de acceso que brinda *Copernicus* a imágenes de los diferentes satélites a través del *EO Browser* con una interfaz web o empleando APIs para automatizar las descargas desde *Copernicus Open Access Hub*. Una imagen de satélite descargada desde estos servicios no es un fichero de imagen digital corriente, sino que es un fichero comprimido que contiene una gran cantidad de información que hay que procesar. Por otro lado, *Copernicus* ofrece otros productos entre los que destacamos el Urban Atlas 2012. Este es un producto que ofrece información con un gran nivel de detalle acerca del uso y cobertura del suelo en grandes áreas urbanas europeas.

En este trabajo presentamos SIWUAC (*Sentinel Images With Urban Atlas Classifications*), una herramienta que automatiza el proceso que se encarga de la descarga y el pre-procesado necesario de las imágenes de la misión Sentinel-2, hasta la creación de conjuntos de imágenes etiquetadas a nivel de pixel como los de la Figura 1 empleando la información de Urban Atlas. La herramienta permite construir conjuntos de imágenes satelitales personalizados en base a diferentes parámetros, organizados por ciudades y adaptados para alimentar redes neuronales de segmentación semántica. Esta herramienta nos permitirá entrenar redes neuronales capacitadas para clasificar zonas que actualmente no están etiquetadas en Urban Atlas y podría ayudar a crear un Urban Atlas futuro más actualizado.

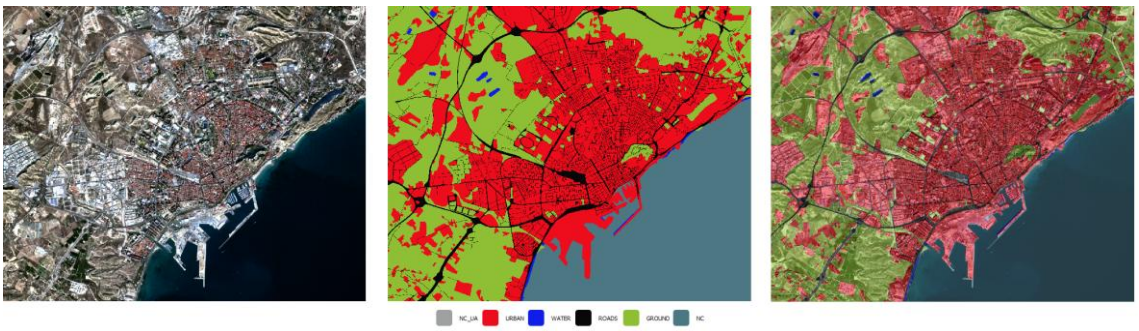


Figura 1 Imagen Séntinel-2 del puerto de Alicante, junto a una posible clasificación generada con SIWUAC

1.1 Motivación

"*Copernicus proporciona conocimiento, pero todo comienza con datos"* (www.copernicus.eu). Uno de los beneficios cruciales del Programa *Copernicus* es que la información y los datos producidos están disponibles de forma completa, abierta y gratuita (sujetos a las condiciones y limitaciones apropiadas) para todos sus usuarios y público en general. Este hecho ha permitido el desarrollo de muchos servicios posteriores basados en sus datos. Por esta razón, la principal motivación del proyecto es introducirnos en el mundo del tratamiento de datos satelitales aprovechando los servicios de *Copernicus* y explotando sus datos con técnicas del *deep learning* como son las redes neuronales convolucionales.

Para resolver el problema de segmentación semántica se emplean arquitecturas denominadas *Fully Convolutional Networks* (FCN). Actualmente existen diversos modelos con arquitecturas muy diferentes como SegNet [7] o DeepLab [8] que consiguen resultados bastante buenos. Este tipo de redes son complejas y profundas, por lo que su entrenamiento necesita, además de una gran cantidad de datos una gran capacidad de computación para procesarlos para conseguir ajustarse hasta obtener buenos resultados.

Para poder trabajar con este tipo de redes aplicadas a imágenes satelitales hemos contado con el apoyo de Tracasa Instrumental, donde he desarrollado este proyecto. Por ello, hemos hecho uso de su infraestructura para las ejecuciones durante la fase de implementación del proyecto.

Figura 2 Logo Tracasa Instrumental

A continuación describimos los objetivos generales y particulares que hemos perseguido.

1.2 Objetivos

Como hemos dicho anteriormente, se han obtenido muy buenos resultados en segmentación semántica aplicada a imágenes de escenas de la vida real, pero su aplicación a imágenes satelitales no está igual de extendida. Un modelo bien entrenado, capaz de clasificar cada pixel de una imagen de satélite puede tener múltiples campos de aplicación. En este caso, estamos limitados a la superficie que cubre Urban Atlas para entrenar los modelos, por lo que estarán ajustados para clasificar escenas donde aparezcan aglomeraciones urbanas. Estos modelos podrán ser utilizados, para aumentar la cobertura de Urban Atlas incluyendo nuevas ciudades o para realizar actualizaciones que cubran más superficie incluyendo nuevas áreas. Por ejemplo, como se puede apreciar en la Figura 3, en Navarra el Urban Atlas 2012 cubre la ciudad de Pamplona y alrededores, pero quedan una gran superficie por catalogar.

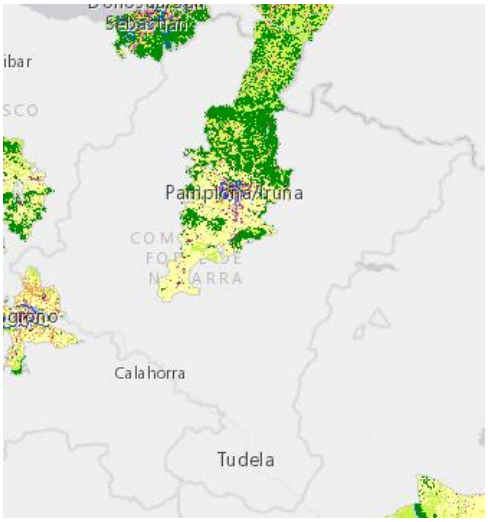


Figura 3 Visor web del Urban Atlas 2012 de Navarra

Urban Atlas se construye en base a la combinación de clasificación (estadística) de imágenes e interpretación visual de imágenes satelitales de muy alta resolución (imágenes SPOT 5 y 6 multiespectrales y Formosat-2 con resolución espacial de 2 a 2,5 metros) por lo que supone unos costes elevados. En cambio las imágenes Sentinel que hemos empleado tienen una resolución espacial máxima 10 metros pero suponen coste cero para el usuario final. En definitiva, el objetivo general de éste proyecto es conseguir un modelo de clasificación basado en redes neuronales de segmentación semántica, capaz de clasificar a nivel de pixel imágenes de áreas urbanas obtenidas de la misión Sentinel-2 con costes mínimos.

Para ello hemos dividido el trabajo en los dos objetivos particulares propuestos:

1. Implementación de una herramienta para la creación de *datasets* satelitales (SIWUAC).
2. Implementación, desarrollo y estudio experimental sobre redes neuronales para la segmentación semántica de imágenes satelitales.

El primero está detallado en el Capítulo 3, la adaptación de redes neuronales a imágenes satelitales en el Capítulo 4 y el estudio experimental realizado queda documentado en el Capítulo 5.

2 Preliminares

Esta sección se divide en cuatro secciones claramente diferenciadas con el objetivo de dotar al lector de los conceptos necesarios para comprender el resto del proyecto. Debido a que nos estamos basando en sus productos y servicios para el desarrollo del proyecto, la Sección 2.1 describe el Programa *Copernicus*, centrándose en las misiones Sentinel y el Servicio de Monitorización Terrestre (CLMS). A continuación, la Sección 2.2 está dedicada a los Sistemas de Información Geográfica (GIS) donde repasamos los conceptos básicos de éste ámbito y detallamos los tipos de datos manejados por estos sistemas. En la Sección 2.3 presentamos una breve cronología de los métodos empleados para resolver la segmentación semántica de imágenes hasta la actualidad. Por último en la Sección 2.4 nombraremos las librerías y tecnologías empleadas durante la implementación del proyecto.

2.1 Programa *Copernicus*

Nos remontamos hasta el Renacimiento, tiempo de *De revolutionibus orbium celestium* título traducido al castellano como *Sobre los giros de los orbes celestes*. Obra en la cual se expone la teoría heliocéntrica y que suele estar considerada como punto inicial de la astronomía moderna y pieza clave en lo que se llamó la Revolución Científica del Renacimiento. Su autor fue *Nicolaus Copernicus* (1473 - 1543) y en él se afirmaba entre otras cosas, que el hombre y la Tierra no eran el centro del universo, y que los planetas giraban en torno al sol. Afirmación que tuvo grandes repercusiones en el ámbito científico, teológico y filosófico de la época y que además es considerada como la causa de su muerte. Por su enorme contribución a la astronomía, en 1935 a modo de homenaje se dio el nombre *Copernicus* a uno de los mayores cráteres lunares, ubicado en el *Mare Insularum.*

El 31 de Mayo de 1975, once países europeos fundaron la ESA. Es una organización internacional dedicada a la exploración espacial, y actualmente cuenta con 22 estados miembros. En las últimas décadas, la sensibilización de la humanidad y de las organizaciones por las consecuencias del cambio climático, ha sido uno de los factores clave para la aparición de misiones dedicadas a la observación y monitorización de nuestro planeta.

En los últimos años, grandes instituciones como la ESA o la Agencia Europea de Medio Ambiente (EEA), han llevado a cabo un gran esfuerzo en I+D dentro del campo de la observación terrestre. El objetivo general de éste esfuerzo se ha centrado entre otras cosas, en mejorar la gestión y conservación del medio ambiente, comprender y mitigar los efectos del cambio climático y asegurar la seguridad civil.

A raíz de este objetivo general surgió la idea de crear un sistema global y continuo de observación terrestre que fue desarrollado bajo el nombre de *Global Monitoring for Enviroment and Security* (GMES). En diciembre de 2012, el Vicepresidente de la Comisión Europea (CE) anunció que también rendían tributo al gran científico con el cambio de nombre del programa GMES por programa *Copernicus*. Actualmente está dirigido por la CE en colaboración con la ESA y la EEA. Desde 2014 ofrece servicios de información totalmente operacionales y de libre acceso para cualquier interesado. El logo actual del programa lo podemos apreciar en la Figura 4.

Figura 4 Logo del programa *Copernicus*

El programa *Copernicus*, es el programa de observación de la Tierra más ambicioso de la historia, diseñado para proporcionar información precisa, actualizada y de fácil acceso para conseguir los objetivos anteriormente expuestos. Se han estimados unos costes de 6700 millones de euros desde 1998 hasta 2020. Costes que son asumidos entre la UE (66%) y la ESA (33%), y unos beneficios aproximados de 30.000 millones de euros para 2030.

Los pilares fundamentales sobre los que el proyecto se sustenta son tres tal y como se puede observar en la Figura 5:

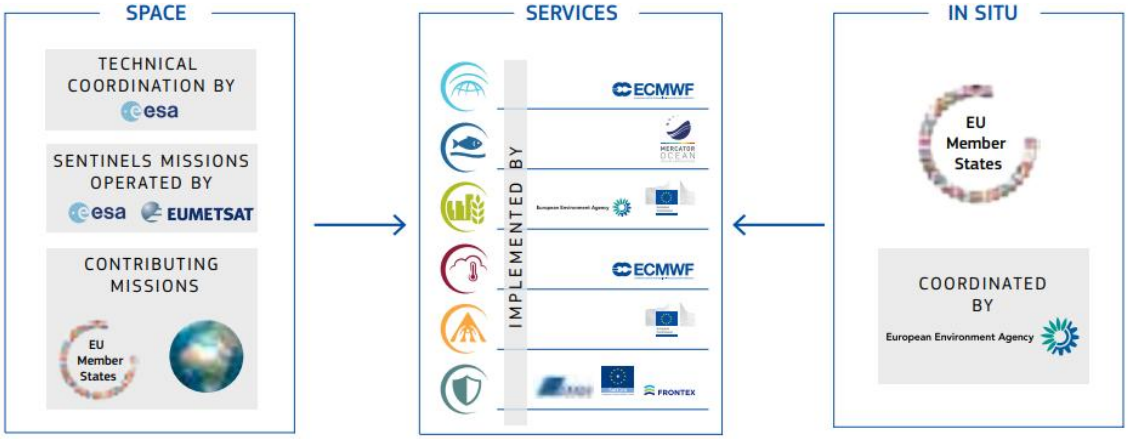


Figura 5 Pilares fundamentales sobre los que se sustenta el programa *Copernicus*

- El segmento espacial gestionado por la ESA: Formado por un conjunto de satélites junto con su infraestructura terrestre. Éste componente agrupa las denominadas misiones Sentinel y misiones de otras agencias.
- El segmento *in-situ* gestionado por la EEA: Infraestructura de red de información terrestre y aerotransportada que recolecta información sobre los océanos, la superficie de los continentes y la atmósfera.
- Los servicios para los usuarios los cuales no serían posibles sin ambos componentes.

2.1.1 Misiones Sentinel

Los satélites que integran el programa de misiones Sentinel, no son satélites de navegación sino satélites de observación del planeta literalmente. Es decir, son centinelas que orbitan y observan la Tierra continuamente con el objetivo de que sus datos sean explotados de forma que proporcionen mejoras en nuestra calidad de vida.

Para la observación y monitorización de los continentes, océanos y la atmósfera, los satélites van equipados con alta tecnología como el radar o los instrumentos de captación de imágenes multiespectrales. Cada una de las misiones está formada por una constelación de al menos dos

satélites cuya función es asegurar la cobertura establecida para cada misión. Estos satélites consiguen generar bases de datos que crecen y se actualizan diariamente para sustentar los servicios ofrecidos por el programa.

A continuación se describe de forma general cada una de la misiones Sentinel.

- **Sentinel-1**: Misión compuesta por la pareja Sentinel-1A y Sentinel-1B, lanzados ambos desde el cosmódromo europeo en la Guayana Francesa el 3 de abril de 2014 y el 25 de abril de 2015 respectivamente. Provee imágenes de radar diurno y nocturno para los servicios oceánicos, terrestres y climatológicos.
- **Sentinel-2**: Una de las piezas clave de este proyecto ya que produce imágenes multiespectrales de alta resolución. Estas imágenes van más allá de lo que el cerebro humano puede interpretar (longitudes de onda del espectro visible) ya que contienen información registrada en diferentes segmentos del espectro electromagnético como el espectro infrarrojo. La instrumentación multiespectral de Sentinel-2 adquiere y ofrece 13 bandas espectrales: cuatro bandas a 10 metros, seis bandas a 20 metros y tres bandas a una resolución espacial de 60 metros. Productos empelados para el monitoreo de la tierra, como el uso del suelo o del agua. Además puede ofrecer información para servicios de emergencias. Sentinel-2A se lanzó el 23 de junio de 2015 y Sentinel-2B se lanzó el 7 de marzo de 2017.
- **Sentinel-3**: Está destinada a tomar medidas acerca de la topografía, temperatura o el color de océanos y continentes. Es un apoyo a los sistemas de predicción oceánica, así como del monitoreo climático. Sentinel-3A se lanzó el 16 de febrero de 2016 y Sentinel-3B se unió a su gemelo en órbita el 25 de abril de 2018.
- **Sentinel-4**: Se lanzará alrededor de 2019 y proporcionará datos para la monitorización de la composición atmosférica. Es decir tomará datos acerca de los gases y aerosoles relevantes para la calidad del aire en Europa en alta resolución espacial con un tiempo de revisión rápido (por hora). No será un satélite independiente, sino que servirá de carga útil embarcada en el *Meteosat Third Generation* (MTG) de la Organización Europea para la Explotación de Satélites Meteorológicos (EUMETSAT). Es decir dotará al MTG de instrumental necesario para realizar su misión.
- **Sentinel-5 Precursor**: Lanzado el 13 de octubre de 2017. Su misión consiste en proporcionar continuidad de datos desde que el *Envisat* (satélite de la ESA para tratar de controlar el calentamiento global) dejó de estar operativo en 2012 hasta el futuro lanzamiento de Sentinel-5.
- **Sentinel-5**: También se dedicará al monitoreo exclusivo de la composición atmosférica y está programado para ser lanzado en 2021. Será también una carga útil embarcada en un *Metop Second Generation* (Metop-SG) de EUMETSAT. Proporciona mediciones precisas de los componentes clave de la atmósfera como el ozono, dióxido de nitrógeno, dióxido de azufre, monóxido de carbono, metano, formaldehido y propiedades de aerosol.
- **Sentinel-6**: Proporcionará altimetría de alta precisión para medir la altura global de la superficie del mar y para estudios climáticos. Es una misión cooperativa desarrollada en asociación entre Europa (UE, ESA y EUMETSAT) y los Estados Unidos (NOAA y NASA). Su lanzamiento está planificado para 2020.

Las misiones corresponden al segmento espacial pero el segmento terrestre también cobra gran importancia en el "control de la misión". Éste segmento controla los satélites Sentinel y las instalaciones terrestres para manejar los datos recibidos, transformarlos y ofrecer productos para los Servicios de *Copernicus*.

2.1.2 CLMS: Servicio de Monitorización Terrestre

Los servicios ofrecidos por el programa se engloban en seis áreas dependientes entre sí: Atmósfera, océanos, tierra, clima, emergencias y seguridad civil. La Figura 6 muestra los logos de las diferentes áreas.

Figura 6 Iconos de los Servicios de *Copernicus*

Para lograr hacer operativos éstos servicios, ha sido necesaria una gran recolecta de información por parte de toda la infraestructura del programa. Esta información es recogida, procesada y puesta a disposición de la comunidad científica o de cualquier otra persona interesada.

En éste proyecto nos centramos en productos ofrecidos por el Servicio de Monitorización Terrestre de *Copernicus* (*Copernicus Land Monitoring Service* abreviado CLMS).

El CLMS se divide en cuatro componentes:

1. **Global**: Produce datos a través de una amplia gama de variables biofísicas a escala global (es decir, a nivel mundial) en media y baja resolución que describen el estado y la evolución de la superficie terrestre como por ejemplo el estado de la vegetación ó el ciclo del agua.
2. **Pan-europeo**: Ofrece conjuntos de datos de alta resolución que describen los principales tipos de cobertura y uso del suelo terrestre: superficies artificiales (por ejemplo, carreteras y áreas pavimentadas), áreas forestales, áreas agrícolas (praderas), humedales, y pequeños cuerpos de agua.
3. **Local**: Proporciona información más detallada que complementa la información del componente Pan-europeo. Se centra en los "puntos calientes" que son propensos a desafíos ambientales detallando el uso y cobertura del suelo.
4. **Imágenes y datos de referencia**: Las imágenes satelitales son la entrada para la creación de productos como mapas de cobertura terrestre o capas de alta resolución. Las imágenes cubren 39 países y están disponibles en forma de mosaicos orto-rectificados y sin fisuras.

En concreto, el componente **Local** ofrece 3 productos principales:

1. **Natura 2000**: Cartografía una selección de sitios ricos en pastizales (lugares donde predomina la vegetación herbácea) y evalúa si esos sitios se están preservando de forma efectiva.

2. **Riparian Zones**: Centrado en controlar la biodiversidad dentro de las principales aéreas a lo largo de los ríos, es decir zonas ribereñas.
3. **Urban Atlas**: Traducido como Atlas Urbano ofrece información detallada sobre el uso y cobertura del suelo en las principales zonas urbanas de la UE, denominadas Áreas Urbanas Funcionales o FUA (*Functional Urban Areas*).

Como hemos dicho anteriormente, las imágenes de Sentinel-2 son pieza clave para nosotros ya que, junto a Urban Atlas constituyen uno de los ejes principales de este proyecto.

2.2 Sistemas de Información Geográfica

En esta sección presentamos una serie de conceptos necesarios que debemos conocer (al menos de forma general) cuando trabajamos con Sistemas de Información Geográfica (GIS). Los datos de *Copernicus* representan el mundo real, por lo tanto se denominan datos espaciales y quedan enmarcados dentro del mundo GIS. Estos datos, actualmente tienen dos formas de representación, formato ráster y vectorial. Cuando decimos que los datos representan el mundo real, nos referimos a que en una imagen, cada pixel representa o contiene información acerca de un área concreta de la superficie de la Tierra. Para determinar la localización exacta de los datos, entra en escena la georeferenciación y los sistemas de coordenadas. Todos estos conceptos los describimos en esta sección y están ligados directamente a las herramientas GIS, por este motivo comenzaremos describiéndolas y nombrando algunas las más usadas en la actualidad.

2.2.1 Herramientas

Se considera un Sistema de Información Geográfica (GIS) a cualquier sistema de información capaz de integrar, almacenar, editar, analizar, compartir y mostrar la información geográficamente referenciada. Además, los GIS son herramientas que permiten a los usuarios crear consultas interactivas, analizar la información espacial, editar datos, mapas y presentar los resultados de todas estas operaciones.

Los datos manejados por estas herramientas deben estar disponibles para ser consultados, transformados, transferidos, superpuestos,..., es decir, geoprocesados. Dentro de la industria comercial y con gran experiencia en el ámbito del geoprocesamiento destacan compañías como ESRI o Autodesk, las cuales comercializan aplicaciones específicas para estos procesos como ArcGIS. Por otro lado, en el software libre de este ámbito destaca la fundación OSGeo la cual da soporte y promueve el desarrollo de proyectos de gran impacto en este ámbito al mismo tiempo que fomenta la interoperabilidad entre herramientas geográficas. Como ejemplo de herramienta de software libre nombramos a *QGIS* ya que ha sido la empleada durante el desarrollo de este proyecto como herramienta de apoyo a la visualización de datos. En la Figura 7 observamos el Urban Atlas de Pamplona visualizado con QGIS.

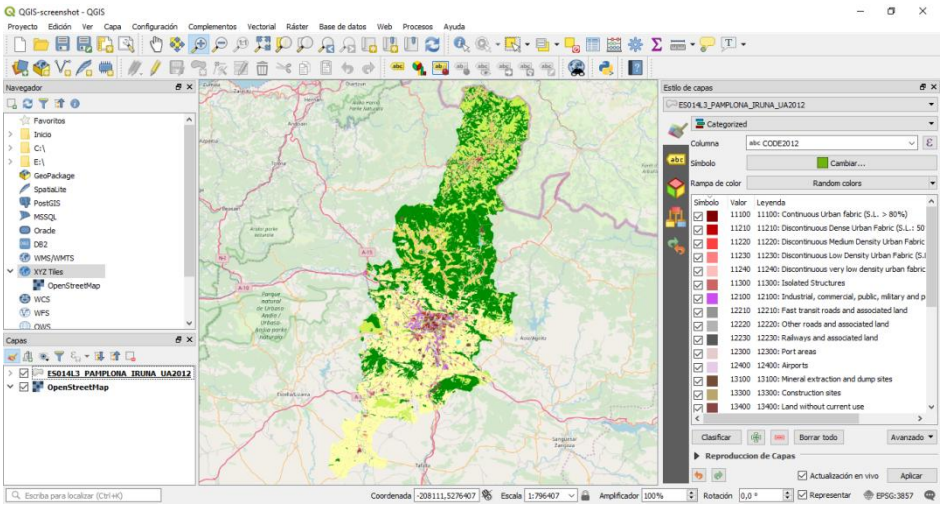


Figura 7 Interfaz de QGIS

2.2.2 Georeferenciación y sistemas de coordenadas

La georeferenciación es un término que empleamos para referirnos al proceso de asociación de un mapa físico o digital con una localización geográfica única y bien definida en un sistema de coordenadas y datum específicos. El término datum se refiere a un conjunto de puntos de la superficie terrestre usados como referencia para tomar mediciones de posición y a un modelo asociado que define la forma de la tierra (elipsoide de referencia) empleado para definir un sistema de coordenadas geográficas.

Existen dos tipos de sistemas de coordenadas según su representación:

- **Sistemas de coordenadas geográficas**, cuando se emplean términos de coordenadas latitud-longitud (Figura 8) asociadas a un datum geodésico específico. El datum más común es el *World Geodetic System 1984* (WGS84), pero en Europa se promueve el uso del *European Terrestial Reference System 1989* (ETRS89).
- **Sistemas de coordenadas proyectadas**, cuando las coordenadas hacen referencia a un plano en el cual se ha proyectado parte de la superficie terrestre modelada con un datum. Esta proyección entre una superficie elipsoidal y un plano, es imposible realizarla sin distorsión. Por este motivo estos sistemas de coordenadas se restringen a regiones suficientemente pequeñas para minimizar esta distorsión. Uno de los más comunes es el sistema de coordenadas universal transversal de Mercator (UTM) el cual usa como base el elipsoide WGS84 y su notación divide el mundo en una rejilla casi regular identificando cada celda con husos (1 - 60) y zonas (C - X). Por ejemplo, en la proyección de la Figura 8 Pamplona se encuentra en la cuadrícula 30T.

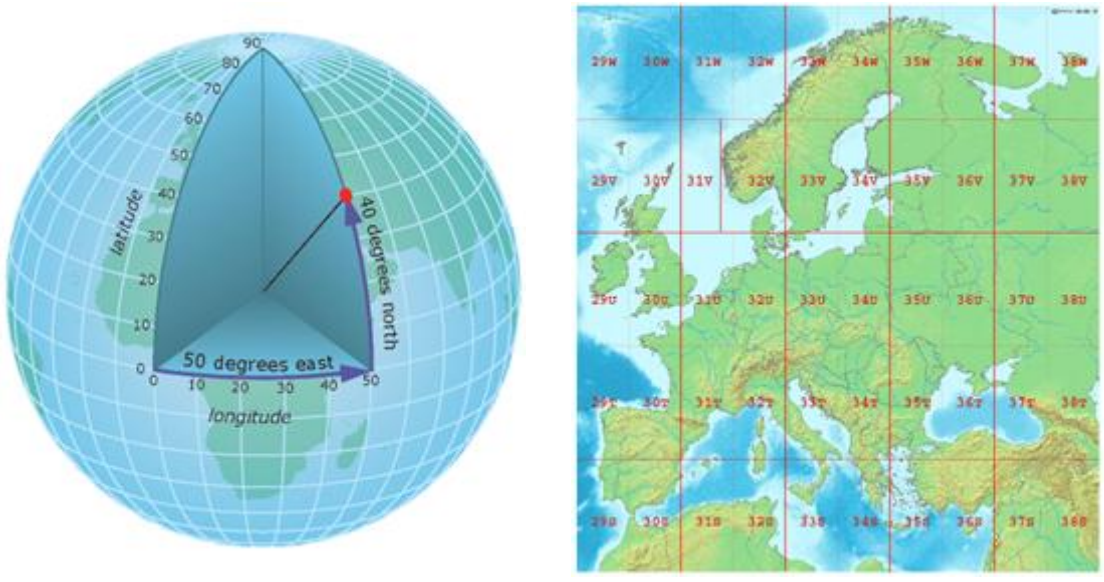


Figura 8 Representación geográfica (a) y proyectada (b)

Cada imagen digital georreferenciada representa una ventana geográfica pero todavía no existe una manera única de almacenar el sistema de coordenadas en el que se han expresado las coordenadas que localizan geográficamente dicha ventana. El método más habitual para identificarlos son los códigos EPSG los cuales fueron difundidos por el *European Petroleum*

Survey Group (EPSG) y están disponibles en una base de datos *Microsoft Access* compatible con la norma ISO 19111 (Define el esquema conceptual para la descripción de la referenciación espacial mediante coordenadas). Esta base de datos contiene información acerca de elipsoides, datums, sistemas de coordenadas, proyecciones, etc. La Tabla 1 muestra algunos de los identificadores más empleados para hacer referencia a los sistemas de coordenadas.

Código EPSG	Sistema de coordenadas	Observaciones
4326	WGS84 Lat Lon	Sistema mundial para dispositivos GPS.
3857	WGS84 Web Mercator ó WGS84 Pseudo-Mercator1	Creado por Google a principios 2005 para Google Maps también empleado por otros servicios de cartografía de Internet como Open Street Map
25829	UTM ETRS89 29T	Sistema de Referencia Terrestre Europeo 1989 referido al huso 29.
25830	UTM ETRS89 30T	Sistema de Referencia Terrestre Europeo 1989 referido al huso 30.
25831	UTM ETRS89 31T	Sistema de Referencia Terrestre Europeo 1989 referido al huso 31.

Tabla 1 Códigos EPSG

Como se aprecia en la Figura 9, para georreferenciar una imagen (localizarla en coordenadas *x* e *y*), es suficiente almacenar información acerca de la geolocalización de la esquina superior izquierda de la imagen (pixel 0,0). Esto es posible debido a que la separación entre pixeles a nivel de coordenadas es constante a lo largo de toda la imagen (filas y columnas). El formato *GeoTIff* (diseñado por la NASA) emplea este modelo para incrustar información geográfica en una imagen de formato *Tagged-Image File Format* (extensión *.tif).*

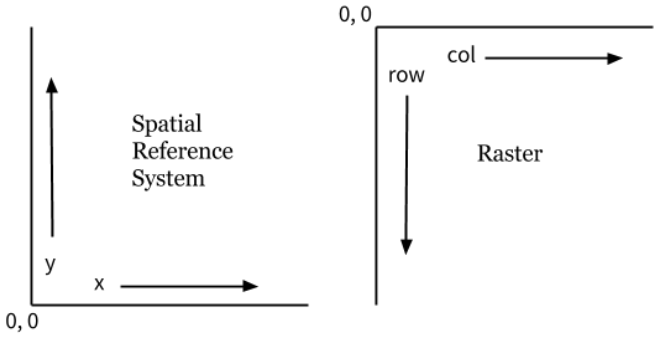


Figura 9 Sistema de coordenadas espaciales y coordenadas de píxel

Este tipo de imágenes son los denominados datos ráster (imágenes geolocalizadas) los cuales los describimos a continuación.

2.2.3 Datos ráster

Los datos almacenados en formato ráster representan fenómenos del mundo real organizados en rejillas de celdas (píxeles) formadas por filas y columnas. Un conjunto de datos ráster (*dataset ráster*) puede ser una fotografía aérea digital, una imagen de satélite, o incluso un mapa escaneado. En los ráster, cada celda posee un valor que representa el fenómeno descrito por el *dataset ráster*. Es decir, los valores pueden ser una categoría, magnitud, altura o un valor espectral. La categoría podría ser una clase acerca del uso del suelo (pradera, bosque, zona urbana, zona industrial, carretera, etc.). Una magnitud podría representar la contaminación acústica o porcentaje de precipitaciones. La altura podría representar la elevación de una superficie por encima del nivel del mar, que se podría emplear para obtener propiedades de la pendiente de un terreno. Los valores espectrales se utilizan en las imágenes de satélite y en las fotografías aéreas para representar la reluctancia (fenómeno en que un espectro de la luz es reflejado por cierto material) de la luz.

Un ráster categórico, contiene una única banda en la que se almacenan los valores asociados a cada categoría, pero también los hay de varias bandas. Esto ocurre con las imágenes de satélite ya que son datos ráster de múltiples bandas espectrales. Es decir, cada banda contiene información acerca de la reflectancia de la luz en diferentes longitudes de onda del espectro electromagnético. Concretamente trabajamos con imágenes Sentinel-2, las cuales son datos ráster que contienen 13 bandas espectrales que abarcan desde el espectro visible (RGB) y el infrarrojo cercano (NIR) hasta el infrarrojo de onda corta (SWIR). Cada una de estas bandas podemos interpretarlas como rásteres de una única banda. Los productos de Sentinel-2 son orto-imágenes de 100x100km^2 en proyección UTM / WGS84 con diferentes propiedades establecidas para cada banda. La Tabla 2 detalla información acerca de las 13 bandas de Sentinel-2 como la longitud de onda (nm), la resolución espacial (m) y la finalidad de cada una de ellas.

Banda	Longitud de onda central (nm)		Resolución espacial (m)	Finalidad
	Sentinel-2A	Sentinel-2B		
1	442.7	442.2	60	Corrección de aerosol
2	492.4	492.1	10	Medición terrestre (Azul) y aerosol
3	559.8	559	10	Medición terrestre (Verde)
4	664.6	664.9	10	Medición terrestre (Rojo)
5	704.1	703.8	20	Medición terrestre (NIR)
6	740.5	739.1	20	Medición terrestre (NIR)
7	782.8	779.7	20	Medición terrestre (NIR)
8	832.8	832.9	10	Medición terrestre y vapor (NIR)
8a	864.7	864	20	Medición terrestre y vapor (NIR)
9	945.1	943.2	60	Corrección de vapor de agua
10	1373.5	1376.9	60	Detección de nubes
11	1613.7	1610.4	20	Medición terrestre (SWIR)
12	2202.4	2185.7	20	Medición terrestre y aerosol (SWIR)

Tabla 2 Detalle espacial y espectral de productos Sentinel-2

Datos satelitales de *Copernicus* en formato ráster

Además de los Servicios de *Copernicus*, los datos satelitales del programa están disponibles desde cuatro puntos de acceso, dos gestionados por la ESA (SciHub y CSCDA) y dos por EUSMESAT (EUMETCast y CODA).

1. ***Copernicus Open Access Hub**:* Anteriormente conocido como *Sentinels Scientific Data Hub* (SciHub). Es un portal que proporciona acceso a los datos de Sentinel a través de una interfaz gráfica interactiva o mediante herramientas como Wget o CURL.
2. ***Copernicus Space Component Data Access*** (CSCDA): Otro portal que proporciona acceso al segmento terrestre de *Copernicus*. Cualquiera puede ver y descubrir datos, pero la descarga de imágenes está restringida a las autoridades públicas, proyectos europeos y servicios de *Copernicus*.
3. ***EUMETCast***: Plataforma que ofrece más de 380 colecciones de diferentes productos de datos ambientales, incluye datos satelitales propios de EUMETSAT, los datos marinos y atmosféricos de *Copernicus* y una amplia gama de otros productos de terceros.
4. ***Copernicus Online Data Access*** (CODA): Es un servicio web que brinda acceso gratuito y abierto a los productos Sentinel-3. El acceso se proporciona a través de la interfaz de usuario CODA o a través de las API CODA. Tanto la interfaz de usuario como las API permiten a los usuarios configurar diferentes parámetros (área geográfica, tiempo, tipo de producto, etc.) para filtrar los productos resultantes de su búsqueda.

Gracias a política de datos completos, abiertos y gratuitos adoptada para el programa hemos tenido acceso a los datos de Sentinel-2 a través de SciHub. Esta plataforma provee acceso a productos de Sentinel-1, Sentinel-2, Sentinel-3 and Sentinel-5P con un simple registro. En concreto vamos a describir los niveles en los que se dividen los productos de la misión Sentinel-2. Estos niveles definen el grado de procesamiento al que han sido sometidos los datos de más bajo nivel:

1. **Nivel-0**: Son datos en bruto que contienen toda la información requerida para generar los productos de nivel 1 y supriores.
2. **Nivel-1A**: Datos sin comprimir, sin procesar, con bandas espectrales y otros datos adjuntos.
3. **Nivel-1B**: Medidas de radiancia corregidos radiométricamente. El modelo geométrico físico se refina utilizando los puntos de control de tierra disponibles.
4. **NIvel-1C** (L1C): Productos con reflectancia *Top-Of-Atmosphere* (TOA) orto rectificada, además incluye máscaras de nubes, tierra y agua.
5. **Nivel-2A** (L2A): Disponibles desde el 28 de marzo de 2017 con reflectancia *Bottom-Of-Atmosphere* (BOA) orto rectificada. En los productos se incluyen un mapa de clasificación de la escena (nubes, sombras de nubes, vegetación, suelos, desiertos, agua, nieve, etc.) a una resolución de 60 metros.

Destacar que los productos L1C se corrigen atmosféricamente utilizando el procesador *Sen2Cor* para generar los productos L2A, por lo que estos productos son publicados 48–60 horas después que los L1C. El resto de niveles no son publicados.

Formato de datos de productos Sentinel-2

Los productos Sentinel-2 están disposición de los usuarios en formato SENTINEL-SAFE, incluyen imágenes en formato JP2 (las bandas ráster espectrales), indicadores de calidad (por ejemplo, máscara de píxeles defectuosos), datos auxiliares y metadatos.

El formato SAFE se ha diseñado para actuar como un formato común para archivar y transmitir datos de la ESA. Está formado por una estructura de carpetas que contienen datos de imagen y metadatos. El formato está diseñado para ser lo suficientemente escalable para representar todos los niveles de productos Sentinel.

Cuando nos referimos a un producto Sentinel-2 nos referimos a un directorio que contiene una gran cantidad de información que se puede apreciar en la Figura 10.

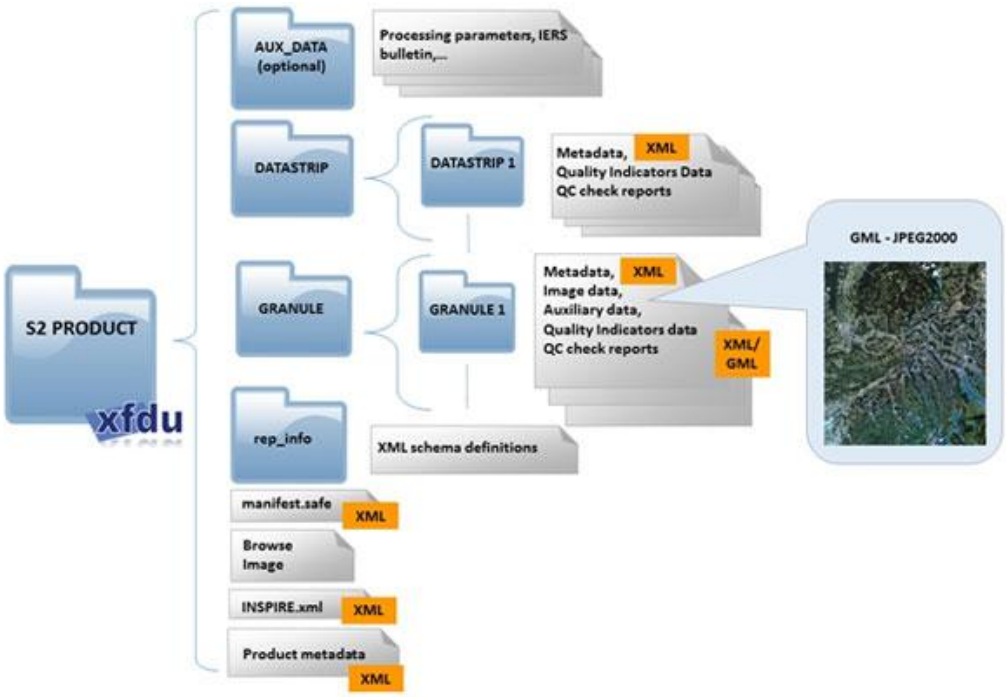


Figura 10 Estructura de carpetas del producto Sentinel-2

A continuación detallamos otro tipo de datos manejados por los GIS y denominados datos vectoriales. Estos son muy diferentes a los ráster pero tienen algo en común, también pueden representar escenas del mundo real por lo que están directamente asociados a la georeferenciación y los sistemas de coordenadas.

2.2.4 Datos vectoriales

Como hemos visto, los datos ráster en definitiva no son más que imágenes geolocalizadas, en cambio los datos vectoriales son bastante diferentes. En este caso, los datos contienen elementos discretos construidos a partir de vértices, y puede que conectados entre sí mediante líneas y/o áreas para representar objetos espaciales. A pesar de que ambos pueden describir propiedades de objetos del mundo real, en los ráster esos objetos no están delimitados, en lugar de ello las formas están representadas utilizando píxeles de distintos valores. La Figura 11 muestra la mima escena representada en formato ráster y vectorial.

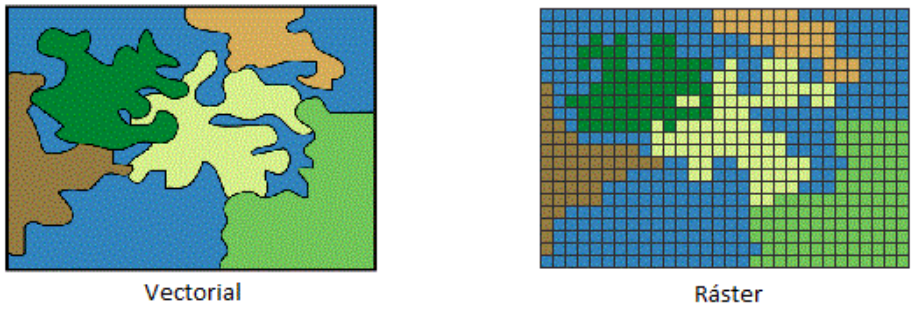


Figura 11 Escena representada en formato vectorial y ráster

Cada representación de un objeto espacial se compone de una geometría y una serie de atributos (opcionales) con información alfanumérica que describe el objeto espacial. La geometría describe la forma (*shape*) del objeto y se compone de uno o más vértices (en un sistema de coordenadas concreto) conectados. Cada vértice describe una posición en el espacio empleando coordenadas (x, y) y opcionalmente un eje z (para representar altura o profundidad en cada vértice, pero no ambos). Los tres tipos básicos de geometrías los podemos visualizar en la Figura 12 y son:

- **Punto**: Geometría formada por un único vértice.
- **Línea**: Geometría compuesta por dos o más vértices siendo el primero y último distintos.
- **Polígono**: Cuando tres o más vértices de la geometría están presentes, y el último vértice es igual a la primero.

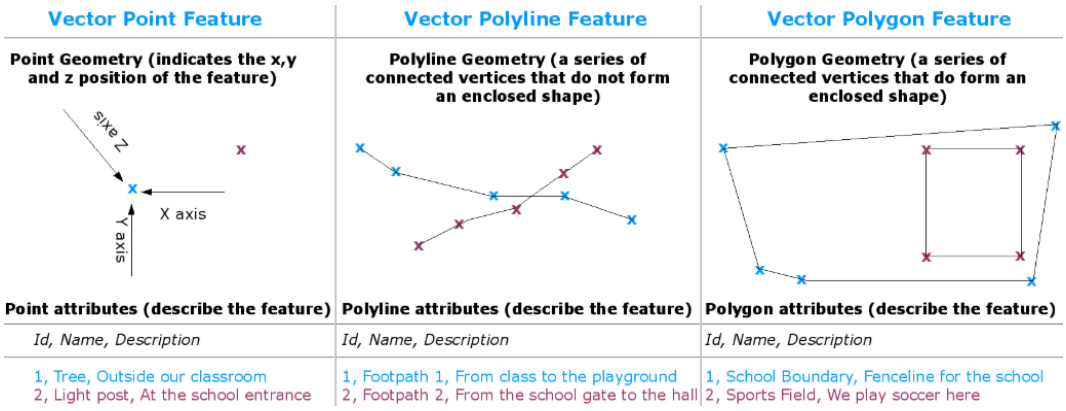


Figura 12 Geometrías básicas de los datos vectoriales

En el caso de los ráster, la resolución espacial hace referencia al área que ocupa cada píxel y por lo tato está directamente relacionada con el espacio necesario para su almacenamiento. Pero en el caso de los datos vectoriales el concepto de resolución espacial hace referencia a la escala del mapa a partir del cual se generaron las geometrías. Para asentar el concepto de datos vectoriales, a continuación describimos en detalle en qué consiste el producto Urban Atlas 2012 ya que está formado por polígonos creados empleando imágenes satelitales con una resolución espacial de 2 a 2,5 metros.

Urban Atlas 2012

Bajo la misma política acceso libre, completo, y gratuito de *Copernicus*, al igual que el SciHub brinda acceso a los satelitales, desde 2016 el componente Local del CLMS pone a disposición de sus usuarios el Urban Atlas 2012 en forma de datos vectoriales. Es un producto o conjunto de datos que proporciona información fiable, inter-comparable y de alta resolución a cerca del uso y cobertura del suelo terrestre para 800 áreas urbanas funcionales (FUA) de 39 países diferentes en el año 2012. Los datos espaciales están disponibles para ser descargados organizados por ciudades (FUA). Al descargar el Urban Atlas de una ciudad concreta, el producto incluye un mapa en formato *shapefile* y un informe con los metadatos para el área en cuestión. La Figura 13 muestra las bandas RGB de una imagen Sentinel2-L1C de la ciudad de Ámsterdam en 2016 a la que se le superpone su correspondiente Urban Atlas empleando QGIS.

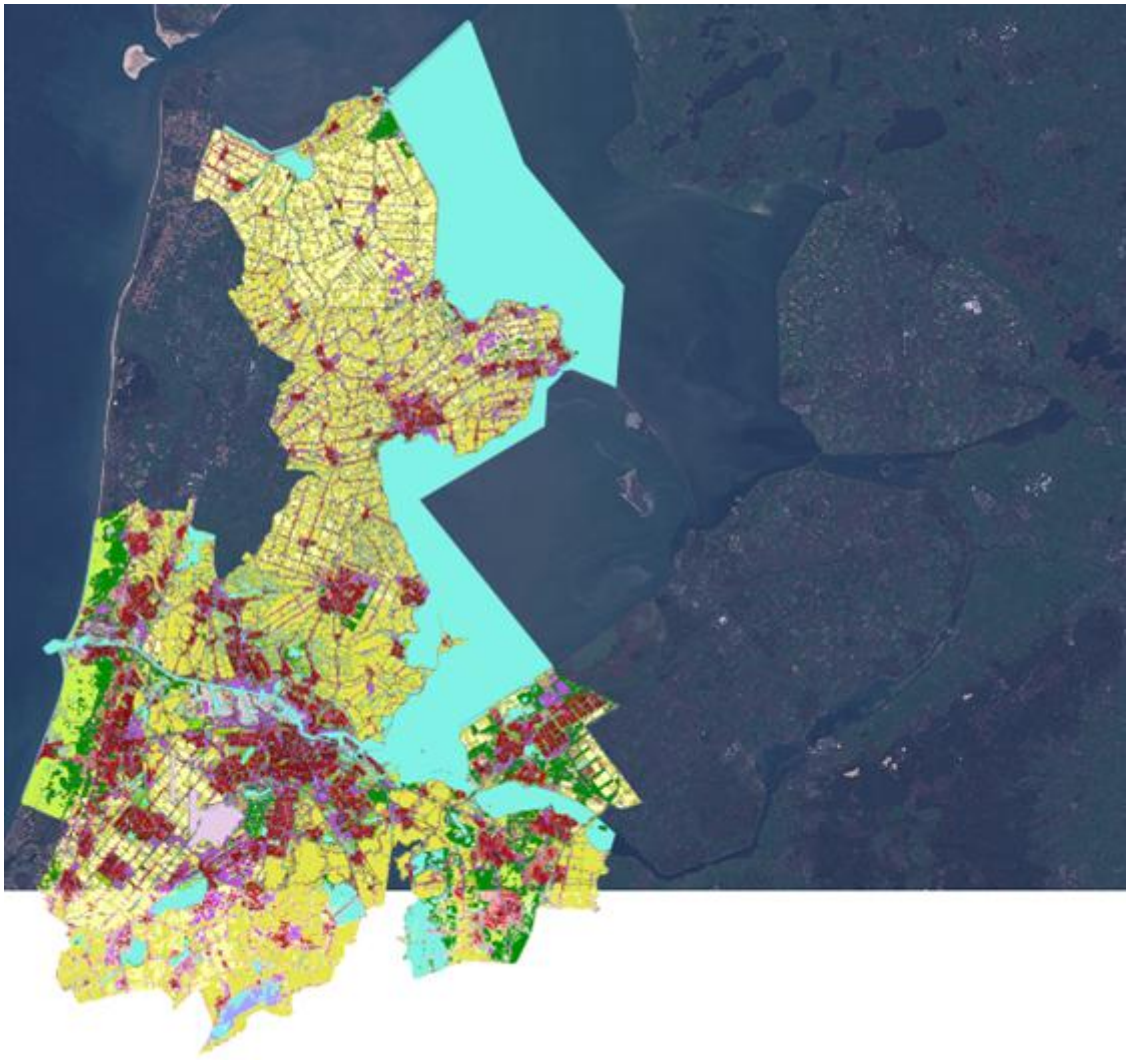


Figura 13 Urban Atlas de Ámsterdam sobre imagen RGB de Sentinel-2 de los Países Bajos

La figura anterior aporta una intuición preliminar de la finalidad general del presente proyecto y el modo de trabajo que hemos seguido. A continuación describimos brevemente el formato de datos más extendido para el almacenamiento de datos vectoriales.

Formato ESRI Shapefile

Actualmente existen multitud de posibilidades de almacenar físicamente datos vectoriales. Algunos de los formatos más empleados son DWG/DXF/DGN, GeoJSON, o las bases de datos espaciales como PostgreSQL + PostGIS u Oracle Spatia las cuales están cogiendo gran fuerza en este ámbito. Pero desde sus orígenes, el formato más estándar y extendido es el formato *shapefile* o *ESRI shapefile* diseñado por la compañía ESRI cuyos productos GIS destacan por estar muy bien documentados como por ejemplo ArcGIS.

Cuando hablamos que una información está en formato *shapefile,* hacemos referencia al menos a tres ficheros con diferentes extensiones y finalidades:

- *.shp*: Almacena las entidades geométricas de los objetos espaciales.
- *.shx*: Contiene el índice de las entidades geométricas.
- *.dbf*: Es un archivo de base de datos donde se almacena la información de los atributos.

Además de estos tres ficheros requeridos, de forma opcional es posible que incluya otros con información adicional:

- *.prj*: Información referida al sistema de coordenadas.
- *.sbn*: Como el *.shx*, almacena el índice espacial de las entidades.
- *.fbn* y *.fbx* Índice espacial de las entidades para los *shapefiles* de solo lectura.
- *.ain* y *.aih*: Almacena el índice de atributos de los campos.
- *.xml*: metadatos del *shapefile*.

Por último, destacar que para que las herramientas SIG interpreten correctamente este formato, todos los ficheros de un *shapefile* deben tener el mismo nombre.

Hasta ahora hemos detallado el origen y formato de los datos que hemos empleado en el proyecto así como herramientas necesarias para su procesamiento y análisis. En el siguiente apartado describimos la finalidad perseguida en este proyecto, la cual consiste resolver el problema de segmentación semántica de imágenes satelitales de Sentinel-2 empleando la información de Urban Atlas 2012.

2.3 Deep learning

Dentro de la Inteligencia Artificial, el Aprendizaje Automático o *Machine Learning* es una disciplina científica que se encarga de construir sistemas (modelos) que optimicen criterios de rendimiento haciendo uso de datos y experiencias anteriores. Existen multitud de modelos muy diversos para múltiples finalidades como por ejemplo los algoritmos genéticos, las máquinas de soporte vectorial o las redes neuronales en las cuales nos centramos en este proyecto. Como se muestra en la Figura 14, el término *Deep Learning* o Aprendizaje Profundo hace referencia a un conjunto de técnicas que hacen uso de redes neuronales cuyas arquitecturas constan de un gran número de capas ocultas.

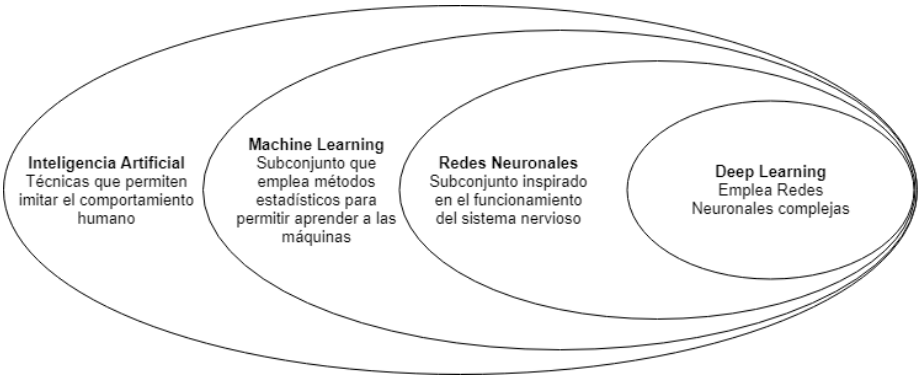


Figura 14 Contextualización del *Deep Learning*

Esta sección se divide en 3 apartados. El apartado 2.3.1 contiene una introducción básica a las redes neuronales, abarcando desde sus orígenes con ejemplos ilustrativos de las redes más simples hasta las redes actuales. El apartado 2.3.2 se centra en describir los conceptos básicos de las Redes Neuronales Convolucionales y su arquitectura. Por último en el apartado 2.3.3 presentamos un breve estado del arte sobre las CNN aplicadas a segmentación semántica.

2.3.1 Redes neuronales

Las redes neuronales están basadas en la combinación de múltiples elementos simples denominados neuronas. En biología, una neurona se compone de un cuerpo celular o *soma*. A partir del soma surgen, un árbol dendrítico compuesto por dendritas y una fibra tubular denominada axón que también se ramifica para conectarse a las dendritas de otras neuronas. La unión entre dos neuronas se denomina *sinapsis*. La principal característica de la señal nerviosa que se transmite entre neuronas es que el pulso generado es “digital”, ya que existe o no existe pulso según se supere cierto umbral (b).

Las neuronas artificiales están inspiradas las neuronas biológicas y se definen formalmente como un elemento que posee un estado interno, denominado nivel de activación y que recibe señales que le permiten, en su caso, modificar dicho nivel de activación y proporcionar una señal de salida. Sus componentes siguen una funcionalidad análoga a los de las neuronas que forman el cerebro humano. La Figura 15 muestra la analogía entre las neuronas biológicas y artificiales.

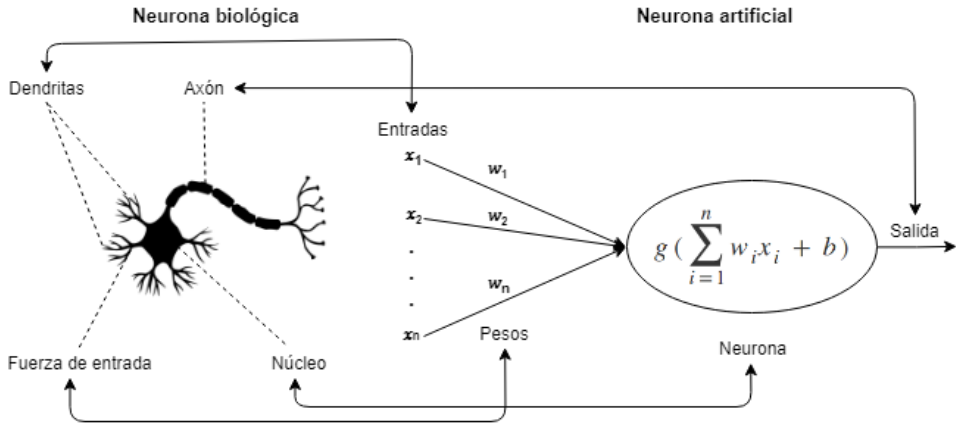


Figura 15 Analogía entre neurona biológica y artificial

Los componentes que forman una neurona artificial son:

- **Entradas**: Sus valores $x_1, x_2, \ldots, x_n$ representan los datos proporcionados a la neurona, ya sea desde el exterior o por otras neuronas y se asocian a las dendritas.
- **Pesos sinápticos**: Representados con $w_1, w_2, \ldots, w_n$, se interpretan como el volumen de neurotransmisores en la sinapsis entre dos neuronas y representan la relevancia de cada una de las interconexiones entre neuronas.
- **Función de propagación**: Se asocia al cuerpo celular donde se procesa la información. Este procesamiento es modelado mediante una función matemática genérica mostrada en la siguiente ecuación, donde w_i y x_i representan cada uno de los valores de los pesos y las entradas respectivamente y la b es el umbral de dicha neurona.

$$\hat{y} = g(\sum_{i=1}^{n} w_i x_i + b)$$

La función $g(x)$ se denomina función de activación de la neurona la cual puede adoptar múltiples formas. La salida que ofrece la neurona está determinada por esta función. Existen varias opciones para emplear estas funciones, a continuación mostramos tres muy extendidas actualmente en las redes profundas.

1. **Sigmoide** $g(\mathrm{z}) = \frac{1}{1+\mathrm{e}^{-\mathrm{z}}}$
2. **Tangente hiperbólica** $g(\mathrm{z}) = \frac{\mathrm{e}^{\mathrm{z}}-\mathrm{e}^{-\mathrm{z}}}{\mathrm{e}^{\mathrm{z}}+\mathrm{e}^{-\mathrm{z}}}$
3. **ReLU** $g(\mathrm{z}) = \max(0, \mathrm{z})$

La Figura 16 muestra las funciones de activación gráficamente:

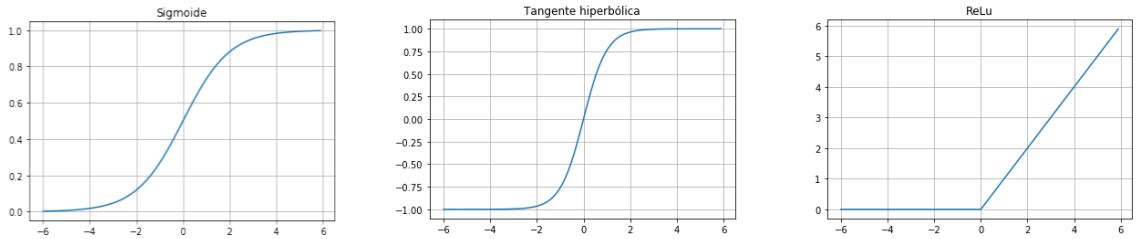


Figura 16 Funciones de activación

En 1958 Rosenblatten [9] creó el primer perceptrón (ver Figura 17), la neurona artificial por excelencia históricamente. El perceptrón es un modelo capaz de realizar tareas de clasificación de forma automática entre elementos de conjuntos de datos linealmente separables. El algoritmo se basaba en aprender los parámetros $w_1, w_2, \ldots, w_n$ que determinaban la ecuación de un hiperplano discriminante entre dos clases a partir de un conjunto de ejemplos etiquetados. En este caso su función de activación no era ninguna de las anteriores ya que simplemente la salida era +1 o -1 para determinar el lado del hiperplano donde quedaba clasificado el ejemplo.

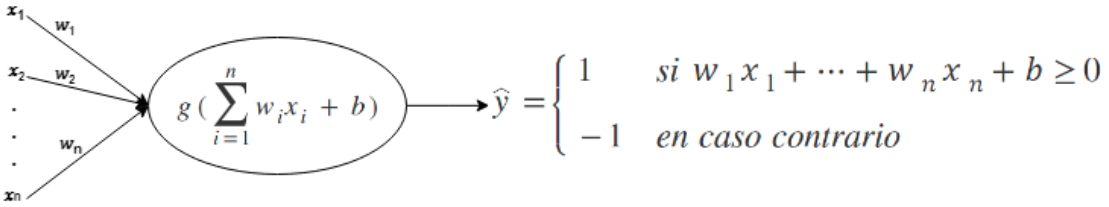


Figura 17 Perceptrón

Diez años después, en 1969 [10] se publicó la idea de crear el perceptrón multicapa (Figura 18) para resolver problemas que no son linealmente separables, lo cual es la principal limitación del perceptrón. Esta publicación además de mostrar cómo combinar perceptrones formando capas para obtener una red capaz de resolver problemas no linealmente separables, también provocó que se perdiera el interés durante unos años en este tipo de modelos. Este hecho en parte fue debido a que la conclusión general del artículo hacía referencia a la incapacidad de estas arquitecturas para resolver problemas matemáticamente triviales como la operación XOR.

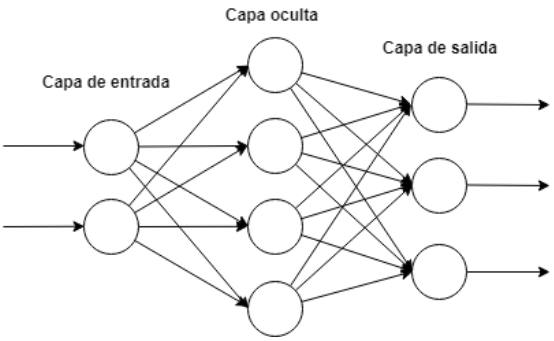


Figura 18 Arquitectura genérica por capas de las redes neuronales

Este tipo de redes han terminado en lo que actualmente conocemos como la arquitectura por capas de las redes neuronales en las que podemos distinguir tres tipos principales capas:

- **Capa de entrada**: Las neuronas solo reciben señales procedentes del entorno, no de otras neuronas.
- **Capa de salida**: Las neuronas proporcionan la respuesta de la red neuronal.
- **Capas ocultas**: Las neuronas no se conectan directamente al entorno, sino con neuronas de otras capas.

El asentamiento de las redes neuronales se puede asociar al año 1974, cuando Paul Werbor propuso una idea para el aprendizaje de pesos y umbrales [11]. La idea fue retomada en 1986

por Rumelhart, Hunton y Williams [12] cuando redescubren el algoritmo de *back-propagation* o de propagación hacia atrás. Adaptaciones de éste algoritmo han desembocado en el método más extendido de minimización del error de las redes neuronales denominado *Descenso por gradiente*.

Este método comienza con la inicialización de los pesos y umbrales de toda la red. Es doblemente iterativo ya que se divide en épocas y en cada época se itera sobre cada uno de los ejemplos del conjunto de entrenamiento empleado propagándolos por la red para conseguir entrenar los pesos. Las operaciones de propagación de cada ejemplo se dividen en dos:

- **Propagación hacia adelante**: El ejemplo se propaga desde la primera capa a través de las capas siguientes de la red, hasta generar la salida. Una vez generada la salida se compara con la salida esperada, entonces se calcula el error del ejemplo en cuestión mediante la denominada *Loss function*. La forma de esta función dependerá del problema que se quiera resolver, por ejemplo en clasificación se emplea el denominado error de entropía cruzada y regresión el error cuadrático medio.
- **Propagación hacia atrás**: El error calculado se propaga partiendo de la capa de salida y pasando por todas las capas ocultas de la red. Las neuronas de estas capas reciben una fracción de la señal total del error. El algoritmo se basa en un modelado matemático basado en derivadas parciales de la función *Loss* que calculan aproximadamente la contribución relativa que cada neurona ha aportado a la salida original. Esta contribución se emplea para modificar los pesos y umbrales de la red.

Al finalizar las operaciones de propagación para todos los ejemplos de entrenamiento se evalúa el error total cometido por la red, el denominado error de entrenamiento. Este proceso se repite durante un número fijado de épocas hasta alcanzar un mínimo del error de entrenamiento.

En torno 1986 fue posible la construcción de las primeras redes neuronales implementadas en silicio gracias a la mejora de las tecnologías de fabricación hardware y la expansión del *back-propagation*. La apertura de la caja de pandora del *deep learning* se produjo a principios de los 90 con la aparición de redes neuronales que empleaban más de una capa oculta (DFF en la Figura 19). El hecho de apilar más capas, en esa época provocó un crecimiento exponencial de los tiempos de entrenamiento, lo que hacía pensar que los modelos de redes neuronales con varias capas ocultas eran poco prácticos.

En los últimos 20 años han ido apareciendo tecnologías basadas en Unidades de Procesamiento gráfico (GPU) que han hecho posible el entrenamiento de este tipo de redes en tiempos asumibles. Este hecho junto a la aparición de grandes y diversas bases de datos anotadas, han provocado que el zoológico de arquitecturas de redes neuronales haya experimentado un crecimiento exponencial. Por este motivo es muy difícil hacer un seguimiento de todas ellas. En 2016 Fjodor van Veen (instituto Asimov) creó una tabla mostrada en la Figura 19 que recoge todas las tipologías de redes neuronales existentes hasta el momento.

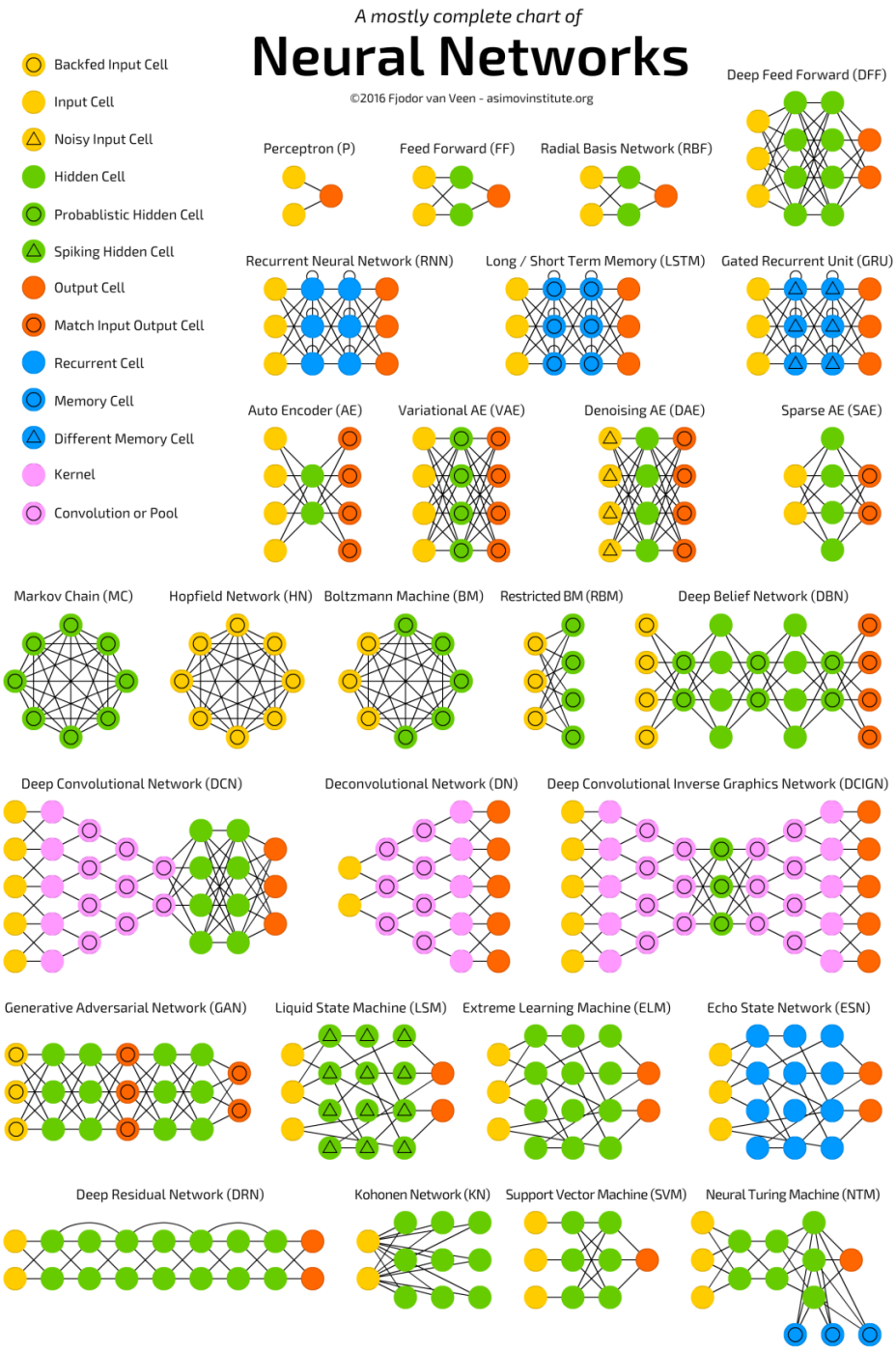


Figura 19 Topologías de redes neuronales

2.3.2 Redes neuronales convolucionales

El origen de las CNN se asocia al Neocognitron, introducido por Kunihiko Fukushima en 1980 [13] pero no fue hasta 2012 cuando fueron implementadas para GPU y alcanzaron resultados impresionantes [14]. Al ser una variación del perceptrón multicapa, también están inspiradas en la forma en que las neuronas de las cortezas visuales del cerebro humano perciben los estímulos visuales. En el campo específico de la visión artificial en el que nos centramos en este proyecto, las CNN tienen por objetivo el análisis de imágenes. Por ello, los datos de entrada a la red son imágenes, es decir arrays de $M \times N \times C$ elementos que representan: *M* el alto de la imagen, *N* el ancho y *C* el número de canales por cada píxel.

En las redes neuronales muy profundas, la extracción de características de los datos de entrada se realiza de forma jerárquica por capas. Este fenómeno se muestra en la Figura 20 donde se ilustra una red neuronal clásica con varias capas intermedias a la que se le suministra un conjunto de imágenes de rostros a la capa de entrada. Las primeras capas extraen características de bajo nivel como pueden ser bordes o texturas presentes en las imágenes. En las capas ocultas intermedias se extraen formas a partir de zonas de los objetos representados en las imágenes como pueden ser los ojos o la boca. Por último las capas finales las redes son capaces de extraer características de alto nivel como son los propios objetos representados en las imágenes, en este caso las caras. Este tipo de redes con múltiples capas ocultas han conseguido excelentes resultados en ámbitos de la visión artificial, la traducción automática o el procesamiento de lenguaje natural.

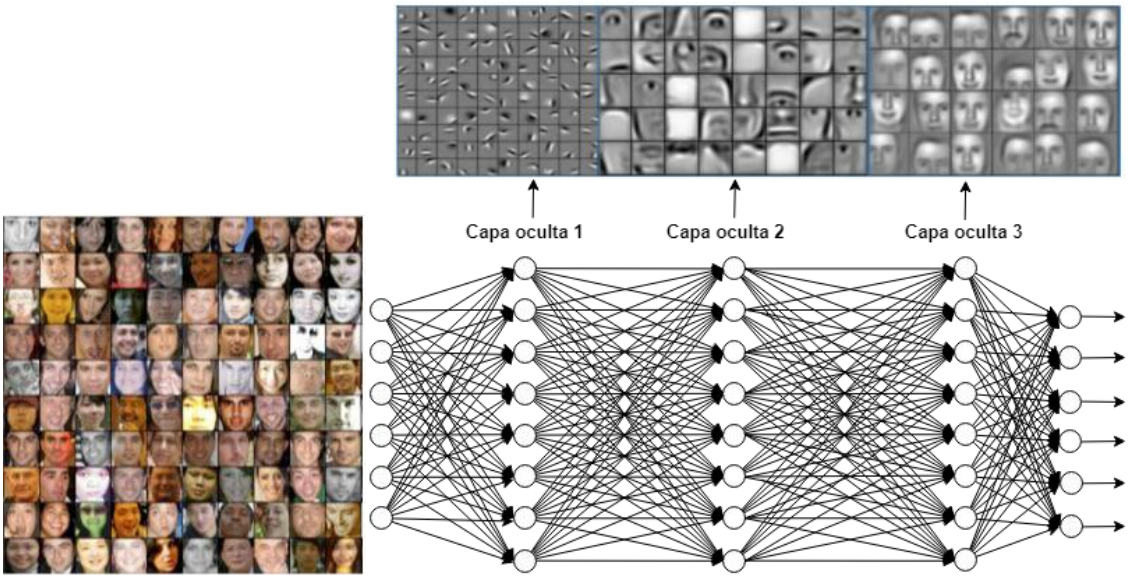


Figura 20 Extracción jerarquizada de características en CNNs

Las redes neuronales convolucionales, son redes profundas en las que las capas ocultas de la red no tienen porqué estar completamente conectadas con las capas adyacentes como es el caso de las capas de la red de la Figura 20, sino que las capas están diseñadas para que realicen diferentes operaciones. A continuación describimos formalmente las operaciones que realizan los diferentes tipos de capas ocultas de las CNN.

Capas convolucionales

La operación básica que realizan estas capas y que permite que las CNN extraigan características es la operación convolución. Esta es una operación matemática que transforma dos funciones (f y g) en una tercera función que muestra la magnitud de intersección entre ambas funciones. Si estas funciones son funciones discretas en dos dimensiones (imágenes), tenemos la operación convolución de imágenes donde f corresponde con la imagen de entrada y la función g corresponde a una pequeña matriz (filtro) que trasladaremos por todo el dominio de f. Esta operación mide, para cada píxel, la relación entre f y g. Formalmente: Sea f una imagen de $M \times N$ píxeles, sea g un filtro de de $m \times n$ píxeles, donde $m = 2a + 1, n = 2b + 1$ con m y n números impares. La convolución de f y g es una nueva imagen en la que, cada pixel (x, y) viene dado por:

$$f * g(x, y) = \sum_{i=-a}^{a} \sum_{j=-b}^{b} g(i, j) f(x + i, y + j)$$

Los valores que componen los filtros determinan la característica que se extrae de la imagen tras la operación sobre todos los píxeles de la imagen. Por ejemplo, con el primer filtro de $f = 3$ (en este caso, f no representa una función sino una constante para identificar el tamaño del filtro) que observamos en la Figura 21, es posible detectar bordes verticales ya que el valor de salida en la convolución será alto cuando un pixel tenga valores muy diferentes a la derecha y a la izquierda. De igual forma, si el filtro lo rotamos 90 grados podremos detectar los bordes horizontales.

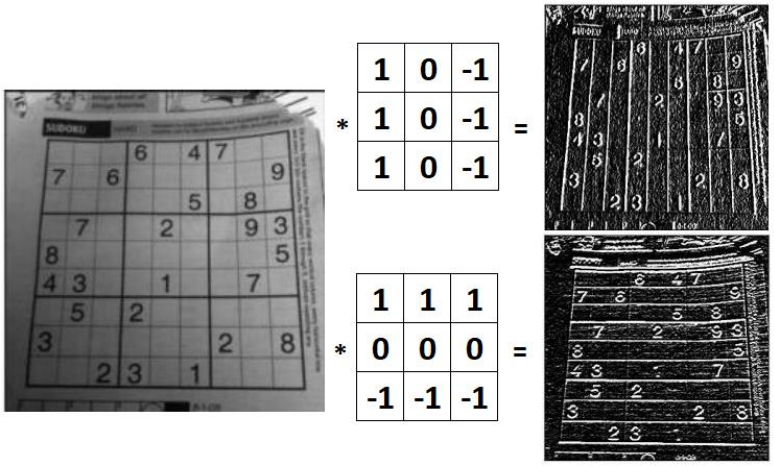


Figura 21 Ejemplo de filtros convolucionales para detección de bordes

El ejemplo de la Figura 21 muestra la operación convolución sobre una imagen de tamaño $M \times N \times 1$ (dos dimensiones o escala de grises) y filtros de $f \times f \times 1$ con $f = 3$, pero también es posible hacer convoluciones sobre imágenes que son volúmenes representados en arrays de $M \times N \times n_c$ donde n_c representa el número de canales de la imagen. En este caso, simplemente cambiará la dimensión del filtro que será de $f \times f \times n_c$ ya que el resultado seguirá siendo una imagen de 2 dimensiones. Por lo tanto, las CNN son válidas tanto para imágenes en escala de grises ($n_c = 1$), RGB ($n_c = 3$) u otras imágenes con más canales como es el caso de las imágenes Sentinel que constan de 13 bandas/canales ($n_c = 13$).

Podemos dividir una capa convolucional en al menos 4 elementos principales:

- **Entrada**: Volumen $m \times n \times n_c$
- **Filtros**: $f \times f \times n_f$
- **Función de activación** (normalmente ReLu) **y umbrales** b de cada neuróna
- **Salida**: Volumen $n - f + 1 \times n - f + 1 \times n_f$

El volumen de salida puede cambiar aplicando otro tipo de operaciones que se realizan en estas capas y que modifican el proceso de convolución:

- ***Padding***: Se emplea cuando nos interesa que el volumen de salida sea $m \times n$ tras la convolución, es decir el mismo volumen que la entrada. Supongamos que tenemos una imagen de dimensión $m \times n$ y queremos aplicar un filtro de $f = 3$. Modificamos la imagen original añadiendo una nueva fila/columna "ficticia" por encima/debajo e izquierda/derecha. La nueva imagen será de dimensión $m + 2 \times n + 2$. Si tras esta modificación ahora aplicamos el filtro 3x3 sobre la nueva imagen, entonces el resultado es $(m + 2 - f + 1) \times (n + 2 - f + 1) = m \times n$. A esta modificación le llamamos aplicar *padding* con $p = 1$.
- ***Stridding***: Se emplea cuando queremos el caso contrario, que la imagen resultante sea todavía mucho más pequeña que la original. Supongamos que una vez aplicada la máscara de convolución sobre el primer bloque, nos movemos dos píxeles más a la derecha en lugar de solo uno. A esto se le denomina convolución con *stride* de $s = 2$.

En general, en una capa convolucional se aplican n_f filtros a la imagen (volumen) de entrada que permiten la extracción de características, pero lo interesante es que, no es necesario fijar manualmente los valores de estos filtros, sino que son calculados por la red durante el entrenamiento. Es decir, es la red quien determina qué filtros emplear para la extracción de características en cada capa. Tras aplicar una convolución sobre una imagen con un filtro de dimensión f, con *padding* p y *stridding* s es: $\left\lfloor \frac{m+2p-f}{s}, \frac{n+2p-f}{s} \right\rfloor$ donde $\lfloor\ x\ \rfloor$ es la operación *floor* ($\lfloor 4.75 \rfloor = 4$).

Por último, en las redes neuronales habituales, cada elemento de una capa está totalmente conectado con las salidas de la capa anterior, en cambio en las capas convolucionales cada elemento está conectado con unos pocos elementos de la capa anterior, los necesarios para calcular las convoluciones.

Capas de *pooling*

Una capa *pooling* se define mediante los siguientes parámetros:

- Tamaño del filtro f
- Tamaño del *stride* s
- Operación que realiza la capa, generalmente máximo (denominada capa *max-pooling*) o media aritmética (*avg-pooling*)

En este caso, el filtro no hace referencia a un filtro de convolución sino al tamaño de la ventana de la imagen original sobre la que se realiza la operación *max* o *avg.* Las capas de *pooling* aceleran el cálculo ya que reducen significativamente la dimensión de los volúmenes. Aportan robustez ya que en el momento que una característica particular ha sido encontrada, se representa mediante un número elevado. Por ejemplo el *max-pooling* ayuda a mantener todos los valores altos que representan las características que han sido detectadas. En la Figura 22 se aprecia el resultado de aplicar *max-pooling* con $f = 2$ y $s = 2$.

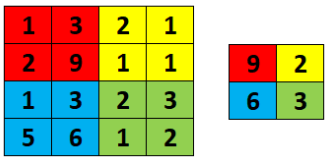


Figura 22 Ejemplo de *max-pooling*

Existen otro tipo de capas en las CNN como pueden ser las capas de normalización o las capas completamente conectadas. Las capas de normalización se suelen emplear tras cada capa convolucional y operan sobre cada uno de los canales de las imágenes. Estas capas realizan una normalización del color de los canales en base a diferentes criterios como podría ser el número de imágenes cargadas en memoria durante el entrenamiento (*batch*). Las capas completamente conectadas son las capas del perceptrón multicapa en las que las neuronas de cada capa están completamente conectadas con las neuronas de las capas adyacentes.

Arquitectura

A pesar de no ser estándar, en las arquitecturas de las CNN suele aparecer el siguiente patrón: Una primera fase de capas encargadas de la extracción de características formadas por varios bloques de capas convolucionales, de normalización y *pooling* encadenados en este orden *(subsampling)*. Después llega un momento en el que transformamos la red en una arquitectura "habitual" mediante la operación *flatten* o aplanamiento y las capas siguientes son capas completamente conectadas al igual que las del perceptrón multicapa, encargadas de las tareas de clasificación. A continuación, en la Figura 23 mostramos la arquitectura de LeNet-5, una CNN clásica [15] para clasificación de imágenes en la que se aprecia el patrón anterior.

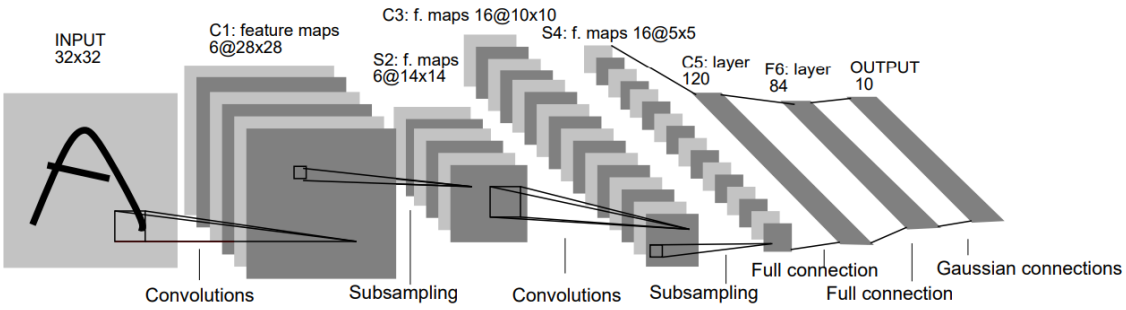


Figura 23 Arquitectura de LeNet-5

La Figura 24 muestra de nuevo este patrón en AlexNet, creada por Alex Krizhevsky, Ilya Sutskever y Geoffrey Hinton para ganar el ILSVRC 2012, desafío de reconocimiento visual a gran escala de ImageNet [16] y su publicación [17] ha tenido una gran influencia en éste ámbito.

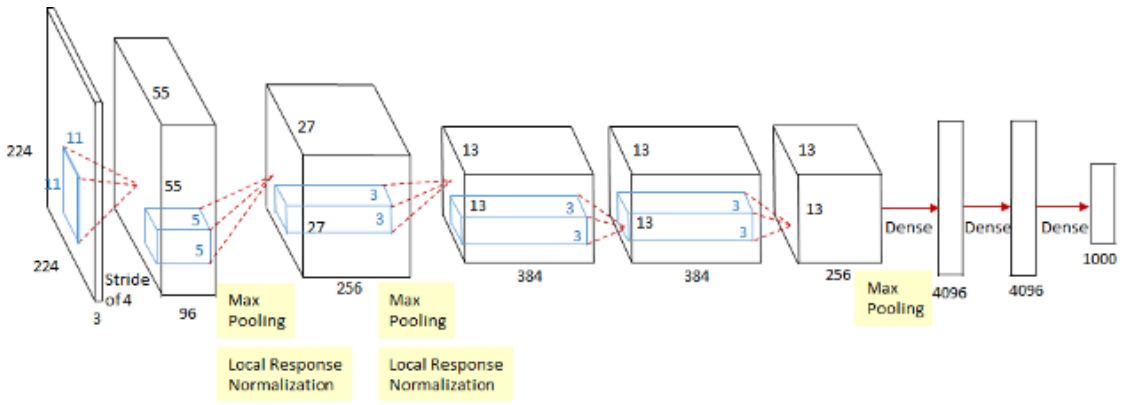
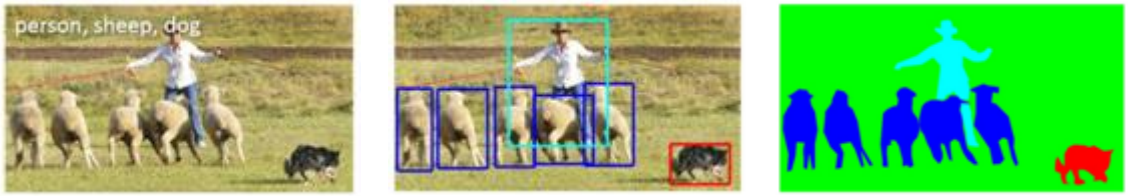


Figura 24 Arquitectura de AlexNet

La capa de salida es la que determina el problema concreto que resuelve la red. Fijándonos en los ejemplos anteriores, LeNet tiene 10 neuronas en su capa de salida y AlexNet 1000. Esto se debe al número de clases que tiene problema de clasificación que se resuelve en cada caso: LeNet diferencia dígitos manuscritos de 0 a 9 y AlexNet venció en 2012 en la competición a gran escala donde se diferencian 1000 clases de imágenes diferentes.

2.3.3 Segmentación semántica

Las CNN actuales, además de estar ampliamente capacitadas para resolver problemas de clasificación y segmentación de imágenes o la localización de objetos en éstas, también pueden ser aplicadas a series temporales o fragmentos de audio. En nuestro nos centramos en la tarea de segmentación semántica. Como ya lo hemos avanzado, este problema requiere tanto de la segmentación como de la clasificación de los objetos segmentados. Es decir, requiere asignar a cada píxel de la imagen una etiqueta o clase.



(a) (b) (c)

Figura 25 Clasificación (a), localización de objetos (b) y segmentación semántica (c) [18]

En cuanto a los conjuntos de datos, como ya hemos dicho en el Capítulo 1, la mayoría de trabajos emplean escenas cotidianas de la vida real con imágenes corrientes en RGB como las de COCO o PASCAL-VOC. También, debido al gran desarrollo tecnológico que están experimentando los vehículos autónomos, están apareciendo conjuntos de datos como Cityscapes y herramientas de etiquetado de fotogramas de vídeos pensados para crear conjuntos de imágenes de escenas urbanas. Estos conjuntos de imágenes son empleados para entrenar modelos los cuales están pensados para ser incluidos dentro del sistema de ayuda a la toma de decisión en vehículos autónomos.

Por este motivo y a pesar de no ser el tema que nos concierne, reducir la carga computacional de estos modelos está siendo otro de los temas actuales de investigación en este ámbito [19] [20] [21] . Este es el caso de las cámaras para vehículos autónomos, cuyos modelos deben funcionar segmentando sus fotogramas en tiempo real. En nuestro caso, la carga computacional no es lo más importante ya que una aplicación basada en imágenes Sentinel, el tiempo real es poder seguir el ritmo del satélite que es “lento”.

Las arquitecturas más destacables de las CNN de segmentación semántica son las FCN (*Fully Convolutional Networks*) [22]. Estas redes son una adaptación de las CNN de clasificación tradicionales donde, como se aprecia en la Figura 26 , se trata de convertir las últimas capas de la CNN, en capas convolucionales totalmente conectadas (con filtros de 1x1). A esto se le suma un mecanismo de *upsampling* hasta obtener una salida del mismo tamaño que la entrada y poder hacer clasificaciones a nivel de pixel.

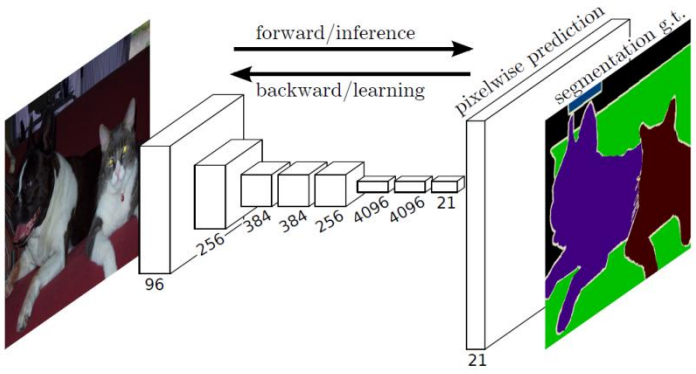


Figura 26 FCN aplicada a segmentación semántica

Uno de los problemas específicos de las FCN se debe a que propagando los ejemplos por capas convolucionales y de *pooling* alternadas, el mapa de características que se obtiene es de dimensión inferior a la original. Por este motivo, las predicciones obtenidas directamente con una FCN suelen ser de baja resolución, lo que provoca que los bordes de los objetos queden poco nítidos, difuminados. En los últimos años se han propuesto una variedad de enfoques más avanzados basados en FCN para abordar este problema, pero la arquitectura general para segmentación semántica puede considerarse la mostrada en la Figura 27 la cual se ha denominado *encoder-decoder* y es implementada por modelos como SegNet [7] o U-Net [23]. Como su nombre indica, consta de dos elementos principales:

- **Codificador**: Al igual que las CNN clásicas, recibe la entrada a la red y genera mapas de características mediante convoluciones y *pooling* que representa la información codificada de la entrada en forma de volumen de características.
- **Decodificador**: Podemos verlo como otra red que generalmente sigue la misma estructura de red que el codificador pero en orientación opuesta. Esta red toma como entrada el mapa de características generado por del codificador. Mediante un proceso de *upsampling,* el decodificador puede usar información del para reconstruir su entrada con las características más importantes extraídas en la primera fase y conseguir así que la salida final sea reconocible como la entrada.

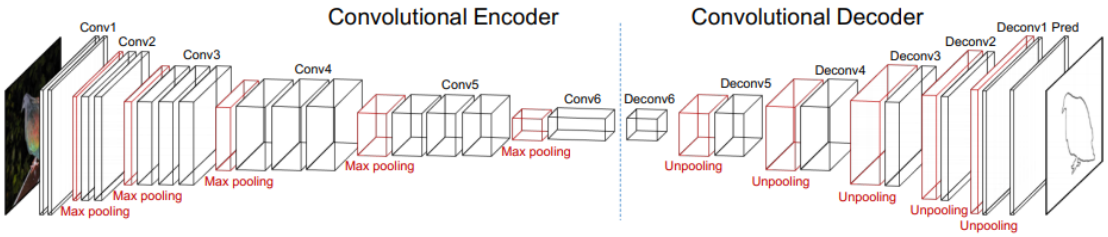


Figura 27 Arquitectura clásica de red *encoder-decoder* [24]

Existen otros enfoques como el de DeepLab [25] que se basa en emplear un procesamiento adicional al final de la red que consiste en un campo aleatorio condicional (CRF). El CRF ayuda a producir líneas y esquinas más definidas en la salida. Además han aparecido modelos como CRFasRNN [26] que integran los CRF a lo largo de la estructura de la red, pero esto implica un aumento de la carga computacional de los modelos. En DeepLabv2 [27] integran en la

arquitectura original redes de capas residuales como ResNet [28] para obtener resultados realmente precisos. Desde que las capas residuales fueron propuestas con ResNet, el uso de este tipo de capas se ha vuelto muy habitual en las CNN profundas (DCNN). Esto es debido a que este tipo de capas permite el uso de redes neuronales profundas al resolver el problema de la degradación o desvanecimiento del gradiente. Este problema ocurre cuando tenemos una red con muchas capas ya que el valor del gradiente cada vez es más cercano a cero porque multiplicamos muchas veces un valor pequeño. Debido a este problema, las primeras capas de una red neuronal son las más lentas y difíciles de entrenar ya que el valor del gradiente que se usa para actualizarlas en cada iteración del entrenamiento es muy pequeño. Y esto causa otro problema, que si las primeras capas no están bien entrenadas, el problema se arrastra a las capas posteriores.

El enfoque de DeepLab ha seguido avanzando y su última propuesta fue DeepLabv3 [8]. A diferencia del enfoque *encoder-decoder*, presenta una arquitectura para controlar la reducción de la señal y el aprendizaje de funciones de contexto a múltiples escalas. En las versiones anteriores, DeepLab utiliza una ResNet entrenada previamente con ImageNet como su principal red de extracción de características. Sin embargo, en ésta versión, proponen un nuevo bloque residual para la extracción de características a múltiples escalas. En lugar de convoluciones estándar, el último bloque de ResNet usa convoluciones dilatadas (*atrous convolutions*). Además, cada convolución (dentro de este nuevo bloque) emplea diferentes ratios de dilatación para capturar el contexto de múltiples escalas. Por último, proponen emplear en la última parte de este nuevo bloque, la denominada *Atrous Spatial Pyramid Pooling* (ASPP) la cual emplea convoluciones con diferentes tasas de dilatación diseñadas para capturar el contexto. ASPP incorpora características de nivel de imagen a través de *Global Average Pooling* (GAP) para agregar información de contexto global.

Además de la arquitectura *encoder-decoder* y la filosofía de DeepLab, existen otras variantes para la segmentación semántica las cuales todas cuentan con capas residuales:

- RefineNet [29] emplea un refinamiento de la salida de la red que empleaba toda la información extraída durante el proceso de *downsampling* y las conexiones residuales para conseguir segmentaciones de gran precisión.
- FRRN [2] combina una red de capas residuales de arquitectura similar a ResNet con un módulo de extracción de contexto.
- LRR [30] propone emplear una pirámide laplaciana para combinar las características extraídas a diferentes escalas.

2.4 Tecnologías

Toda la fase de implementación de este proyecto ha sido desarrollada con Python ya que cuenta con una comunidad que ha desarrollado una gran cantidad de librerías de código abierto dedicadas a una amplia variedad de cosas: tratamiento de ficheros, serialización, comunicación web, análisis y visualización de datos, etc. A continuación describimos la brevemente las principales librerías que hemos empleado en este proyecto:

Rasterio: Permite manejar ficheros de formato GeoTIFF capacitados para almacenar los denominados *datasets ráster* como es el caso de las imágenes de satélite o modelos digitales del terreno. Esta librería está basada en matrices Numpy N-dimensionales y el formato GeoJSON.

Antes de Rasterio, existía una opción de Python para acceder a los diferentes tipos de archivos de datos ráster utilizados en el campo GIS: los enlaces de Python (*Python bindings*) distribuidos con la Biblioteca de Abstracción de Datos Geoespaciales, GDAL. Estos enlaces proporcionan poca abstracción para la API C de GDAL lo que significa que los programas Python que los usan tienden a leerse y ejecutarse como los programas en C. Por ejemplo, los enlaces Python de GDAL requieren que los usuarios estén atentos a los punteros típicos de C, en cambio Rasterio proporciona además del alto rendimiento de GDAL, un código más limpio y transparente.

GeoPandas: Su objetivo es facilitar el trabajo con datos geoespaciales en Python. Combina las capacidades de Pandas (manipular tablas numéricas y series temporales) y Shapely (manipulación y análisis de objetos geométricos planos) por lo que proporciona una interfaz de alto nivel para las geometrías de Shapely, a la vez que permite operaciones geoespaciales mediante Pandas que de lo contrario requerirían una base de datos espacial como PostGIS.

Keras: API de alto nivel para de redes neuronales, implementada en Python y capaz de ejecutarse sobre TensorFlow [31], CNTK o Theano. Fue desarrollada enfocada a permitir la experimentación rápida, por lo que permite crear modelos y prototipos de redes de forma fácil y rápida. Admite redes neuronales convolucionales y recurrentes, así como combinaciones de ambas y funciona a la perfección en CPU y GPU.

Sentinelsat: Facilita la búsqueda, descarga y recuperación de los metadatos de imágenes satelitales de Sentinel desde el *Copernicus Open Access Hub*. Ofrece una interfaz de línea de comandos fácil de usar y una poderosa API de Python.

EO-Learn: Es una colección de paquetes de código abierto de Python que se han desarrollado para acceder y procesar las secuencias de imágenes *espacio-temporales* adquiridas por cualquier flota de satélites de manera automática. Tiene un diseño modular y fomenta la colaboración, comparte y reutiliza tareas específicas en flujos de trabajo típicos de extracción de valores de la Observación de la Tierra (EO), como enmascaramiento de nubes, extracción de características, clasificación, etc. Permite que la extracción de información valiosa de imágenes satelitales sea tan fácil como definir una secuencia de operaciones dependientes entre ellas. Por ejemplo la Figura 28 ilustra una cadena de procesamiento que mapea el agua en imágenes de satélite en la región de interés (ROI) especificada por el usuario.

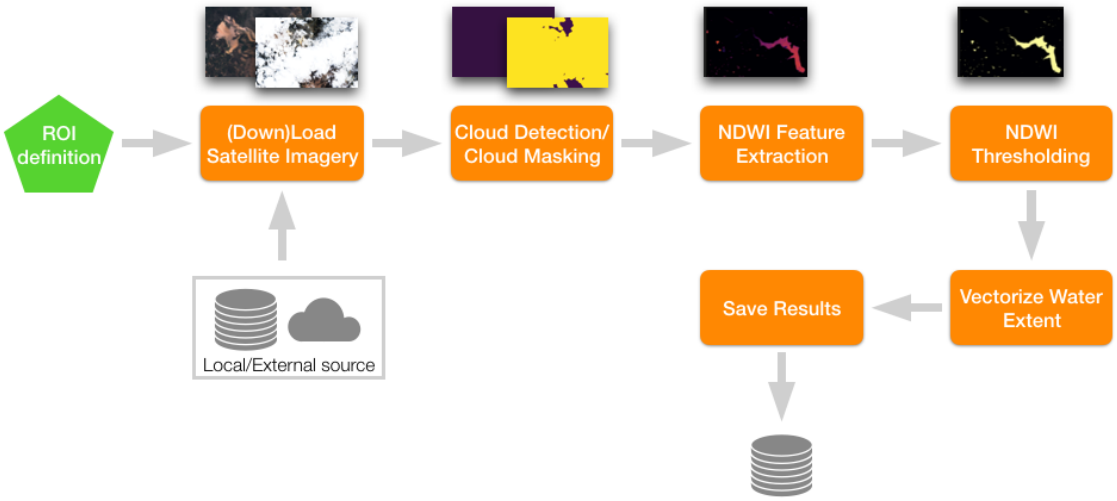


Figura 28 Ejemplo de flujo de tareas de EOLearn

Esta librería actúa como un puente entre la observación de la Tierra (ámbito de la teledetección) y el ecosistema de Python para la ciencia de datos y el aprendizaje automático. Está implementada en Python y emplea matrices NumPy para almacenar y manejar datos de sensores remotos. Sus objetivos son:

1. Otorgar facilidades a los no expertos en el campo de la teledetección para su iniciación
2. Proveer de las herramientas más avanzadas existentes en el ecosistema de Python para visión por computador, el aprendizaje automático y el *deep learning* a los expertos en éste ámbito.

A modo de resumen, por su estructura modular, eo-learn está dividido en subpaquetes según sus funcionalidades:

- **eo-learn-core**: Subpaquete principal. Implementa bloques de construcción básicos (EOPatch, EOTask y EOWorkflow) y funcionalidades de uso común.
- **eo-learn-coregistration**: Herramientas y EOTasks para el co-registro de imágenes
- **eo-learn-features**: Utilidades para extraer propiedades de datos y la manipulación de características.
- **eo-learn-geometry**: Utilizado para la transformación geométrica y la conversión entre datos vectoriales y ráster.
- **eo-learn-io**: Se ocupa de la etrada/salida. Es decir, encargado de obtener de datos de los servicios de Sentinel Hub o de guardar y cargar datos de forma local.
- **eo-learn-mask**: Utilizado para el cálculo de máscaras de nube y otros enmascaramientos de datos.
- **eo-learn-ml-tools**: Herramientas que pueden utilizarse antes o después del proceso de aprendizaje automático.

Su diseño sigue el paradigma de programación de flujo de procesamiento, para ello consta de los siguientes tipos de construcción:

- **EOPatch**: objeto de datos común para datos de observación de la tierra (espacio-temporales y no EO espacio-temporales) y sus derivados.

- **EOTask**: Una acción única y bien definida que se realiza en EOPatch(s) existentes. Cada EOTask toma un EOPatch como entrada y devuelve un EOPatch modificado.
- **EOWorkflow**: Colección de EOTask con dependencias definidas entre ellas. Representa una cadena de procesamiento en forma de grafo dirigido acíclico de EOTasks.
- **EOExecutor**: Gestiona las ejecuciones de un EOWorkflow en paralelo con diferentes datos de entrada.

3 SIWUAC: Sentinel Images With Urban Atlas Classification

En este capítulo se describe el funcionamiento de la herramienta he hemos desarrollado durante el proyecto y hemos denominado SIWUAC. Esta denominación pretende resumir el producto final que la herramienta permite generar: imágenes de las misiones Sentinel junto a máscaras de clasificación generadas a partir de la información que proporciona Urban Atlas. La Sección 3.1 argumenta las necesidades de emplear una herramienta de este tipo al trabajar con los datos del programa *Copernicus*. A continuación, en la Sección 3.2 se resumen las tareas encomendadas a cada una de las EOTask que forman el EOWorkflow que implementa SIWUAC. Para finalizar este capítulo, la Sección 3.3 describe el conjunto de imágenes que hemos empleado para cumplir con el segundo objetivo parcial del proyecto, realizar un estudio sobre redes neuronales de segmentación semántica aplicadas a imágenes satelitales.

3.1 Propuesta

Hasta hace unos pocos años, la disponibilidad de imágenes satelitales únicamente era asumible por grandes empresas o instituciones gubernamentales. Esto era debido a la infraestructura que requiere tanto su obtención como su almacenamiento, pero esto era antes de las misiones Sentinel y de los servicios de datos de *Copernicus*. El programa, ha hecho posible la aparición de multitud de proyectos basados en análisis de productos satelitales, por ejemplo el proyecto PyrenEOS (liderado por el Gobierno de Navarra) el cual que tiene como objetivo generar servicios basados en el tratamiento y explotación de las imágenes de los satélites Sentinel para dar soporte a aplicaciones relacionadas con la agricultura, la gestión forestal y los recursos hídricos, incluyendo previsiones de inundaciones.

Es un hecho que se ha estandarizado el uso de los conjuntos de imágenes de COCO y PASCAL-VOC en la gran mayoría de trabajos en segmentación semántica. Los resultados de estos se evalúan haciendo uso de dichas imágenes ya que cuentan con máscaras de clasificación y son escenas de la vida cotidiana. Por este motivo, las redes presentadas no son válidas para imágenes satelitales ya que sus propiedades son diferentes y las clases que se buscan también. Otro hecho relevante ha sido el gran impulso que ha tomado el desarrollo de los vehículos autónomos. Gracias a ello, han aparecido trabajos que emplean de imágenes de escenas urbanas tomadas desde vehículos (por ejemplo el *dataset* Cityscapes [32]), pero la falta de conjuntos de imágenes satelitales anotadas con máscaras de clasificación es evidente. Este motivo, junto a al acceso libre a imágenes Sentinel que proporciona *Copernicus*, y sumado a la inquietud de estudiar el comportamiento de las CNN aplicadas a imágenes satelitales, han sido los principales factores que nos han llevado a la creación de SIWUAC.

Actualmente existen multitud de aplicaciones GIS permiten manejar datos ráster (imágenes Sentinel) y vectoriales (Urban Atlas). Por este motivo, si necesitamos disponer de imágenes satelitales anotadas a nivel de pixel como las de la Figura 1 que nos sirvan de ejemplos de entrenamiento en CNN, estas herramientas permiten generarlas de forma cómoda e intuitiva. Sin embargo, hay dos grandes inconvenientes cuando utilizamos este tipo de herramientas para este objetivo:

- **El proceso no es automatizable**: La interfaz gráfica de las herramientas GIS permite que sea "sencillo" realizarlo, pero requiere un tiempo excesivo si queremos extenderlo a una gran cantidad de áreas o ciudades.
- **No todas las herramientas son software libre**: A pesar de la existencia herramientas *open source* como QGIS, también existen otras más sofisticadas que pertenecen a compañías privadas y suponen un alto coste en licencias, por ejemplo ArcGIS de ESRI.

Nosotros hemos desarrollado una herramienta que automatiza las tareas necesarias de pre-procesado de los productos Sentinel hasta generar conjuntos de imágenes de áreas urbanas geolocalizadas y anotadas a nivel de pixel con máscaras de clasificación. Estas tareas se han implementado en forma de EOTasks y como se aprecia en la Figura 29, abarcan desde la descarga en bruto de las imágenes desde *Copernicus Open Access Hub* hasta la creación de los mosaicos de ciudades. SIWUAC dota a los investigadores de una forma cómoda y rápida de producir *datasets* para trabajar con ellos. Además, por su filosofía de *flujo de tareas* (EOWorkflow) permite adaptarla a otros datos que sigan el formato de a Urban Atlas como por ejemplo los *shapefiles* de Open Street Map.

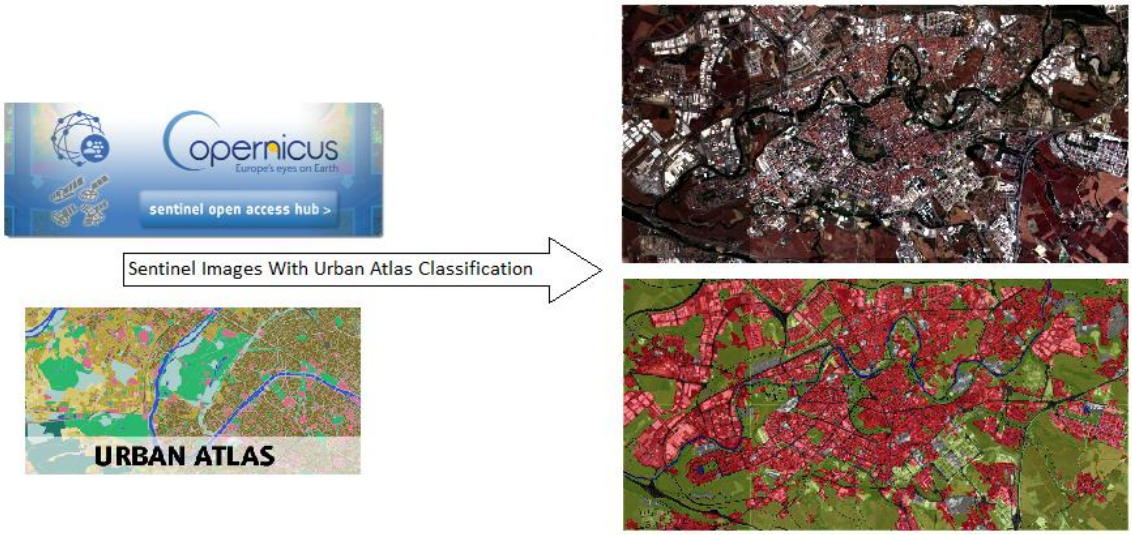


Figura 29 Entrada y salida general de SIWUAC

A continuación se detallan cada una de las tareas implementadas en EOTask que forman el EOWorkflow de SIWUAC. Para cada EOTask se especifican los parámetros necesarios que permiten configurar la ejecución del EOWorkflow y el procesamiento que se realiza en cada uno de ellos.

3.2 Implementación

Durante el desarrollo de SIWUAC hemos creado un flujo de tareas (EOWorkflow de EOTasks según la filosofía de EOLearn) que admite ejecutarse en paralelo, empleando el EOExecutor. Cada instancia de EOWorkflow es un grafo dirigido acíclico de tareas que se ejecutan creando un EOPatch que se le pasa como entrada a la primera o primeras EOTasks del EOWorkflow. El EOPatch es una estructura de datos preparado para almacenar datos multitemporales obtenidos por teledetección de un solo parche (área) de la superficie de la Tierra. Esta zona está típicamente definida por un cuadro delimitador en un sistema de referencia de coordenadas específico. No hay limitación en la cantidad de datos, o el tipo de datos que se pueden almacenar en un EOPatch, pero normalmente, toda la información se almacena internamente en forma de matrices Numpy y Geodataframes. Podemos verlo como un diccionario de Python con las siguientes claves: DATA, MASK, SCALAR, LABEL, VECTOR, DATA_TIMELESS, MASK_TIMELESS, SCALAR_TIMELESS, LABEL_TIMELESS, VECTOR_TIMELESS, META_INFO, BBOX y TIMESTAMP. Cada uno de ellos pensado para diferentes fines pero en este proyecto solo hemos empleado los siguientes:

- DATA_TIMELESS: Datos independientes del tiempo y dependientes de la posición. Matrices Numpy $n \times m \times d$ de tipo Float. Los emplearemos para almacenar las bandas de las imágenes satelitales.
- VECTOR_TIMELESS: Datos independientes del tiempo y dependientes de la posición. Colección de geometrías almacenadas en Geodataframes. Almacenará el *shapefile* del Urban Atlas de la ciudad correspondiente.
- MASK_TIMELESS: Datos independientes del tiempo y dependientes de la posición. Matrices Numpy $n \times m \times d$ de tipo Float. Los emplearemos para almacenar la máscara de clasificación de la escena representada en DATA_TIMELESS.
- BBOX: Cuadro delimitador del parche que es una instancia de sentinelhub.BBox y contiene información sobre el sistema de coordenadas.
- META_INFO: Un diccionario de información adicional de metadatos, por ejemplo la resolución de la escena o rutas de almacenamiento.

En EOLearn, para construir el grafo dirigido acíclico que forma el EOWorkflow se siguen los siguientes pasos:

1. **Instanciar todas las EOTask**: Con los *parámetros de inicialización* se define el comportamiento de las tareas en todas las ejecuciones. Por ejemplo, si queremos generar los mosaicos de varias ciudades con el mismo porcentaje de cobertura nubosa en las imágenes Sentinel de cada ciudad, este valor sería un *parámetro de inicialización* de la EOTask que se encarga de descargar las imágenes.
2. **Construir el EOWorkflow**: Se definen las dependencias entre todos los nodos (EOTasks) instanciados anteriormente. Por ejemplo, no podemos crear las imágenes GeoTIFF si no hemos descargado los productos Sentinel de SciHub.

A la hora de ejecutar el grafo en paralelo empleamos el EOExecutor, el cual permite gestionar cada una de las ejecuciones mediante los *parámetros de ejecución*. Además, al final genera un informe que contiene el resumen de la ejecución completa. El ejemplo más claro de *parámetro*

de ejecución es el nombre de la ciudad ya que cada ejecución será encargada de generar el mosaico de una ciudad. La *lista de argumentos de ejecución,* contiene diccionarios de Python que definen los *parámetros de ejecución* de las EOTasks y se le pasan al EOExecutor como parámetro. En definitiva, una ciudad se traduce en un EOPatch que "viaja" por un conjunto de tareas las cuales completan sus diferentes apartados para lograr un objetivo.

El diagrama de la Figura 30 muestra el grafo de EOTasks que implementa SIWUAC para generar el mosaico de imágenes etiquetadas. El proceso comienza con la instanciación del EOPatch y termina con la ejecución de la última EOTask (la que genera el mosaico).

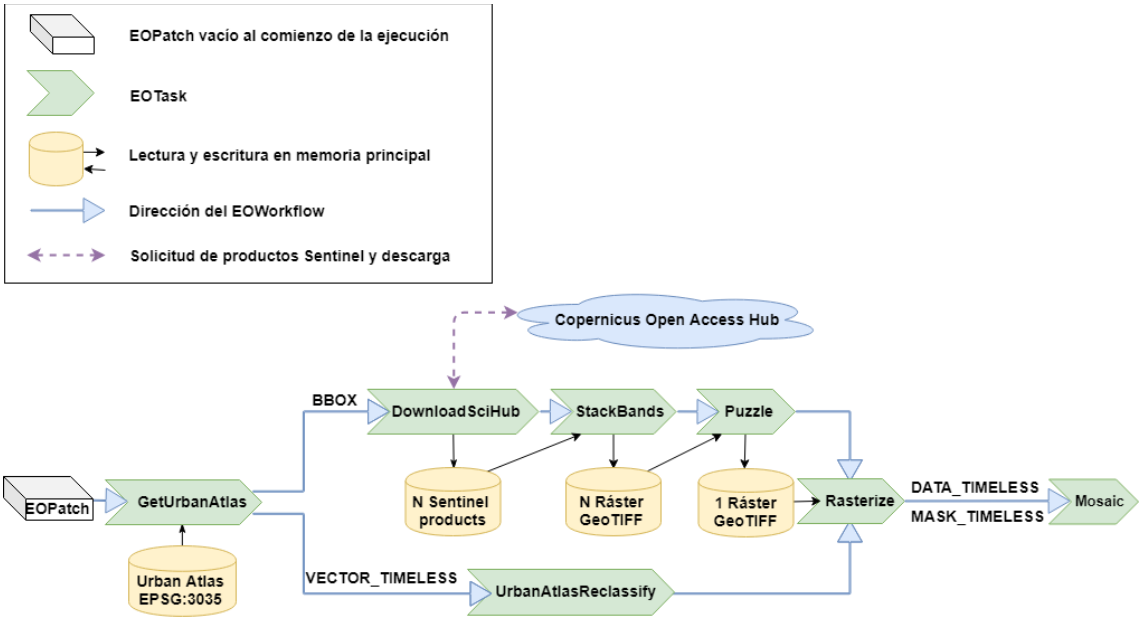


Figura 30 Topología del EOWorkflow de SIWUAC

A continuación mostramos detalles de implementación y funcionamiento de cada una de las EOTask. Todas ellas se apoyan en META_INFO para almacenar información auxiliar como las rutas a los ficheros necesarios o el nombre de la ciudad para facilitar la búsqueda y acceso a los ficheros que utilizan las diferentes EOTasks.

1. GetUrbanAtlas

- *Parámetros de inicialización*: Directorio donde se encuentra almacenado Urban Atlas.
- *Parámetros de ejecución*: EOPatch vacío que inicia el flujo y nombre de la ciudad.
- *Salida*: EOPatch que contiene en VECTOR_TIMELESS la información de Urban Atlas (EPSG:3035) de una ciudad y en BBOX las correspondientes coordenadas del cuadro delimitador.
- *Funcionamiento*: Busca el *shapefile* de la ciudad correspondiente y lo almacena en VECTOR_TIMELESS. A partir de este, se extraen los límites totales del área espacial que cubre y se almacena en BBOX.
  A continuación, en la Figura 31 mostramos la BBOX que se extrae de la información de Urban Atlas de Alicante para realizar la petición a SciHub en la siguiente EOTask.

La Figura 31 y las que siguen en esta sección están generadas con QGIS y muestran gráficamente el proceso que realiza SIWUAC. Para ello nos hemos basado en la ciudad de Alicante.

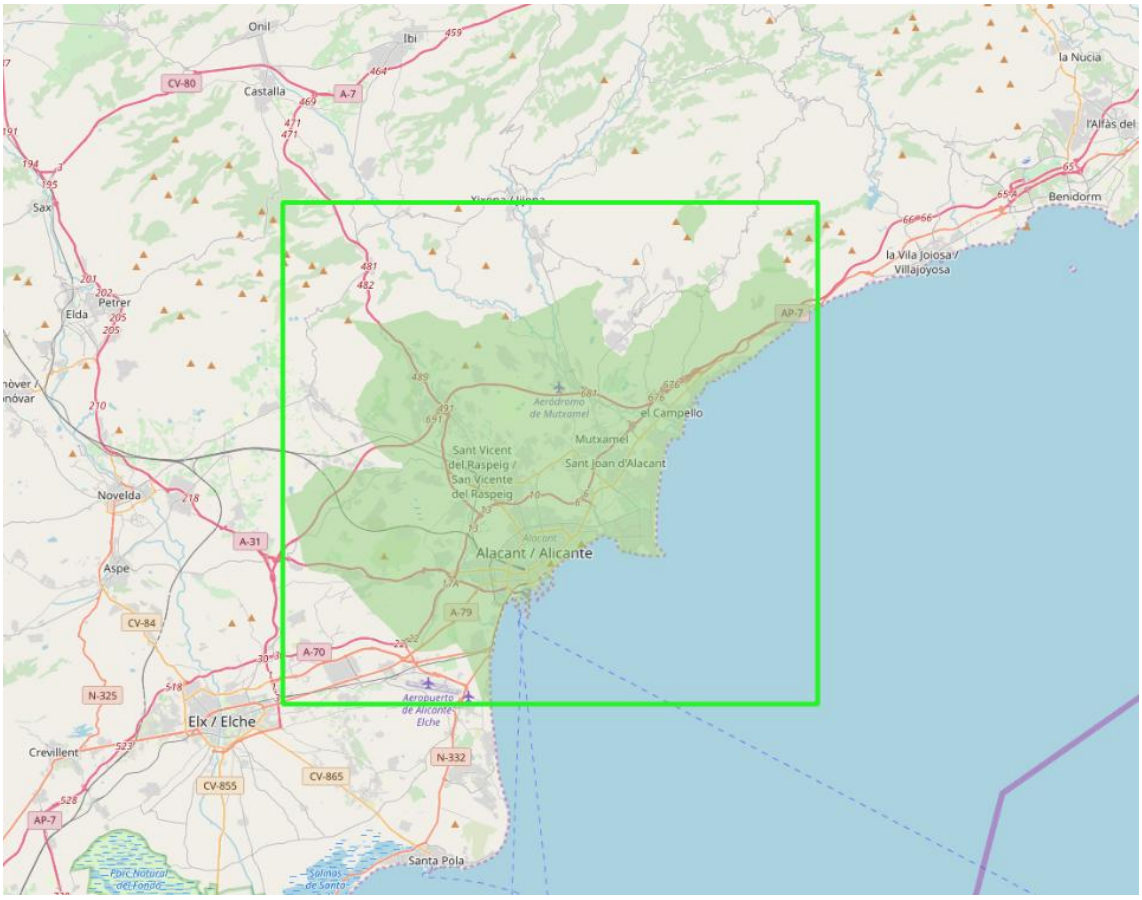


Figura 31 En verde la superficie que cubre Urban Atlas en Alicante, y la BBOX extraída

2. DownloadSciHub

1. *Parámetros de inicialización*: Intervalo de tiempo, plataforma (Sentinel-1, Sentinel-2, Sentinel-3), tipo de producto (nivel L1C, L2A,…), dirección de la órbita, modo de operación, cobertura de nubes, orbitas excluidas y modo de selección, empleados para solicitar productos Sentinel y filtrarlos. También tiene otros para decidir si descomprimir y eliminar los archivos descargados y para indicar la carpeta de descarga.
2. *Parámetros de ejecución*: Credenciales de *Copernicus Open Access Hub*
3. *Salida*: EOPatch con meta-información de las descargas y N productos Sentinel (SAFE) almacenados en disco.
4. *Funcionamiento*: Realiza una petición de productos al servidor de SciHub según la BBOX (WGS84), la fecha, la cobertura de nubes, la dirección de órbita y el tipo (nivel) de producto. La consulta devuelve una lista de productos que intersecan con la BBOX los cuales pueden ser de la misma zona en diferentes fechas, diferente cobertura nubosa, orbita,... Existen cuatro formas de filtrar esta lista de productos según las necesidades del usuario:
    0) Más próximo a la fecha de Urban Atlas y menor nubosidad
    1) Menor nubosidad
    2) Más reciente
    3) Todas
    4) Búsqueda adaptada: Debido a que necesitamos las imágenes más cercanas a los datos de Urban Atlas, se eligen productos de la siguiente manera: Las imágenes que formen el puzle deberán ser las más cercanas a la fecha de Urban Atlas pero penalizando la cobertura de nubes posible y la mínima distancia temporal entre ellas

Al ejecutarse, una vez que la API de Sentinel Hub nos devuelve los productos seleccionados, comienza la descarga de estos. SciHub admite dos descargas simultáneas por usuario, este hecho nos obliga a tener que crear tantas cuentas en SciHub como ciudades nos interese paralelizar. Los ficheros .zip obtenidos tras la descarga se descomprimen y se eliminan si así se ha indicado en los *parámetros de inicialización*.

En la Figura 32 mostramos las bandas RGB de los dos productos Sentinel que obtenemos con SIWUAC al hacer la petición a SciHub con la BBOX que define el Urban Atlas de Alicante.

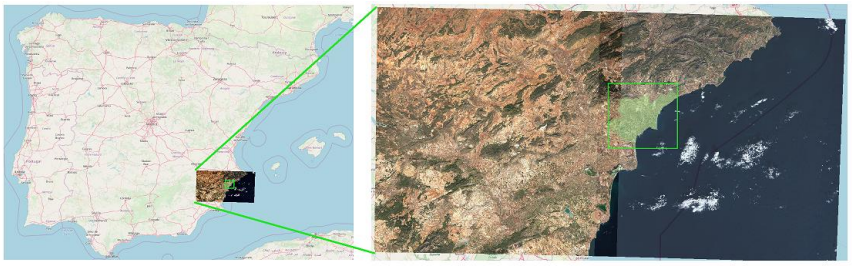


Figura 32 Productos Sentinel para cubrir la zona delimitada por el Urban Atlas de Alicante

Las imágenes están con el 75% de transparencia para apreciar el solapamiento espacial que tienen los productos Sentinel. En la zona central de la imagen ampliada se muestra como la superficie de Urban Atlas interseca con amas imágenes.

3. **StackBands**

- *Parámetros de inicialización*: Bandas Sentinel a emplear, en nuestro caso siempre son la 4, 3, 2 y 8 correspondientes a las componentes R, G, B e Infrarrojo, aunque este parámetro es configurable. También el nombre del entorno Python donde queramos ejecutar las llamadas a la función gdal_merge.py de los enlaces de GDAL para Python ya que puede ser diferente al entorno donde se ejecute el EOWorkflow. También se le indica la ruta de almacenamiento de los ficheros generados por gdal_merge.py.
- *Salida*: Tantos archivos GeoTIFF como productos Sentinel se hayan descargado en la tarea anterior.
- *Funcionamiento*: Construye una imagen satelital en formato GeoTIFF con las bandas indicadas por cada producto SAFE. Las bandas son archivos JP2 localizados dentro del archivo SAFE (ver Sección 2.3.3). Para ello construye el comando que ejecuta la función gdal_merge.py y lo ejecuta tantas veces como productos tengamos en procesos diferentes al del propio EOWorkflow, es decir, se trata cada imagen Sentinel obtenida en la tarea anterior por separado. Por este motivo, esta EOTask espera a que terminen estos procesos para continuar.

4. **Puzzle**

- *Parámetros de inicialización:* Un valor opcional para truncar los valores de las bandas hasta un límite, la ruta de almacenamiento del fichero "puzzle" y debido a que también emplea gdal_merge.py se le debe indicar el entorno de ejecución.
- *Salida*: Un archivo GeoTIFF resultante de combinar varios de diferentes zonas.
- *Funcionamiento*: Del mismo modo que la tarea anterior, construye el comando para ejecutar gdal_merge.py pero en esta EOTask, solo se ejecuta una vez para unir varios archivos GeoTIFF en uno único que cubra de sobra la BBOX del Urban Atlas de la ciudad en cuestión. La Figura 33 muestra el puzle creado con ambos productos.

Figura 33 Imagen GeoTiff generada a partir de las dos imágenes Sentinel descargadas

La figura anterior es una imagen con bandas R, G, B e Infrarrojo pero para la visualización de las figuras mostramos únicamente las bandas RGB.

5. UrbanAtlasReclassify

- *Parámetros de inicialización*: El principal argumento es un archivo JSON que define el etiquetado que queremos generar en las imágenes del mosaico final. En él se definen las agrupaciones de las clases originales de Urban Atlas que forman las nuevas clases. Por ejemplo las clases 40000 y 50000 de Urban Atlas correspondientes a Humedales y Agua pasan a ser la clase Agua. Además, se le indica el nombre de la nueva clasificación y una ruta opcional para almacenar el *shapefile* de la clasificación configurada.
- *Salida:* EOPatch con la información de Urban Atlas reclasificada.
- *Funcionamiento*: Las clases de Urban Atlas están codificadas en la columna CODE2012 del Geodataframe almacenado en VECTOR TIMELESS. Para cada nueva clase definida en el JSON, se extraen los polígonos correspondientes a la agregación de clases originales y se va creando un nuevo Geodataframe con nuestro etiquetado. Este sobre escribe el original almacenado en VECTOR_TIMELESS para ser rasterizado en la próxima EOTask.
  La Figura 34 muestra el *shapefile* de Alicante que proporciona Urban Atlas con la leyenda de colores que incluye el producto.

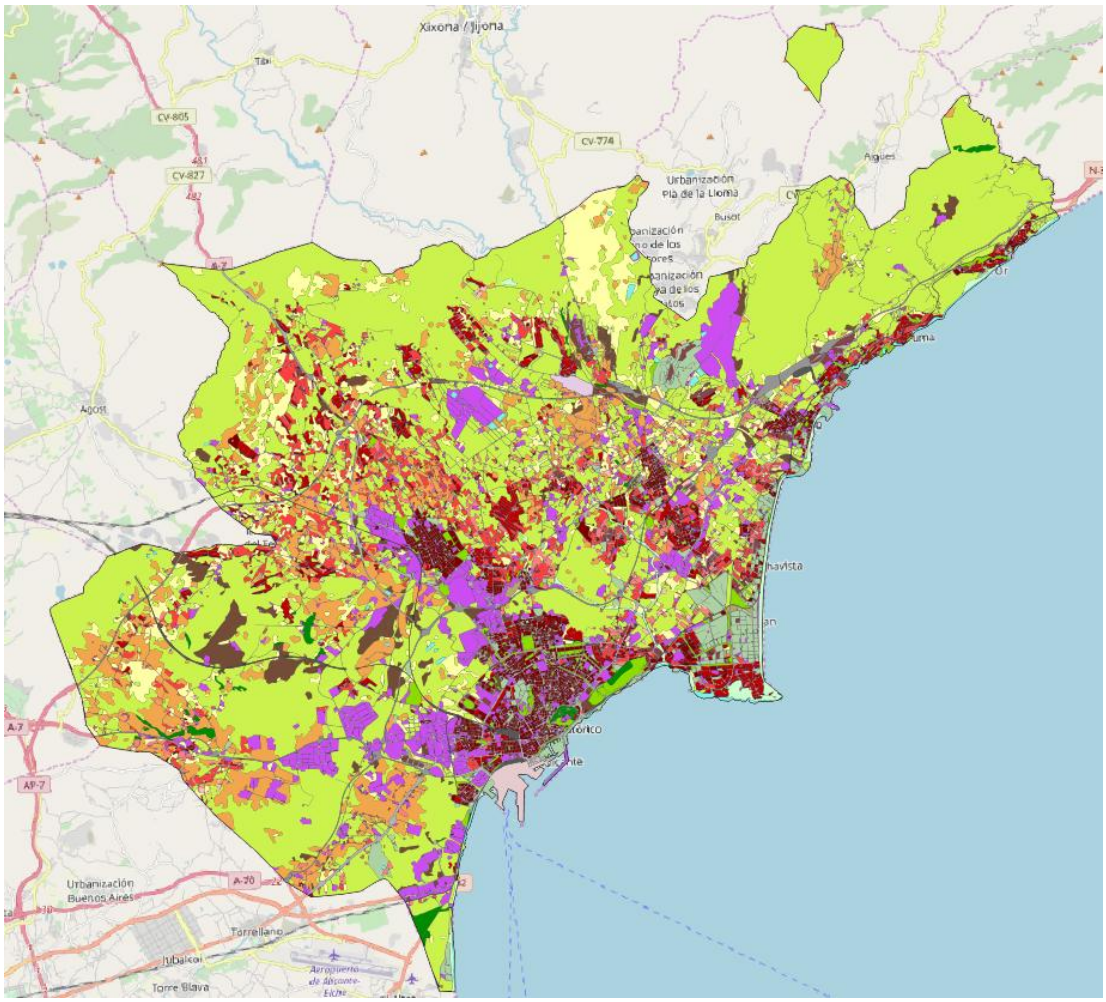


Figura 34 Urban Atlas de Alicante

La Tabla 3 muestra la codificación y el nombre de las clases que cubre Urban Atlas 2012. La agregación y denominación de clases que hemos definido para nuestra reclasificación se muestra en la Tabla 4.

Leyenda de Urban Atlas	
CODE2012	**ITEM2012**
11100	Tejido urbano continuo (S.L .:> 80%)
11210	Tejido urbano denso discontinuo (S.L .: 50% - 80%)
11220	Tejido urbano discontinuo de densidad media (S.L .: 30% - 50%)
11230	Tejido urbano discontinuo de baja densidad (S.L .: 10% - 30%)
11240	Tejido urbano discontinuo de muy baja densidad (S.L .: <10%)
11300	Estructuras aisladas
12100	Unidades industriales, comerciales, públicas, militares y privadas
12210	Vías de tránsito rápido y tierra asociada
12220	Otras carreteras y terrenos asociados
12230	Ferrocarriles y terrenos asociados
12300	Zonas portuarias
12400	Aeropuertos
13100	Extracción de minerales y vertedero
13300	Sitios de construcción
13400	Terreno sin uso actual
14100	Áreas urbanas verdes
14200	Instalaciones deportivas y de ocio
21000	Tierra arable (cultivos anuales)
22000	Cultivos permanentes
23000	Pastos
24000	Patrones de cultivo complejos y mixtos
25000	Huertos al margen de las clases urbanas.
31000	Bosques
32000	Vegetación herbácea
33000	Espacios abiertos con poca o ninguna vegetación
40000	Humedales
50000	Agua
91000	Sin datos (Nubes y sombras)
92000	Sin datos (Imágenes que faltan)

Tabla 3 Codificación de clases de Urban Atlas

Clasificación propia		
AGRUPACION	**ITEM**	**CODE**
11100,11210,11220,11230,11240,11300,12100,12300,12400,13100, 13300,14100,14200	URBANO	1
12210,12230,12220	VIAS	2
13400,21000,22000,23000,24000,25000,31000,32000,33000	SUELO	3
40000,50000	AGUA	4
91000,92000	NO CLASE	0

Tabla 4 Clasificación empleada en los experimentos del proyecto

Esta EOTask permite crear cualquier agregación de clases de Urban Atlas, nosotros para las pruebas realizadas en este proyecto hemos fijado la que se muestra en la tabla anterior. Al ser los primeros experimentos con este tipo de máscaras comenzamos con pocas clases y fácilmente diferenciables por el ojo humano en imágenes Sentinel como son las zonas urbanas, los viales (autopistas, vías de tren o carreteras principales) agua y el resto de suelo en general, pero también hay que tener en cuenta que hay zonas no clasificadas por Urban Atlas (CODE 0 en la Tabla 4). Por otro lado, el Geodataframe que mantiene en memoria tal clasificación es posible almacenarlo en formato *shapefile* si se le indica ruta del archivo de salida en los parámetros de inicialización de la tarea. En la Figura 35 se muestra el *shapefile* generado con la agregación de clases definida en la Tabla 4.

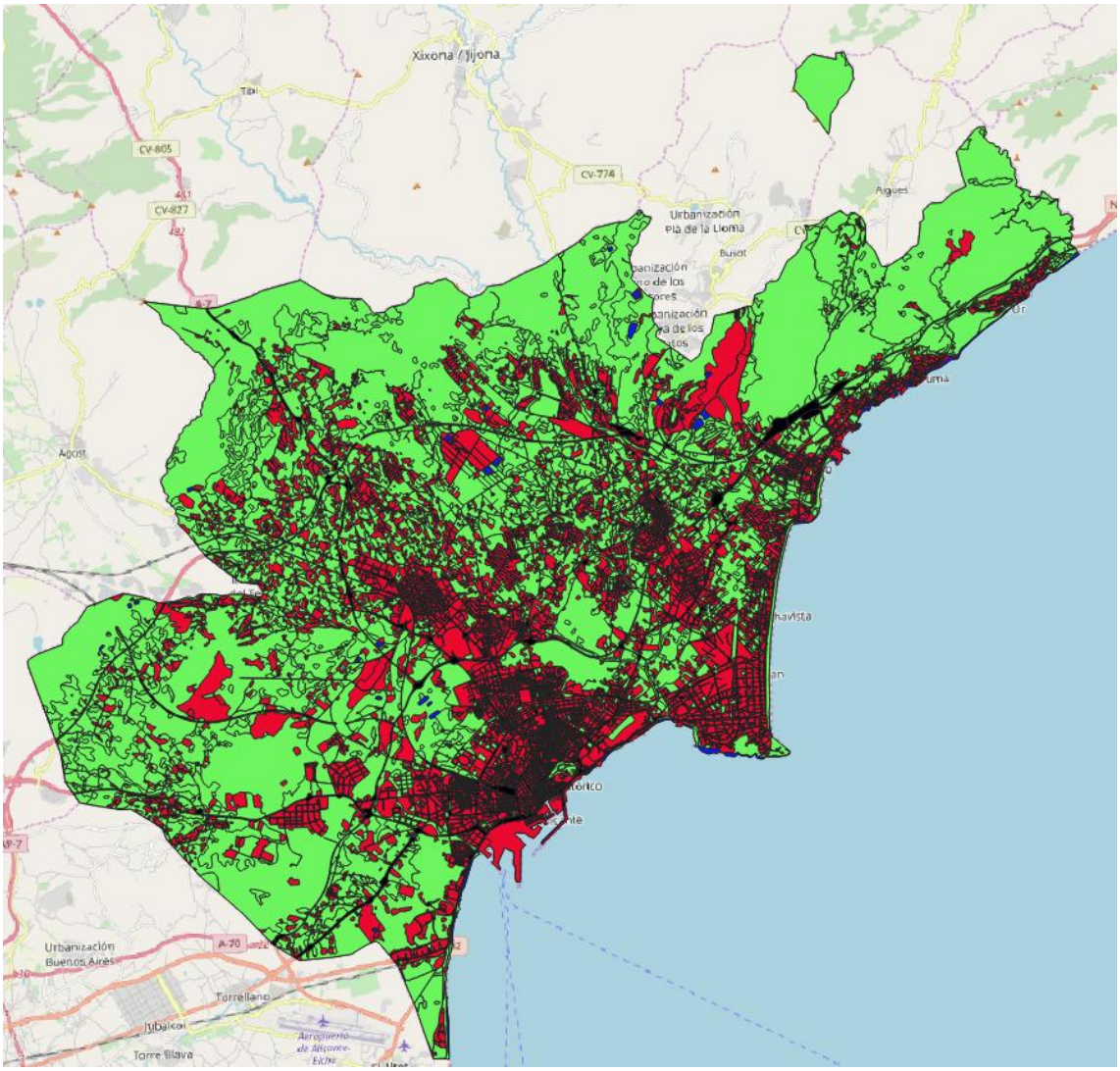


Figura 35 Shapefile creado mediante la agrupación de clases de Urban Atlas

6. Rasterize

- *Parámetros de inicialización*: La ruta de salida para la rasterización y un factor de máscara (FM) que sirve para aumentar la resolución de las máscaras de clasificación, ambos opcionales.
- *Parámetros de ejecución*: Esta EOTask tiene la característica que recibe dos EOPatchs ya que une los dos caminos del EOWorkflow en uno. Como se aprecia en la Figura 30, recibe el EOPatch que devuelve la ejecución de la EOTask Puzzle y también el que devuelve la ejecución de UrbanAtlasReclassify.
- *Salida*: EOPatch que contiene en DATA_TIMELESS (Figura 36 a) una matriz que almacena las 4 bandas Sentinel de la superficie definida por la BBOX y en MASK_TIMELESS una matriz de una banda que contiene la máscara de clases (Figura 36 b), es decir la rasterización.
- *Funcionamiento*: Rasterizar es un proceso que consiste en convertir datos vectoriales en datos ráster, en nuestro caso concreto, convertir el Geodataframe que contiene Urban Atlas reclasificado en una máscara de clases bidimensional del mismo tamaño que el ráster que contiene los datos satelitales. Para ello, con la BBOX, se recorta el puzle que hemos generado con las funciones de GDAL para Python y se almacena en forma de matriz Numpy en DATA_TIMELESS. Seguidamente se crea una matriz bidimensional del mismo tamaño que el recorte completa con el valor 255 que servirá como plantilla para la rasterización. Sobre esta matriz, se rasteriza el Geodataframe de VECTOR_TIMELESS mediante la función rasterize de la librería Rasterio. Por último, almacenamos en MASK_TIMELESS la matriz donde hemos rasterizado el Geodataframe. Debido a que los datos de Urban Atlas tienen más resolución que las imágenes Sentinel, podemos crear máscaras de con más detalle ampliando la plantilla del rasterizado con el factor pixel en la inicialización de la EOTask.

(a) DATA_TIMELESS (b) MASK_TIMELESS c) Visualización con QGIS

Figura 36 Resultado de la rasterización

A continuación mostramos la diferencia entre rasterizar sobre una plantilla de la misma resolución que Sentinel (10m/px) y rasterizar sobre máscaras aumentadas con diferentes factores de FM. En la Figura 37 se muestra el *shapefile* de VECTOR_TIMELESS (ampliado para ver el detalle) y la imagen Sentinel que se emplea como plantilla para el rasterizado.

(a) VECTOR_TIMELESS (b) DATA_TIMELESS

Figura 37 *Shapefile* (a) e imagen Sentinel (b)

El *shapefile* tiene gran detalle, las calles y carreteras se definen perfectamente, incluso aquellas que son difíciles de apreciar a simple vista en las zonas muy aglomeradas de la imagen Sentinel. La Figura 38 muestra la máscaras generadas mediante la rasterización con FM=1 y FM=2.

(a) MASK_TIMELESS (FM=1) 10m/px (a) MASK_TIMELESS (FM=2) 5m/px

Figura 38 Máscaras de clasificación con diferentes valores de factor de máscara

Como vemos, este factor permite al usuario crear máscaras con más resolución, lo que permite definir mucho mejorar las formas de carreteras y bordes. El problema es que si entrenando redes con estas imágenes, la entrada no tiene las mismas dimensiones que la salida. Esto obligaría a la red a inferir las clases de varios píxeles en la predicción a partir de un solo pixel en la entrada.

7. CreateMosaic

- *Parámetros de inicialización*: El tamaño de ventana con la que queremos recortar el rasterizado, mínimo grado de cobertura de clases (MCC), lista de solapamientos y ruta de salida de los ficheros.
- *Salida*: Directorio que contiene las imágenes y máscaras correspondientes para formar el mosaico de una ciudad.
- *Funcionamiento*: Comenzamos generando las máscaras de clases que emplearemos en el entrenamiento de las CNN. Para ello, se toma la matriz de MASK_TIMELESS y se van creando las ventanas del tamaño indicado de la siguiente forma. Para cada grado de solapamiento, nos desplazamos por la matriz obteniendo ventanas del tamaño indicado teniendo en cuenta el grado de solapamiento. Cada ventana se guarda en disco solo si el número de píxeles etiquetados que contiene (con valor diferente de 255) supera el grado de cobertura de clases en número de píxeles. Durante el recorrido de la máscara, también se almacenan los índices de las ventanas que se han generado.

  El siguiente paso es generar las correspondientes imágenes satelitales a partir de la matriz almacenada en DATA_TIMELESS. Haciendo uso de los índices almacenados se generan tantas imágenes como máscaras se han creado en el paso anterior.

  Los parámetros de esta tarea junto al factor de pixel en el rasterizado, determinan la topología del conjunto de imágenes generado. Además, todos los archivos generados están geolocalizados por lo que el mosaico compuesto por las imágenes de cada ciudad podemos visualizarlo con QGIS. La Figura 39 muestra el mosaico que forman 9 imágenes de 1024 filas y 1024 columnas, con solapamiento del 25% de píxeles entre ellas y mínima cobertura de clases al 0%.

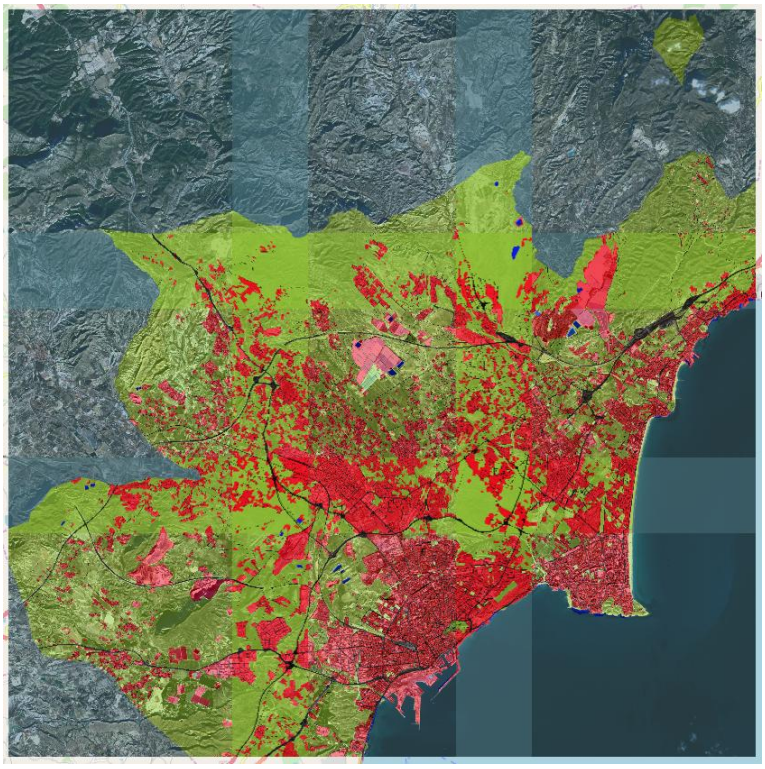


Figura 39 Mosaico de Alicante formado por 9 imágenes 1024x1024x4, con el 25% de píxeles solapados

3.3 Conjunto de imágenes sintéticas

SIWAC ofrece ciertos parámetros de los cuales dependerán las características del conjunto de imágenes que generamos para cada ciudad. Por ejemplo la cobertura de clases, el tamaño de la imagen y los grados de solapamientos son factores que influyen en el número de imágenes generadas para cada ciudad.

Como hemos dicho en el Capítulo 2, Urban Atlas ofrece información acerca del uso y cobertura terrestre para 800 áreas urbanas funcionales (FUA) de 39 países europeos. Nosotros nos hemos ceñido principalmente en nuestro país y hemos empleado imágenes Sentinel-2 de las 10 ciudades que mostramos en la Tabla 5 para nuestros experimentos. En total hemos generado 634 imágenes organizadas en ciudades con sus correspondientes máscaras obtenidas con la rasterización del Urban Atlas aplicando la agrupación de clases representada en la Tabla 4.

Como conjunto de imágenes básico, hemos establecido el tamaño de la imagen a 1024 pixeles por cada fila y columna (tamaño de ventana), 0% y 25% como valores de grado de solapamiento. Es decir, para cada ciudad generamos dos mosaicos: uno con 0 píxeles solapados y otro con 256. Como resultado, SIWUAC ha creado 620 imágenes correspondientes a dos mosaicos de 10 ciudades. En la Tabla 5 Número de imágenes por cada ciudad mostramos el número de imágenes que componen ambos mosaicos de cada ciudad.

Ciudad	Número de imágenes		Total
	Grado de solapamiento		
	0%	25%	
Alicante	4	9Figura 39	13
Cartagena	8	15	23
Córdoba	30	42	72
Jaén	12	20	32
Lorca	30	42	72
Madrid	110	132	242
Murcia	24	35	59
Pamplona	32	45	77
Reus	1	2	3
Santander	12	15	27
Total	**263**	**357**	**620**

Tabla 5 Número de imágenes por cada ciudad

Los datos mostrados en la Tabla 5 corresponden a un *dataset* en el que no se tienen en cuenta el número de píxeles etiquetados en cada imagen, es decir el mínimo de cobertura de clases establecido al 0%. Este parámetro determina la forma del mosaico generado de la ciudad correspondiente, además de permitir reducir el número de imágenes manteniendo las que más píxeles etiquetados tengan. Para mostrar esto gráficamente en la Figura 40 hemos generado los mosaicos de Alicante con imágenes de tamaño 256, sin solapamiento y con diferentes grados de cobertura de clases.

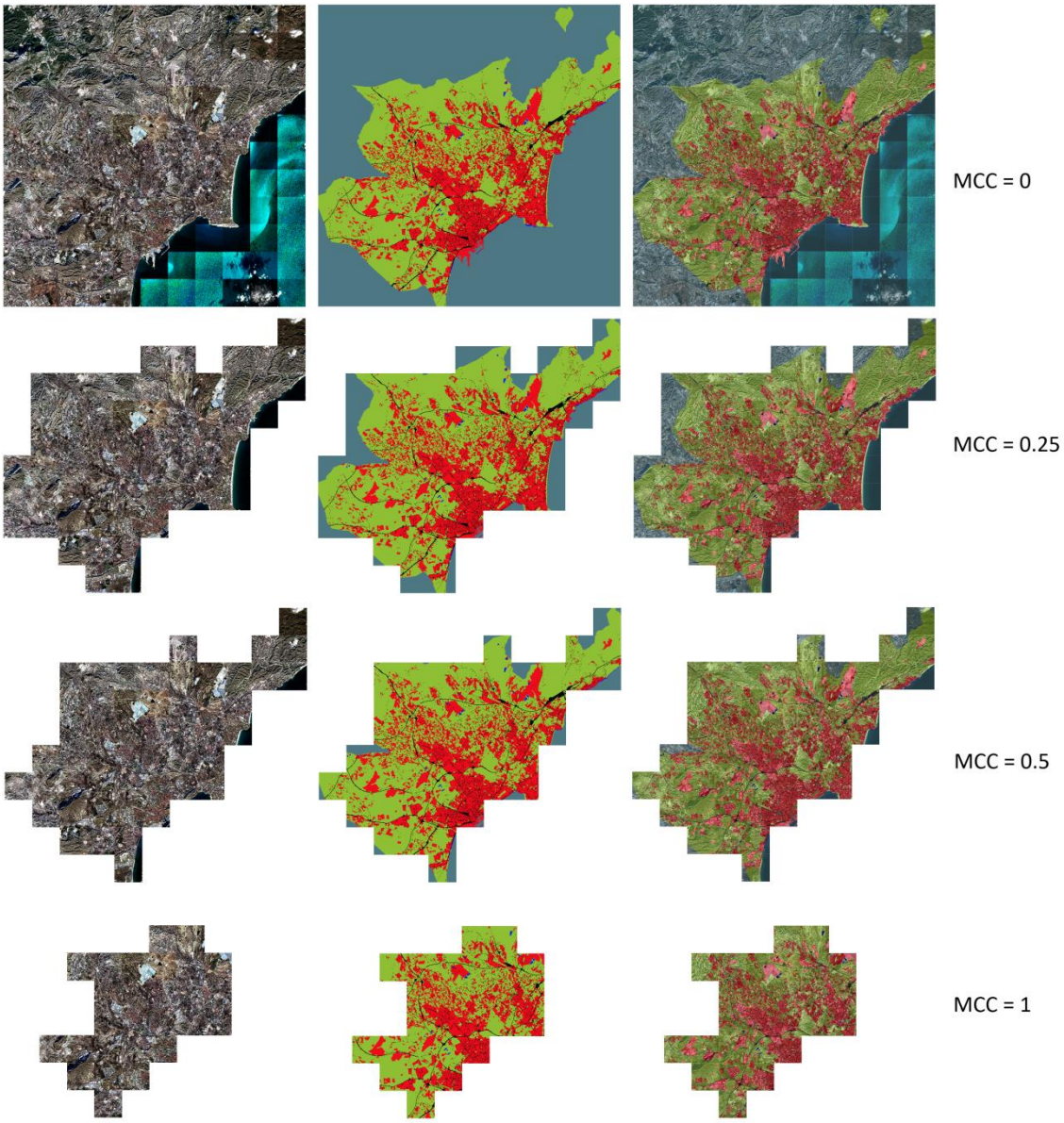


Figura 40 Mosaico de Alicante con diferentes grados de cobertura de clases (MCC)

4 SegNet

La red neuronal para segmentación semántica SegNet [7], fue desarrollada en la Universidad de Cambridge por Alex Kendall, Vijay Badrinarayanan, y Roberto Cipolla. Su arquitectura, mostrada en la Figura 41 sigue la filosofía *encoder-decoder*. Normalmente, para el *encoder* se emplea una red pre-entrenada que suele ser VGG-16 o ResNet50, pero cambiando las capas completamente conectadas de estas redes por el *decoder*.

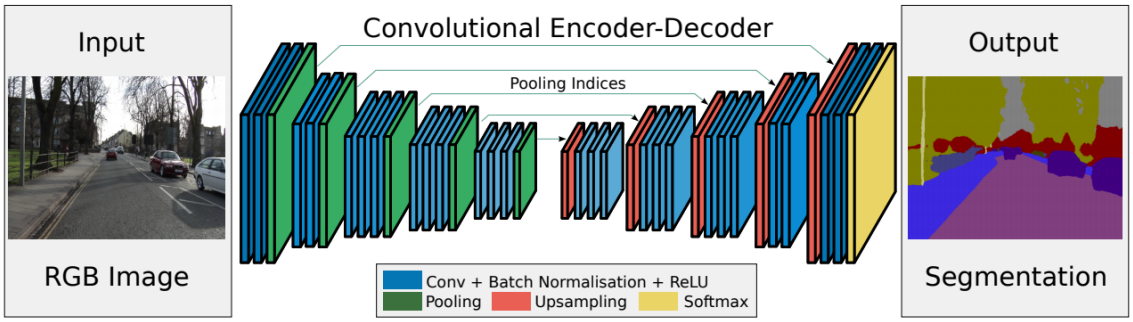


Figura 41 Arquitectura de SegNet

SegNet fue relevante ya que cambió la percepción de usar un único bloque de *upsampling* tras las capas completamente conectadas de una gran CNN para reemplazarlo por el *decoder*. De esta forma conseguía reducir el número de parámetros entrenables significativamente eliminando las capas completamente conectadas y permitiendo el entrenamiento extremo a extremo de la red. En el artículo original, fue evaluada empleando imágenes típicas de carreteras del *dataset* CamVid [33] e imágenes de interiores SUN RGB-D [34].

Con la publicación de SegNet se proporcionó una implementación en Caffe. Sin embargo, nosotros trabajamos con Keras por lo que nos hemos basado en SegNet para crear un modelo similar pero con las excepciones que describimos en la Sección 4.2.

4.1 Arquitectura original

La arquitectura de la parte del *encoder* es topológicamente idéntica a las 13 capas convolucionales de la red VGG16 [35] diseñadas para la clasificación de objetos. La salida del *decoder* se emplea como entrada a la función *soft-max* de múltiples clases para generar la probabilidad de las clases para cada píxel independientemente y asignar a cada pixel la etiqueta de la clase más probable.

Encoder

Las capas del *encoder* son las típicas que componen las CNN: Convolucionales, de normalización y *pooling* pero organizadas en bloques denominados codificadores. En la Figura 41 cada codificador se representa con varias capas, azules (*Conv+BatchNormalisation+ReLU*) y una verde (*Pooling*). Las capas convolucionales tienen varios filtros para producir mapas de características locales de su entrada, las capas de *pooling* con $f = 2$ y $s = 2$ disminuyen la dimensión del mapa de características generado con las convoluciones y propagan las características extraídas al siguiente codificador siguiendo la estructura clásica de CNN de aprendizaje jerarquizado de características (Figura 20). Las capas de normalización normalizan la distribución de los datos de entrenamiento con el objetivo de acelerar el aprendizaje y los índices del *pooling* de cada codificador se almacenan para emplearlos en el *decoder* en la reconstrucción de la salida hasta llegar al tamaño original de entrada.

Básicamente, el *encoder* a partir de la imagen original (información de bajo nivel) extrae características de alto nivel (como "coche", "peatón") como cualquier otra CNN. El *decoder* recibe esta información y la asigna de nuevo a un nivel medio y la transforma hasta el tamaño original haciendo uso de los índices del *pooling* para no perder la posición espacial de las principales características.

Decoder

Cada codificador del *encoder* tiene su decodificador simétrico en el *decoder*. Estos comienzan con la fase de *upsampling* (ventanas rojas de la Figura 27), aquí es donde los índices del *pooling* de los codificadores se emplean para agrandar el mapa de características y seguidamente convolucionarlo con las capas azules del decodificador correspondiente. La red SegNet que emplea esta técnica de decodificación fue denominada SegNet-Basic y se muestra en la Figura 42 [7]. En el artículo original proponen otras técnicas de decodificación basadas en *Fully Convolutional Networks*, pero en nuestro proyecto no profundizamos en este tema.

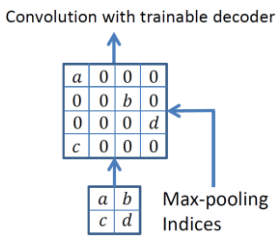


Figura 42 a, b, c, d corresponden a los valores del mapa de características que SegNet-Basic emplea para el upsampling del mapa de características y convolucionarlo seguidamente

La fase de *upsampling* del *decoder* (*Decoder Stage* en la Figura 43) es lo opuesto/simétrico al *pooling* del *encoder* (*Encoder Stage*). Aquí es donde convertimos cada celda (1x1) del resultado del *pooling* (*Pooled Map* en la Figura 43) en 2x2. Dado que la red es simétrica, los valores de *pooling* (*Decoder Stage*) se colocan en la misma posición que estaban en el mapa de características original (*Input* en la Figura 43) del codificador correspondiente haciendo uso de los *índices de pooling.*

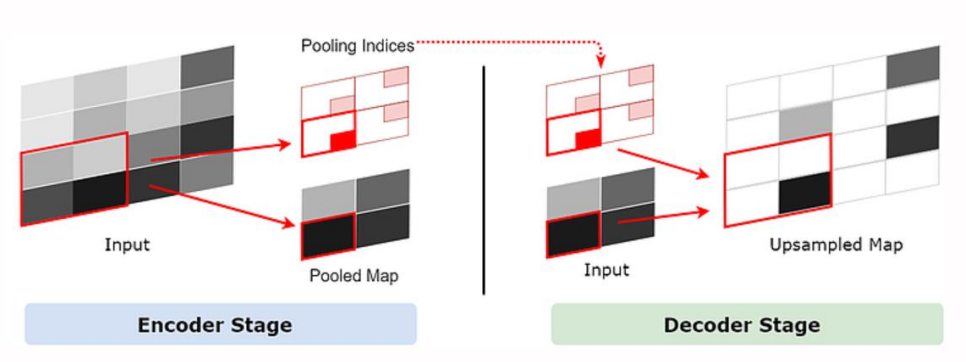


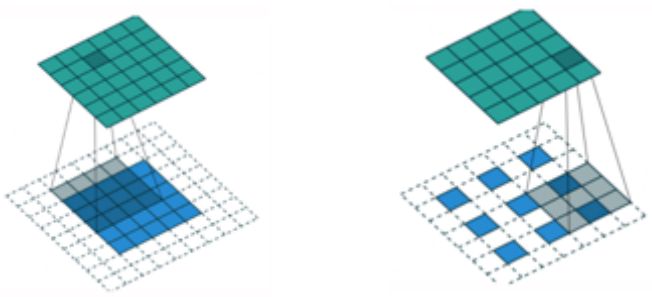
Figura 43 *Upsampling* en SegNet-Basic

Capas convolucionales en el *encoder* y en el *decoder*

Las capas convolucionales del *encoder* y del *decoder* funcionan exactamente igual, pero tienen interpretaciones ligeramente diferentes. En la fase del *encoder*, extraen características locales y las pasan a la capa de *pooling*. Es decir, el entrenamiento hace que se aprendan los filtros de convolución adecuados para extracción de características.

Por el contrario, en la fase del *decoder* el mapa de características se aumenta con los valores e índices del *pooling* antes de las capas convolucionales donde por cada área de 2x2 solo un valor se propaga desde el *encoder*, el resto serán completados por la capa de convolución. Por lo que las capas convolucionales de los decodificadores aprenden a "añadir" características al mapa original para "suavizarlo".

En conclusión, las capas convolucionales son computacionalmente idénticas, pero debido a la entrada que reciben, da la sensación de que se comportan de manera diferente. En cierta literatura se les denomina "*Deconvolución*" o "*Convolución transpuesta*". A continuación mostramos una figura que ilustran la convolución típica (Figura 44 a) y la denominada "Deconvolución" (Figura 44b).



(a) (b)

Figura 44 Azul: input; Verde: output; Convolución (a) y "*Deconvolución*" (b)

Capa de salida

El mapa de características de alto nivel generado tras las convoluciones del último decodificador del *decoder* se emplea para alimentar un clasificador *softmax* de N clases. La salida de la capa *softmax* es una imagen del K canales donde K es el número de clases y contiene la probabilidad de que cada pixel sea clasificado a cada una de las clases. La segmentación final corresponde a la seleccionar la clase con mayor probabilidad cada píxel.

A diferencia de SegNet, U-Net [23] (propuesta para imágenes médicas) no reutiliza los índices de *pooling*, sino que transfiere todo el mapa de características (a costa de más memoria) a los decodificadores correspondientes, por lo que estos no necesitan la fase de *upsampling*.

4.2 Modificaciones

Transferencia de información al *decoder*

La principal novedad de SegNet fue su forma de transferir información al decodificador. En la Figura 45 mostramos un ejemplo concreto del mecanismo original de *upsampling* que emplea los índices descrito en la sección anterior y el mecanismo final que hemos empleado. En primera instancia implementamos SegNet original en Keras pero encontrábamos resultados sospechosamente incorrectos. Nuestra primera sospecha fue que el método de *upsampling* de SegNet no funcionaba correctamente ya que para el resto de operaciones de la red hemos empleado funciones de Keras. Por este motivo decidimos simplemente probar con un mecanismo de *upsampling* tradicional que replica cada valor y los resultados mejoraron significativamente.

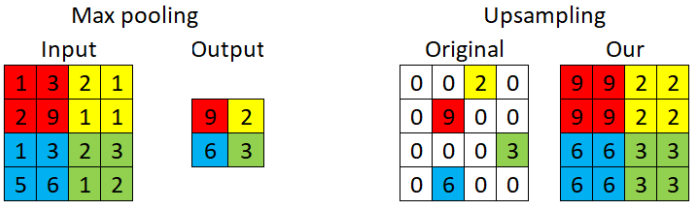


Figura 45 *Upsampling* modificado

En conclusión, no transferimos los índices sino que únicamente los valores resultantes del *pooling* replicados en los correspondientes huecos del mapa de características.

Balanceo de clases y *loss function*

Al igual que en el modelo original, empleamos la *entropía cruzada categórica* como función de coste (*loss function*) a optimizar en cada *batch*. En segmentación semántica, a la hora del entrenamiento de la red, debemos tener en cuenta el número de píxeles de cada clase. Por ejemplo, en el *dataset* CamVid con el que se presentó el modelo original, los píxeles de carretera, el cielo y el edificio dominan el *dataset*. Cuando ocurre esto, existe una gran variación en el número de píxeles de cada clase en el conjunto de entrenamiento, entonces es necesario ponderar la aportación de cada pixel al *loss* según el número de píxeles de cada clase que aparece en un *batch* determinado, lo que se denomina balanceo de las clases.

En el artículo original, para el balanceo de las clases emplearon *median frecuency balancing* [36]. Debido a que nosotros hemos entrenado SegNet adaptada a las imágenes de SIWUAC, el balanceo de las clases ha sido uno de los aspectos de estudio durante la fase de experimentación, donde hemos realizado pruebas con diferentes métodos de balanceo en el cálculo del *loss*:

1. *Natural frequency balancing*: El aporte al *loss* de cada píxel es inversamente proporcional al número de píxeles de su clase.
2. *Median frequency balancing:* Cada píxel se pondera con $\alpha_c = median\_freq/freq(c)$ donde $freq(c)$ es el número de píxeles de la clase c dividido por el número de píxeles de las imágenes donde c está presente y $median_freq$es la mediana de estas frecuencias.

3. Sin balanceo: No se tiene en cuenta la cantidad de píxeles por clase.

Por último, como hemos explicado en la Sección 3.2 tras el rasterizado, las máscaras pueden contener valores con 0 y 255 que indican que se desconoce la clase de ese pixel. Por lo tanto hemos modificado estos métodos de balanceo para que en el cálculo del *loss,* el modelo no tenga en cuenta dichos valores en las máscaras de tal forma que no aprende a predecirlos, ya que no tenemos datos en la máscara (*ground truth*) con el que aprender.

5 Estudio experimental

Debido a que SIWUAC permite generar *datasets* de imágenes con características muy variadas, hemos decidido comenzar estudiando el comportamiento de SegNet con imágenes de diferentes tamaños ya que este es uno de los parámetros más importantes de SIWUAC, el tamaño de ventana. Durante el entrenamiento de los modelos hemos empleado los mosaicos sin solapamiento de varias ciudades debido a que los mosaicos con píxeles solapados se pueden entender como un método de *data augmentation* ya que incluyen zonas repetidas en el *dataset* generado, lo cual requeriría su correspondiente estudio experimental.

5.1 Marco experimental

Conjuntos de imágenes de SIWUAC

Como podemos observar en la Tabla 5 disponemos de 263 imágenes de 1024x1024x4, sin solapamiento y organizadas en 10 ciudades para estudiar el comportamiento de SegNet. Las máscaras correspondientes, al menos tienen un pixel clasificado (valor diferente de 255 o 0), es decir no existen imágenes con máscaras completas con 255 (No cubierto por Urban Atlas) y/o 0 (Clasificado como nube, sombra,.. en Urban Atlas).

A partir de estas 263 imágenes, generamos otros dos *datasets* con imágenes de menor tamaño, en concreto 256 y 512. En este proceso se ha establecido el 25% como mínima cobertura de clases, es decir, las máscaras al menos tendrán un 25% de píxeles con valor diferente de 0 y/o 255. Por lo tanto, tenemos tres *datasets*, con tamaños, 1024, 512 y 256 cuyas imágenes ocupan 8.64MB, 2MB, y 512KB cada una respectivamente. La Tabla 6 muestra el número de imágenes de los mosaicos sin solapamiento de las 10 ciudades empleadas en los experimentos mostrados en la siguiente sección.

Ciudad	Tamaño de ventana		
	1024	**512**	**256**
Alicante	4	13	45
Cartagena	8	21	70
Córdoba	30	91	316
Jaén	12	42	142
Lorca	30	90	314
Madrid	110	344	1275
Murcia	24	65	224
Pamplona	32	87	296
Reus	1	4	13
Santander	12	40	132
Total	**263**	**797**	**2827**
GB	**2.22**	**1.56**	**1.38**

Tabla 6 Número de imágenes de tres datasets sin solapamiento

A partir de estos *datasets*, seleccionamos 6 ciudades para entrenar (*conjunto train*) y 4 para evaluar los modelos entrenados (*conjunto test*). Ambos conjuntos contienen ciudades de diferentes localizaciones geográficas con el objetivo de que exista diversidad entre ellos. En

cuanto al número de imágenes, hemos mantenido un 75% y 25% aproximadamente entre el entrenamiento y evaluación. Las Tabla 7, Tabla 8 y Tabla 9 muestran el número de imágenes de cada ciudad empleadas para entrenamiento y evaluación de los *datasets* de 1024, 512 y 256.

		División *train* y *test*						Total	
1024	***Train***	**Cartagena**	**Córdoba**	**Jaén**	**Lorca**	**Madrid**	**Reus**	**191**	**72.62%**
		8	30	12	30	110	1		
	Test	**Pamplona**	**Murcia**	**Santander**	**Alicante**			**72**	**27.38%**
		32	24	12	4				
								263	**100%**

Tabla 7 División de ciudades de entrenamiento y evaluación del *dataset* de 1024

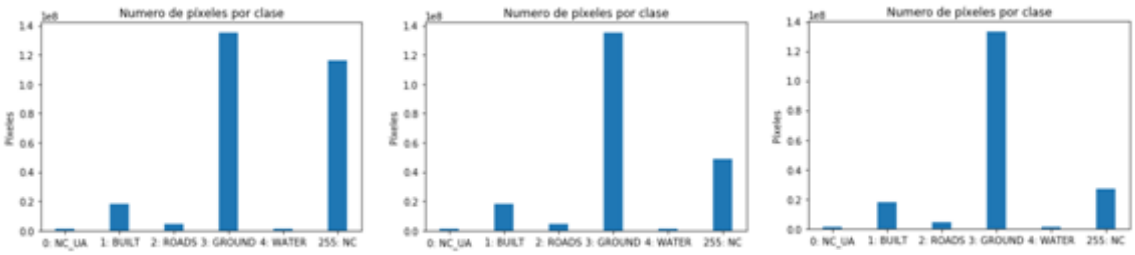
		División *train* y *test*						Total	
512	***Train***	**Cartagena**	**Córdoba**	**Jaén**	**Lorca**	**Madrid**	**Reus**	**592**	**74.28%**
		21	91	42	90	344	4		
	Test	**Pamplona**	**Murcia**	**Santander**	**Alicante**			**205**	**25.72%**
		87	65	40	13				
								797	**100%**

Tabla 8 División de ciudades de entrenamiento y evaluación del *dataset* de 512

		División *train* y *test*						Total	
256	***Train***	**Cartagena**	**Córdoba**	**Jaén**	**Lorca**	**Madrid**	**Reus**	**2130**	**75.34%**
		70	316	142	314	1275	13		
	Test	**Pamplona**	**Murcia**	**Santander**	**Alicante**			**697**	**24.66%**
		296	224	132	45				
								2827	**100%**

Tabla 9 División de ciudades de entrenamiento y evaluación del *dataset* de 256

Las gráficas de la muestran la cantidad de píxeles que tiene cada clase en cada uno de los *datasets* anteriores. En ellas apreciamos que el número de píxeles no clasificados (NC) se reduce a medida que se reduce el tamaño de las imágenes. Esto es debido a que al dividir una máscara de 1024 en otras más pequeñas, las imágenes resultantes pueden estar en zonas sin etiquetas (clase 255) y por tanto se eliminan (no llegan al grado mínimo de clases establecido). reduciendo el número de píxeles total pertenecientes a esta clase.



(a) 1024 (a) 512 (a) 256

Figura 46 Píxeles por clase en cada *dataset*

Medidas de rendimiento

Para poder evaluar adecuadamente la calidad de los modelos entrenados, necesitamos medir honestamente la calidad de los mismos. Para ello es necesario definir las medidas de rendimiento que vamos a emplear. Si vemos la segmentación semántica como una clasificación multiclase a nivel de pixel, la primera opción que tenemos es calcular el porcentaje de píxeles bien clasificados, es decir el *accuracy.* Otra alternativa es la medida conocida como IoU (*Intersection over Union*), que se emplea habitualmente para evaluar la localización de objetos.

El *accuracy* consiste en el conteo del número de píxeles que coinciden entre la máscara original y la predicción obtenida por el modelo presentado como porcentaje.

$$accuracy = \frac{píxeles\ correctamente\ clasificados}{píxeles\ totales}$$

El IoU, también es conocido como índice de Jaccard, mide el porcentaje de superposición entre la máscara real y la predicción obtenida a la salida del modelo. Como su nombre indica, se calcula dividiendo el número de píxeles comunes entre la máscara y la predicción entre el número de píxeles totales de ambas máscaras. Este cálculo se realiza para cada clase por separado y luego se promedia para obtener la medida de IoU global de nuestra predicción

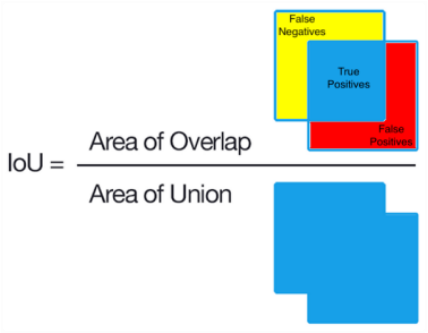


Figura 47 Verdaderos positivos, los falsos positivos y los falsos negativos descritos

Es necesario destacar que en la evaluación de los modelos, al igual que en el cálculo del *loss*, hay que tener en cuenta que las máscaras contienen valores con 0 o 255. En este caso, a la hora de comparar las predicciones con máscaras originales para calcular el rendimiento del modelo, los píxeles etiquetados con 0 o 255 son ignorados en el cálculo ya que en la predicción nunca estarán presentes.

5.2 Pruebas realizadas

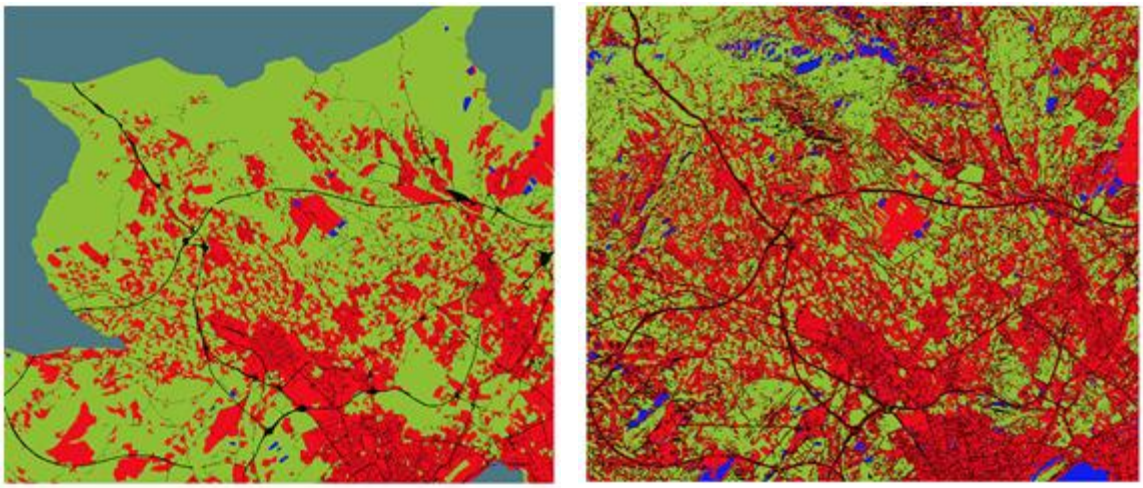
En esta sección describimos brevemente las pruebas realizadas y conclusiones obtenidas hasta alcanzar los resultados que presentamos en la Sección 5.3. La mayor parte del proyecto nos ha llevado el desarrollo de SIWUAC y la generación de *datasets.* Por este motivo, tras desarrollar nuestro modelo de SegNet modificado, hemos seguido los siguientes pasos para encontrar un modelo que proporcione buenos resultados.

5.2.1 Establecer el tamaño de imagen

Con los tres *datasets* descritos en la sección anterior y las medidas de rendimiento definidas, disponemos de un amplio marco de trabajo formado por imágenes satelitales etiquetadas de diferentes tamaños para comenzar con la experimentación. El objetivo de esta primera prueba ha sido decidir cuál es el tamaño de imagen adecuado para trabajar en segmentación semántica con SegNet. Para ello, hemos configurado su entrenamiento de la siguiente manera:

- Épocas: 600
- Tamaño de *batch*: 12 (256), 4 (512) y 1 (1024) por cuestiones de memoria.
- Optimizador: Adam
- *Callbacks*:
  1. Detener el entrenamiento si no mejora el *loss* en el conjunto de *test* durante 50 épocas.
  2. Reducir la tasa de aprendizaje (*learning rate*) si el *loss* en *test* no mejora durante 30 épocas
- Balanceo de clases: *Natural frequency balancing*

Durante el aprendizaje obtenemos 8, 25, y 42 segundos por cada época con los *datasets* de 256, 512 y 1024, respectivamente ejecutándose sobre una GPU Nvidia GeForce RTX 2080. Los resultados, todavía son de baja calidad. En la Figura 48 mostramos la primera predicción obtenida sobre una zona de Alicante (*conjunto de test*) con el *dataset* de 256 por ser el que mejores resultados ha generado.



(a) *Ground truth* (b) Primera predicción

Figura 48 Zona de Alicante: *ground truth* (a) y primera predicción (b)

La Tabla 10 muestra únicamente el *accuracy* obtenido con los *datasets* de 256 y 512 debido que el de 1024 ha finalizado por falta de memoria.

	256	**512**
Accuracy	0.5348	0.2653

Tabla 10 Resultados de la primera prueba

Con esta prueba preliminar no hemos obtenido buenos resultados, pero han sido muy útiles para determinar las siguientes conclusiones:

1. **Tamaño adecuado 256**: Mejor rendimiento en *test* (*accuracy* de 53.48%) entre los tres *datasets.*
2. **Necesaria una reclasificación de las máscaras**: La clasificación presentada en el Capítulo 3, ha resultado ser demasiado grosera ya que las máscaras han podido confundir a la red. Por esto hemos decidido realizar un repaso de la agrupación de clases de Urban Atlas empleada.
3. **Necesario un balanceo de clases diferente**: Urban Atlas tiene muy pocas superficies de agua clasificadas como tal, ya que se centra en grandes zonas urbanas. En cambio, en las primeras predicciones obtenidas se observan píxeles predichos como agua en excesivas zonas, tanto urbanas como exteriores. Nuestra primera sospecha se ha centrado en su posible relación con el balanceo de clases empleado ya que favorece a predecir las clases que menos píxeles tienen.
4. **Correcto funcionamiento en zonas no cubiertas por Urban Atlas**: En la Figura 48 podemos observar como las grandes carreteras son clasificadas perfectamente en las zonas que quedan fuera de la superficie que cubre Urban Atlas del *ground truth*.

A partir de estas conclusiones se han definido los siguientes pasos a seguir:

1. Reclasificación de las máscaras y creación de un nuevo *dataset* (presentado a continuación)
2. Estudio del rendimiento con diferentes métodos de balanceo de clases (Sección 5.3)

5.2.2 Reclasificación de Urban Atlas

Debido que hemos observado que los resultados no eran de excesiva calidad, hemos realizado un repaso de la clasificación presentada en la EOTask UrbanAtlasReclassify y hemos encontrado el siguiente problema: Urban Atlas contiene clases en las que dentro de sus áreas, hay superficies que nos gustaría subdividirlas, pero las tenemos que asignar a una única clase en nuestra clasificación debido a que Urban Atlas no las diferencia. Por este motivo hemos realizado los cambios que presentamos en la Tabla 11 con el objetivo de mejorar los resultados de las primeras pruebas.

Urban Atlas	Primera clasificación	Nueva clasificación	Comentarios
12400 Aeropuertos	URBANO	NO CLASE	Contienen enormes carreteras
13100 Extracción de minerales y vertederos	URBANO	NO CLASE	Suelos muy diversos y zonas edificadas.
13300 Sitios de construcción	URBANO	SUELO	Son zonas urbanas pero sin edificaciones, suelo edificable.
14100 Áreas urbanas verdes	URBANO	SUELO	Parques urbanos, la mayoría es suelo, sin edificaciones, medianas,…
14200 Instalaciones deportivas y de ocio	URBANO	NO CLASE	Contienen desde campos de golf hasta polideportivos, hoteles,…

Tabla 11 Ajuste de la clasificación en EOTask UrbanAtlasReclassify

En resumen, hemos decidido no aprender de las clases 12400, 13100 y 12400 clasificándolas como NO CLASE ya que contienen zonas que nos gustaría dividir en otras clases y pueden confundir a la red en la predicción. Por ejemplo en los aeropuertos nos gustaría mantener las carreteras como VIAS, el suelo como SUELO. En la zona marcada en amarillo de la Figura 49 podemos observar las grandes carreteras del aeropuerto de barajas clasificadas como URBANO.

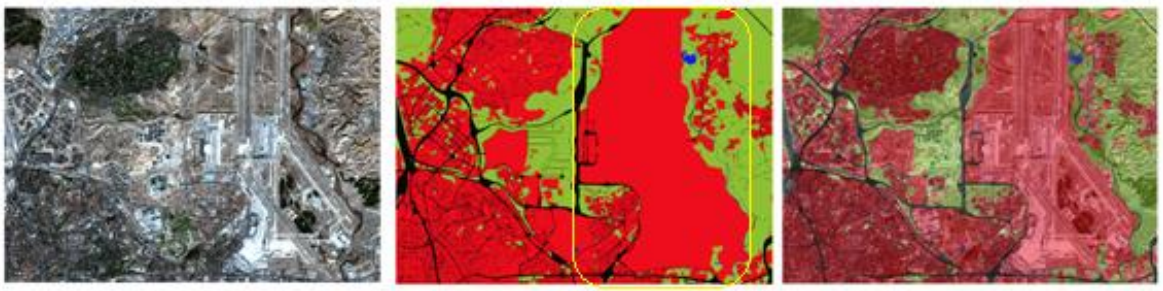


Figura 49 Aeropuerto de barajas con grandes carreteras sin definir en la máscara como

A la vista de estas deficiencias del conjunto de entrenamiento, hemos decidido continuar con nuestra experimentación con un nuevo *dataset* de imágenes de 256. La clasificación empleada para los siguientes experimentos es la que mostramos en la Tabla 12.

Clasificación propia mejorada		
AGRUPACION	**ITEM**	**CODE**
11100,11210,11220,11230,11240,11300,12100,12300	URBANO	1
12210,12230,12220	VIAS	2
13400,21000,22000,23000,24000,25000,31000,32000,33000,**13300**, **14100**	SUELO	3
40000,50000	AGUA	4
91000,92000,**12400,13100,14200**	NO CLASE	0

Tabla 12 Nueva clasificación

Al modificar las máscaras, estamos modificando la distribución de las clases en el *dataset*, por este motivo en la siguiente sección presentamos resultados obtenidos entrenando SegNet con la clasificación anterior aplicando diferentes métodos de balanceo de clases.

5.3 Resultados

Una vez definida una nueva clasificación y generado un nuevo *dataset*, en esta sección presentamos los resultados obtenidos empleando diferentes métodos de balanceo de clases. En primer lugar en el apartado 5.3.1 mostramos los resultados cuantitativos de todos los experimentos y a continuación las gráficas de la evolución del aprendizaje de SegNet en el mejor de los casos. Por último, en el apartado 5.3.2 mostramos los resultados cualitativos de todos los modelos sobre una zona de estudio de cada ciudad del conjunto de *test*. Como resultados finales mostramos el mosaico de clasificación completo formado por las imágenes que forman el conjunto de *test* al completo (Alicante, Murcia, Pamplona y Santander).

Como ya hemos dicho, nuestro *dataset* no está equilibrado, es decir, existen clases que dominan el *dataset* como SUELO y también clases que casi no aparecen como AGUA (ver Figura 46). Por este motivo, los experimentos los hemos enfocado a solventar este problema variando los métodos de balanceo de clases. Los experimentos realizados se han llevado a cabo en el siguiente orden cronológico.

1. ***Natural frequency balancing*** (Natural)
2. ***Median frequency balancing*** (Median)
3. ***Median frequency balancing*** (Median_w1): Modificado para que los píxeles de AGUA se ponderen con el valor de la clase con presencia mediana.
4. ***No balancing*** (No)
5. ***Natural frequency balancing*** (Natural_w1): Modificado para que $\alpha_{water} = 1$

En este caso, para cada una de las pruebas hemos empleado la misma configuración de parámetros que en el experimento anterior pero con dos salvedades:

1. **Tamaño de batch**: 24, en la primera prueba ha sido 12.
2. ***Data augmentation:*** A cada imagen de entrenamiento se le aplican *FlipAugmenter* y *RotationAugmenter*. El primero consiste en voltear una la imagen vertical y horizontalmente, y el segundo en girarla 90, 180 y 270 grados, por lo que con una imagen generamos 5 más.

5.3.1 Resultados cuantitativos

En primer lugar mostramos cuantitativamente los resultados obtenidos con cada uno de los experimentos. La

Métricas	Natural	Median	Median_w1	No	Natural_w1
Aaccuracy	0.8658	0.8080	0.7197	0.9338	0.8531
mIoU	0.4277	0.3193	0.3602	0.3821	0.4498

Tabla 13 contiene el *accuracy* y el mIoU obtenidos con escalas de color, representando el rojo el valor más pequeño y el verde el mayor.

Métricas	Natural	Median	Median_w1	No	Natural_w1
Aaccuracy	0.8658	0.8080	0.7197	0.9338	0.8531
mIoU	0.4277	0.3193	0.3602	0.3821	0.4498

Tabla 13 Rendimiento de SegNet en *test* con diferentes métodos de balanceo

A continuación, en la Figura 50 podemos observar los resultados anteriores pero representados en un gráfico de barras.

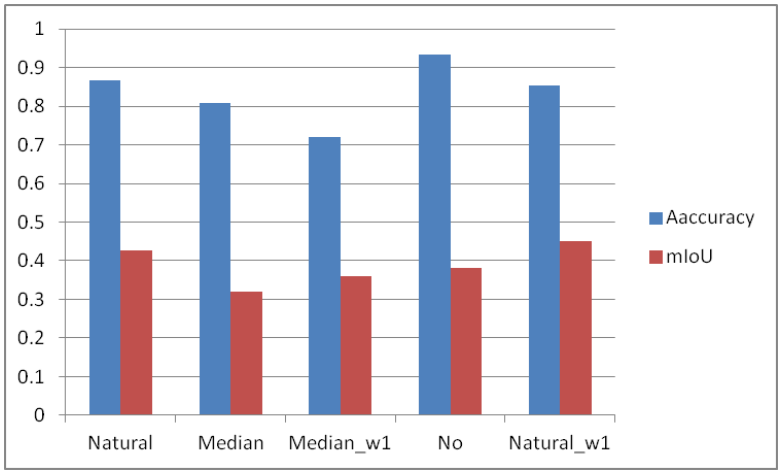


Figura 50 Representación gráfica de las métricas

A primera vista observamos que los mejores resultados son correspondientes al *Natural frequency balancing* original y modificado, junto con el experimento sin balanceo. Curiosamente al contrario que en el artículo original, *Median frequency balancing* no ha sido adecuado para nuestro conjunto de datos ya que ha obtenido los peores resultados, tanto con el original como con el modificado.

Debido a que las diferencias numéricas no son excesivamente significativas, en la Tabla 14 mostramos el desglose del mIoU y al igual que en el caso anterior, la Figura 51 representa los mismos resultados en un gráfico de barras.

	Natural	Median	Median_w1	No	Natural_w1
IoU URBANO	0.2933	0.3073	0.1363	0.3530	0.3243
IoU VIAS	0.1617	0.1446	0.0799	0.0816	0.1629
IoU SUELO	0.8043	0.7683	0.6455	0.9147	0.7992
IoU AGUA	0.4517	0.0571	0.5789	0.1793	0.5129

Tabla 14 Desglose del mIoU por cada clase

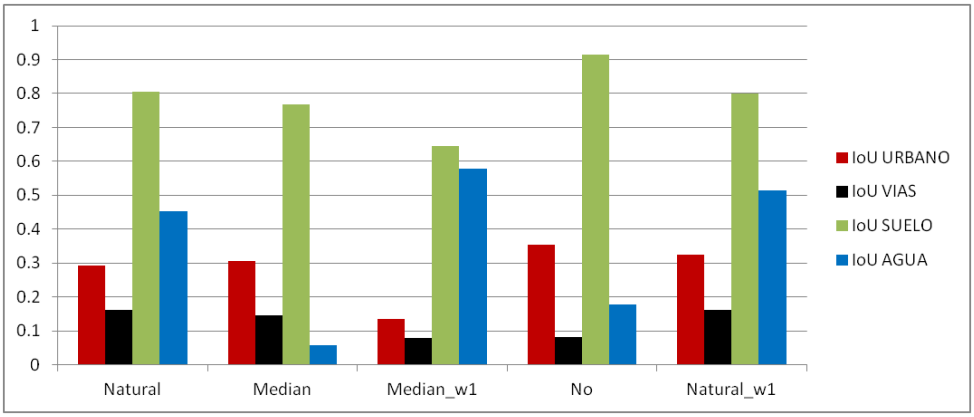


Figura 51 Desglose de mIoU

Seguimos apreciando que los peores resultados los obtienen los métodos Median y Median_w1. Si nos centramos en Natural y Natural_w1, el segundo supera o iguala en todas las clases excepto en SUELO pero por una diferencia de 0.0051. Además el *No balancing* obtiene muy buenos resultados en las clases URBANO y SUELO (las que dominan el *dataset*) pero es deficiente prediciendo VIAS y AGUA. Por lo tanto, cuantitativamente el método que mejores resultados ofrece el Natural_w1.

5.3.2 Resultados cualitativos

En este apartado, se confirman los resultados anteriores mostrando las predicciones obtenidas en una zona de estudio de cada ciudad del conjunto de *test* pero mostrados en imágenes. La Figura 52 muestra las zonas del conjunto de *test* que hemos seleccionado para mostrar las predicciones obtenidas con cada uno de los experimentos realizados.

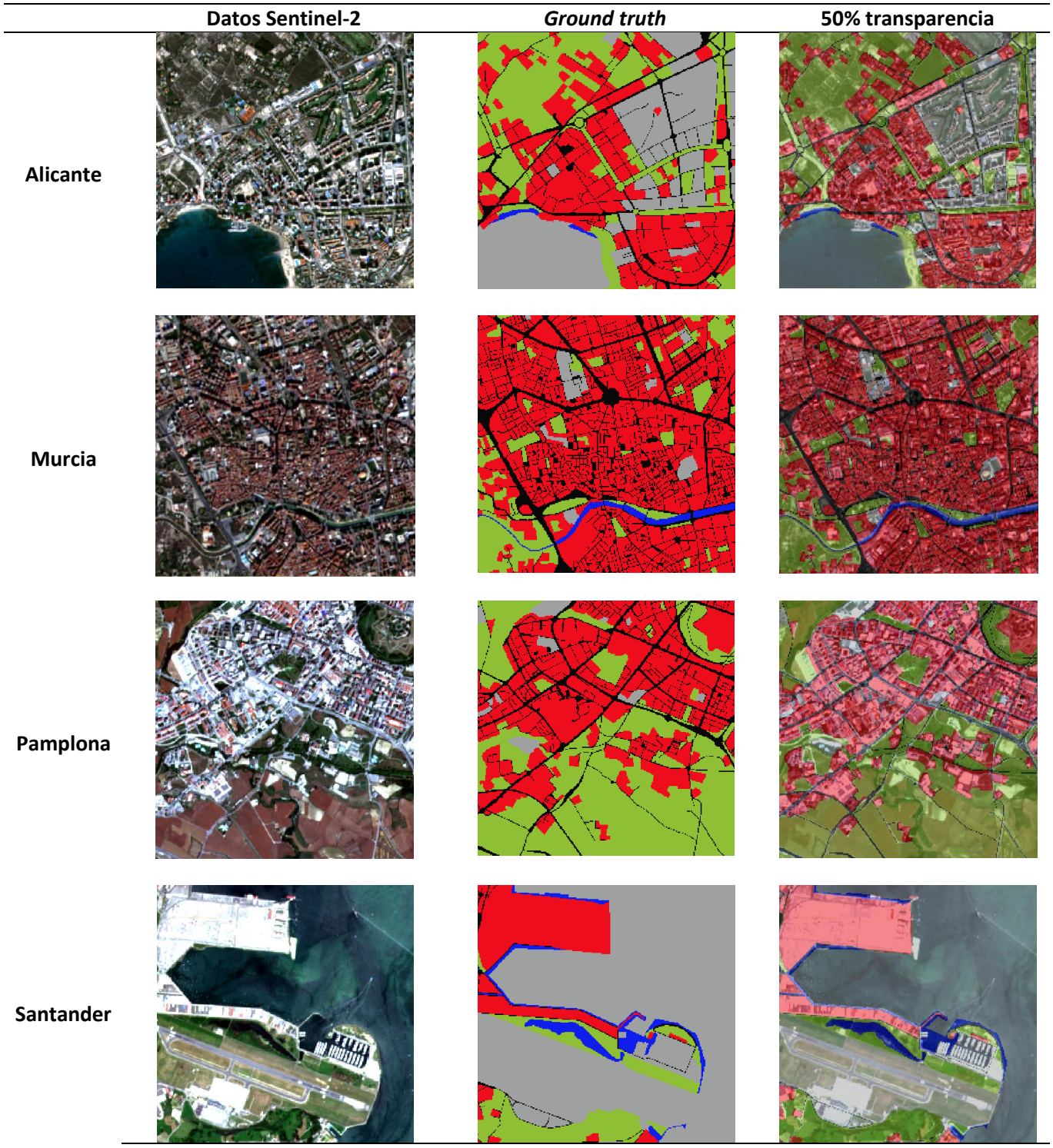


Figura 52 Zonas de estudio del conjunto de *test*

Concretamente en la máscara (*Ground truth*) de Santander, se aprecia como los aeropuertos han pasado de estar clasificados como URBANO a NO CLASE en el nuevo *dataset*. A continuación, la Figura 53 muestra la predicción que obtenemos con cada uno de los modelos entrenados con diferentes métodos de balanceo de clases (NFB, MFB, MFB1, NB NFB1) en las zonas de estudio mostradas en la Figura 52.

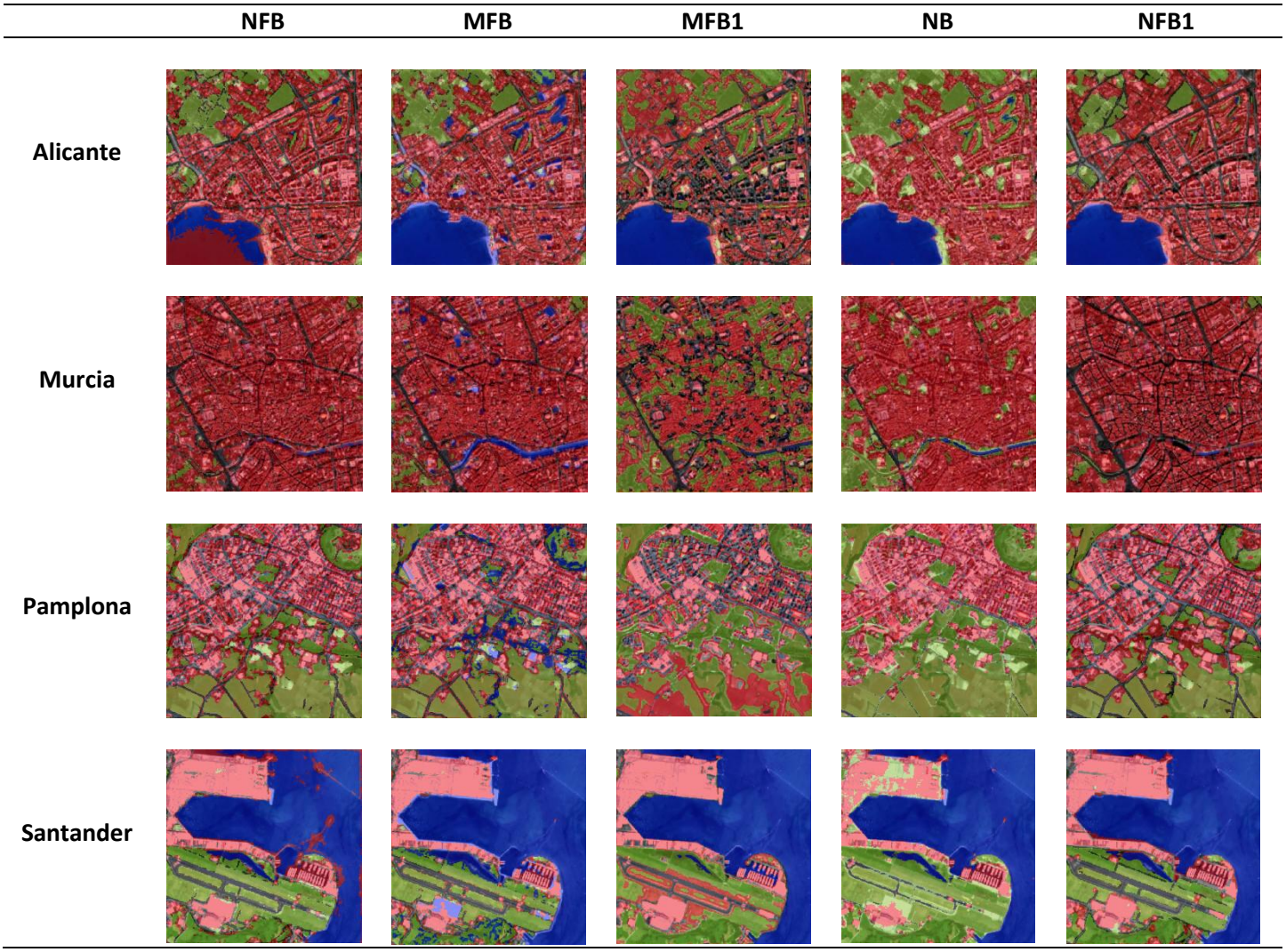


Figura 53 Resultados cualitativos de cada uno de los modelos en las zonas de estudio seleccionadas

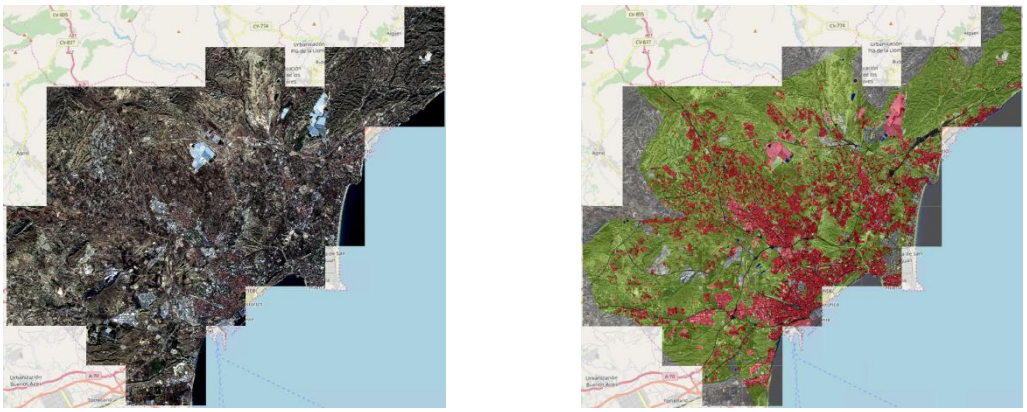
Como ya hemos comentado, estos experimentos fueron realizados cronológicamente según las decisiones tomadas observando los resultados cualitativos que ofrecía cada modelo. A continuación enumeramos las principales interpretaciones de los resultados que hemos obtenido:

1. **Natural**: Buenos resultados en la mayoría de las clases siendo el mayor problema el que zonas de mar son predichas como URBANO. Por lo demás, define formas correctamente y no presenta excesivo ruido y cuantitativamente son medidas aceptables en comparación con el resto de métodos. Para solventar el problema, la primera medida tomada fue modificar el método y emplear el usado en el artículo original, *Median frequency balancíng.*
2. **Median**: Se corrigió el problema del método anterior ya que el mar ya no es predicho como URBANO pero han aparecido problemas más graves: Se predice como AGUA zonas

del centro de la ciudad y zonas verdes por lo que el modelo no es aceptable. Con estos resultados, decidimos modificar el *Median frequency balancing* para evitar que el AGUA tome demasiada importancia.

3. **Median_w1**: En este caso en los resultados se aprecia que el modelo ha aprendido un patrón que hace que aparezcan zonas clasificadas como URBANO a los lados de las carreteras (Santander) y que muchas zonas urbanas se clasifican como VIAS (Alicante). También muchas carreteras dejan de ser detectadas y la mayoría de la imagen pasa a estar clasificada como SUELO, incluso zonas claramente urbanas (Murcia). En general, no son resultados aceptables ya que no han mejorado los anteriores. La siguiente experimentación consistió en probar a no emplear ningún método de balanceo.
4. **No**: Sorprendentemente los resultados numéricos no son excesivamente deficientes pero el modelo falla detectando las clases minoritarias, es decir define formas muy grandes de SUELO y zonas URBANAS pero no llega a definir correctamente las carreteras ni las zonas de AGUA.
5. **Natural_w1**: Ha sido el método que mejores resultados ha obtenido numérica y visualmente por lo que aquí decidimos finalizar nuestra experimentación y mostrar los resultados.

En conclusión, el Natural_w1 es el método que mejores resultados ha obtenido. Por este motivo y para concluir esta sección, las figuras que siguen a continuación muestran los mosaicos completamente clasificados de las cuatro ciudades de *test* del *dataset* obtenido tras la reclasificación de las máscaras.



(a) (b)

Figura 54 Mosaico de Alicante (a) y su *ground truth* (b)

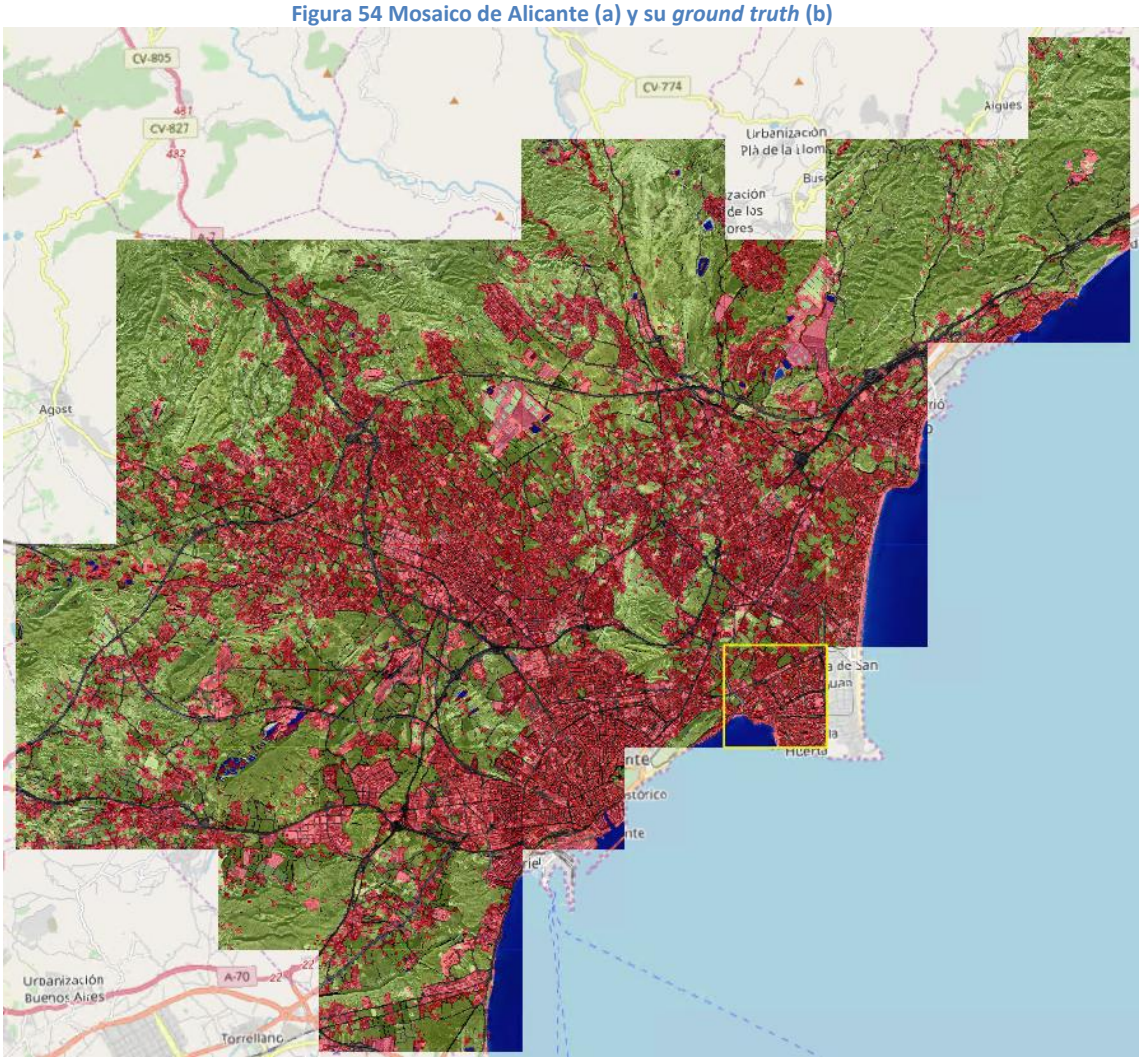


Figura 55 Predicción del mosaico de Alicante

Observamos que los resultados no presentan ruido como en las primeras predicciones de la Figura 48. El modelo es capaz de segmentar las zonas urbanas y de suelo. También predice las zonas de agua correctamente incluso las que no están presentes en la máscara como el mar. Las grandes carreteras se definen muy bien, pero para observar el comportamiento en detalle es necesario ampliar la imagen a la zona de estudio.

(a) (b)

Figura 56 Zona de estudio de Alicante (a) y su *ground truth* (b)

Figura 57 Predicción de la zona de estudio de Alicante

En la imagen se observa como separa correctamente el mar de las playas, clasificándolo como zona urbana. A este nivel, donde podemos apreciar cada pixel, observamos que con la resolución de Sentinel-2 (10m/px) es difícil definir las vías estrechas del interior de las aglomeraciones de edificios ya que muchas se representan únicamente con un pixel o "ninguno". Destacar que la parte superior izquierda, corresponden a campos de golf que no se incluyen en las máscaras y el modelo es capaz de diferenciar las zonas verdes como suelo y los lagos (casi inapreciables con el ojo humano en estas imágenes) como agua. A continuación mostramos los resultados de Murcia.

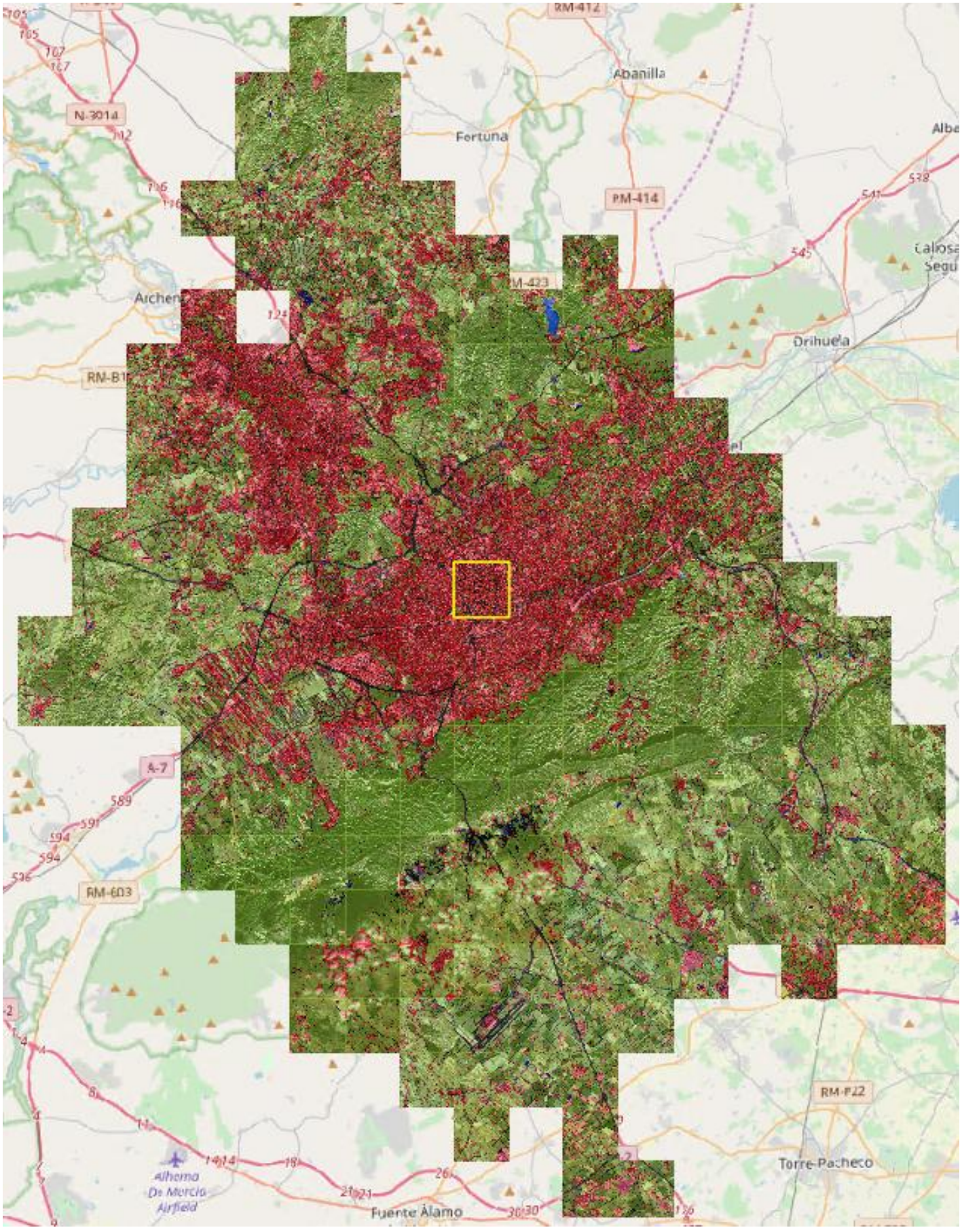


Figura 58 Predicción del mosaico de Murcia

(a) (b)

Figura 59 Zona de estudio de Murcia (a) y su *ground truth* (b)

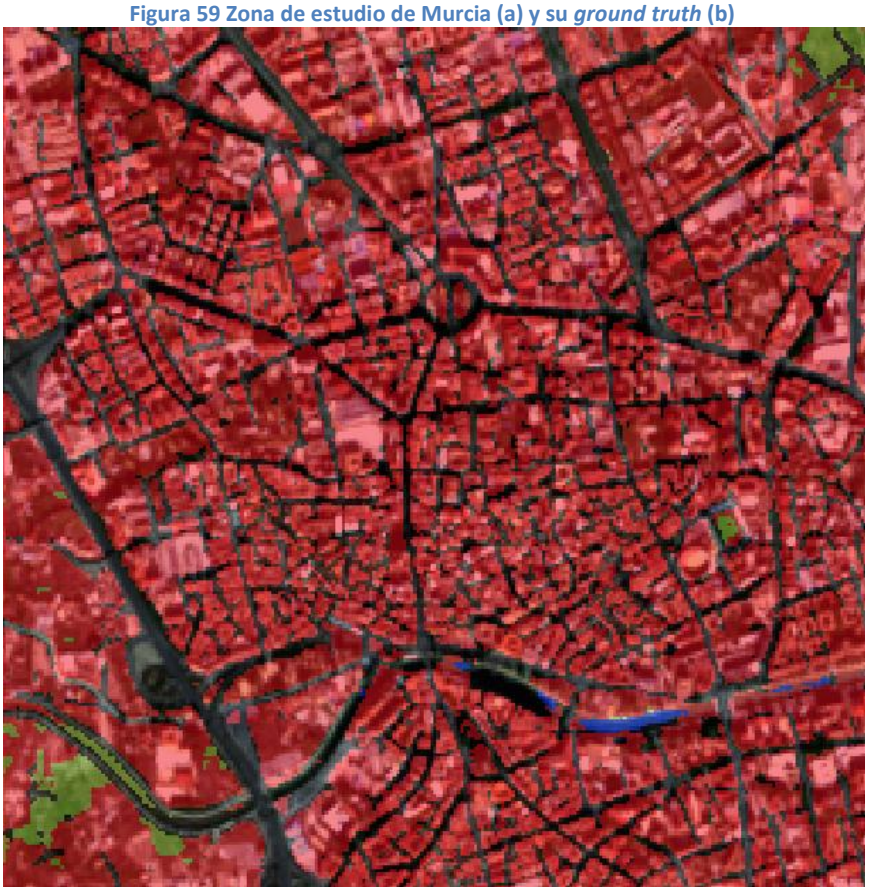


Figura 60 Predicción de la zona de estudio de Murcia

Las sombras presentes en la imagen Sentinel hacen confundir al modelo, la parte central del río Segura como carretera ya que el color es demasiado oscuro. También, en la parte izquierda, donde el cauce se estrecha y prácticamente no se reconoce agua en la imagen Sentinel, la red lo predice con carretera, pero después acaba prediciendo suelo. En esta imagen todavía se aprecia mejor la dificultad para predecir las calles o carreteras de los centros urbanos.

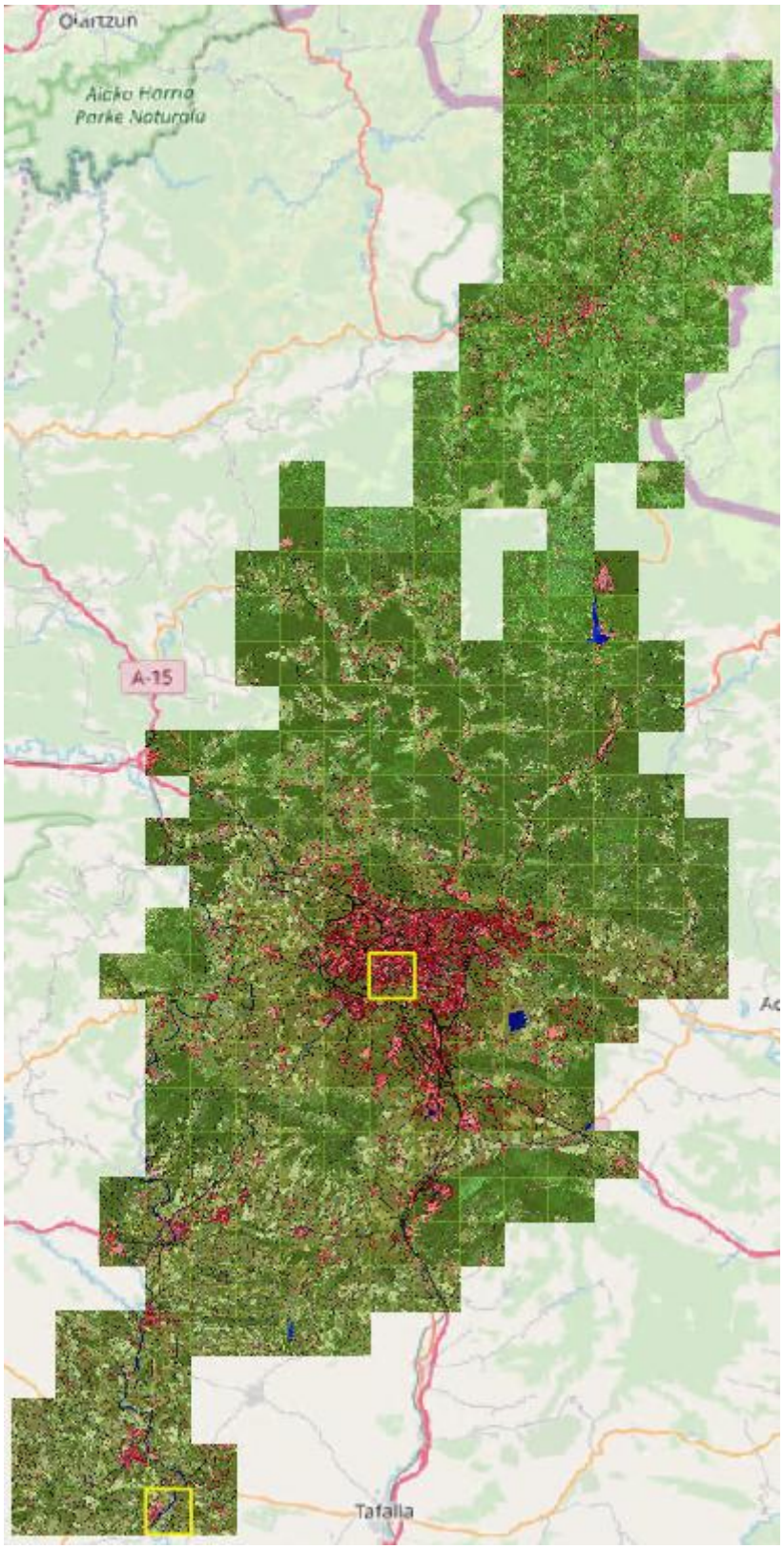
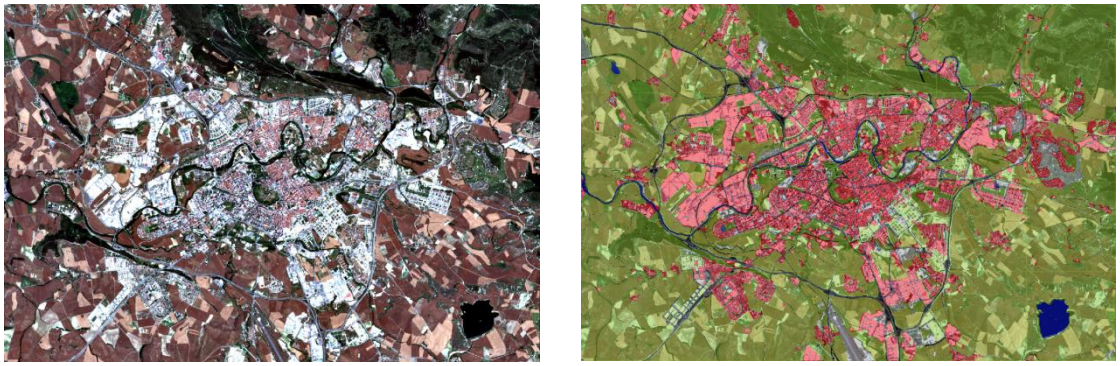


Figura 61 Predicción del mosaico de Pamplona

Debido a que el Urban Atlas de Pamplona cubre desde el norte de Navarra hasta prácticamente el centro de Navarra, el mosaico queda muy alargado y no se aprecian los detalles. Por este motivo, en este caso emplearemos dos zonas de estudio, una correspondiente a una zona de Pamplona y otra zona de estudio extra correspondiente a la

una localidad de la zona media de Navarra. En primer lugar mostramos la predicción de Pamplona al completo.



(a) Imagen Sentinel (b) *Ground truth*

Figura 62 Pamplona y alrededores formado con 24 imágenes de 256x256

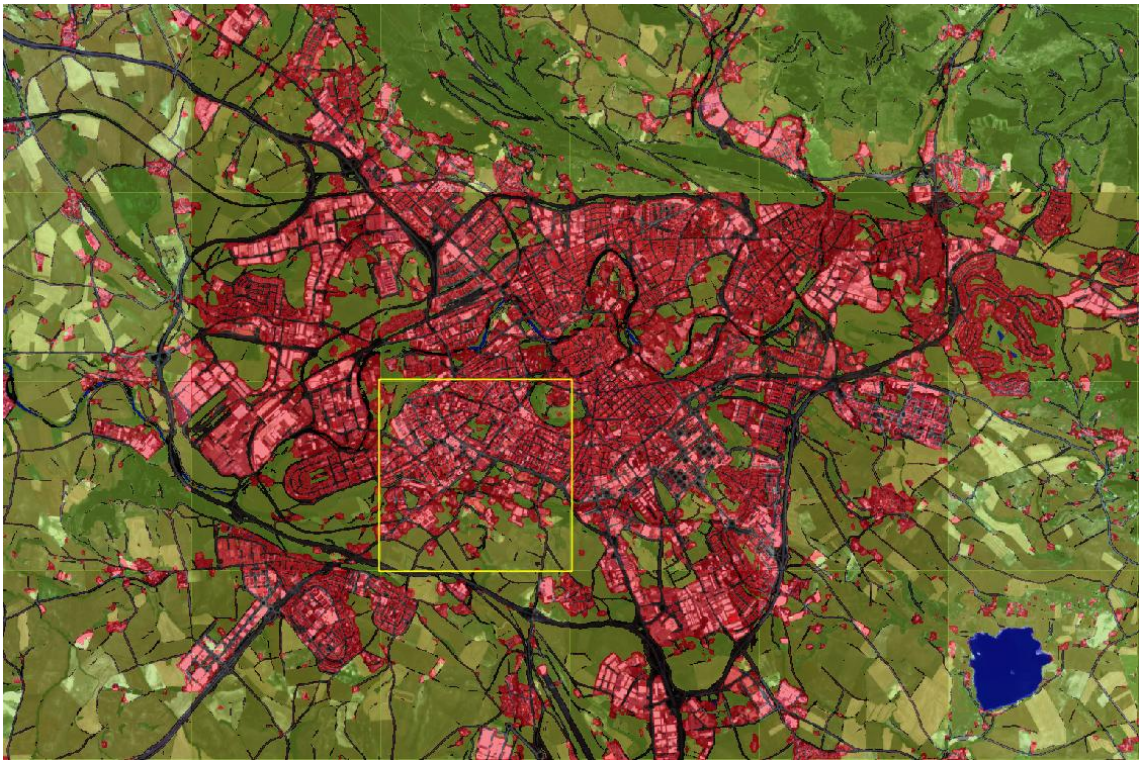


Figura 63 Predicción de Pamplona y alrededores

Lo más destacable es que el modelo es capaz de detectar los caminos de concentración como vías a pesar de no estar etiquetados como tal el *ground truth*. Por otro lado el modelo también predice edificaciones aisladas en las periferias como urbanas que en la máscara están etiquetadas como suelo. Esto, provoca una mejora visual, pero reduce las medidas de rendimiento. Por último, el modelo se confunde y el río Arga es predicho como carretera. A continuación mostramos los resultados de la primera zona de estudio ampliada.

(a) (b)

Figura 64 Primera zona de estudio de Pamplona (a) y su *ground truth* (b)

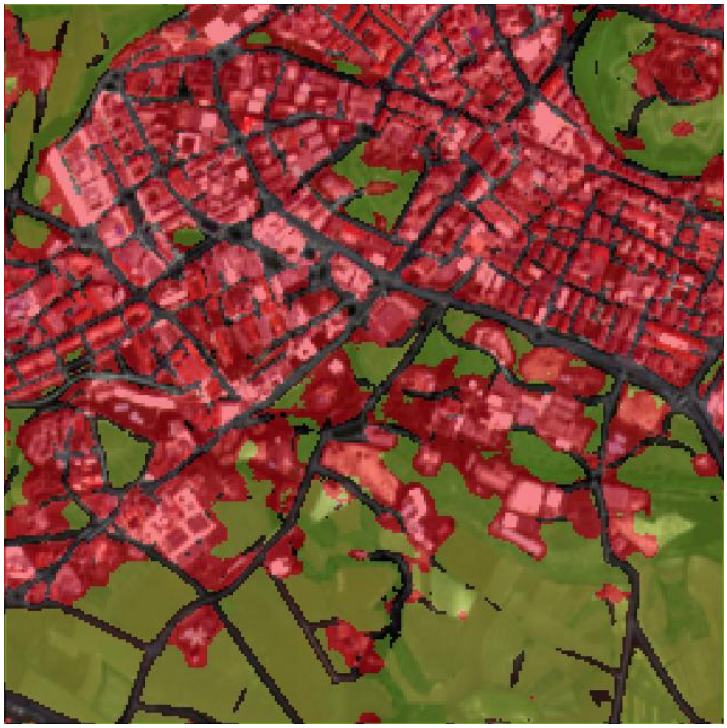


Figura 65 Zona de estudio de Pamplona

En este caso, lo el modelo diferencia de forma aceptable las zonas verdes de Pamplona. Además se observa que las carreteras se definen más gruesas en la predicción que en el *ground truth*. Como hemos comentado, debido a la extensión de Urban Atlas en el caso de Pamplona, añadimos un área de estudio extra correspondiente a la zona media de Navarra con pequeñas localidades. A continuación se muestran las imágenes de la parte más inferior del mosaico de Pamplona donde se aprecia el final de la zona mapeada por Urban Atlas.

Figura 66 Imagen Sentinel de la zona media de Navarra con 10 parches de 256x256

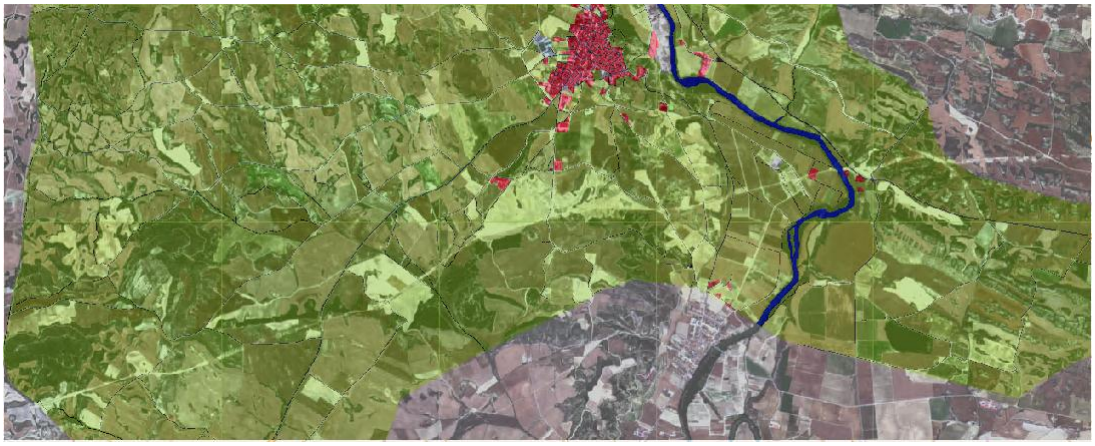


Figura 67 *Ground truth* de la zona media de Navarra

Figura 68 Predicción de la zona media de Navarra

En la Figura 67 apreciamos el límite sur de Urban Atlas de Pamplona y las zonas con NO CLASE en las máscaras, en la Figura 68 se aprecia que la red predice perfectamente los núcleos urbanos incluso los que no están presentes en el *ground truth.* El río Arga, al contrario que en Pamplona es detectado como agua perfectamente. Por otro lado, en el campo se detectan la gran mayoría de caminos de concentración y pequeñas zonas urbanas. Estas zonas, algunas corresponden a pequeñas edificaciones como almacenes, pero en otras el modelo se equivoca.

(a) (b)

Figura 69 Zona de estudio extra en la zona media de Navarra (a) y su *ground truth* (b)

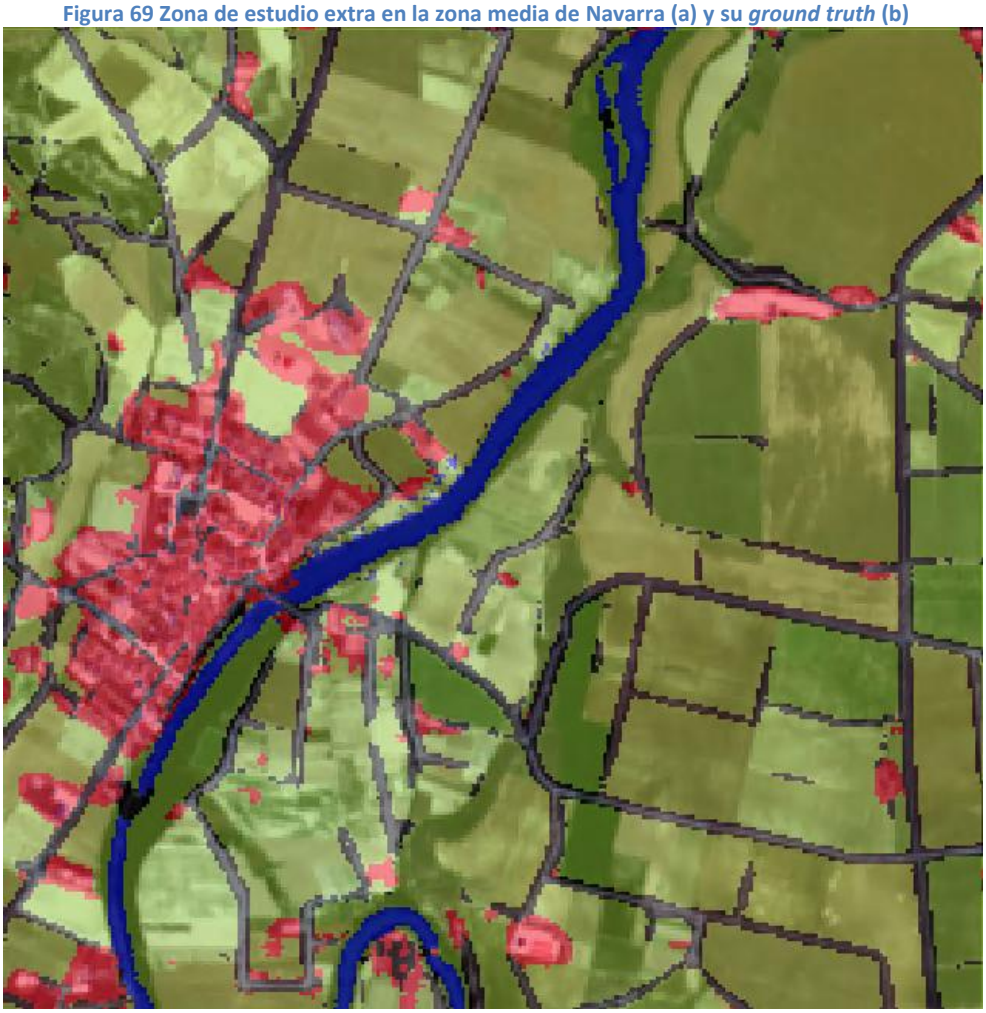
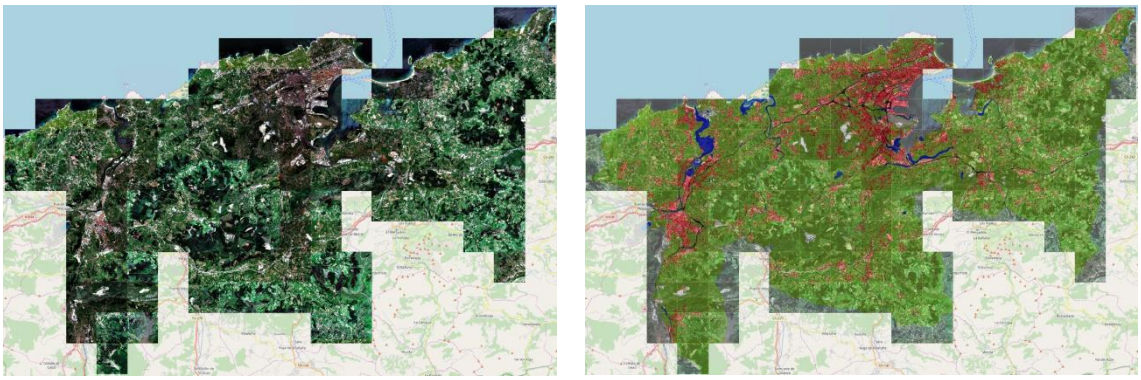


Figura 70 Predicción del área de estudio extra en la zona media de Navarra

En esta zona apreciamos claramente cómo se detectan zonas construidas en los exteriores de las localidades, el río y los caminos de concentración. Además, la parte predicha como urbana rodeada por agua en la parte inferior de la foto corresponde a una zona donde el modelo se confunde. A continuación, para finalizar con los resultados cualitativos mostramos Santander.



(a) (b)

Figura 71 Mosaico de Santander (a) y *ground truth* (b)

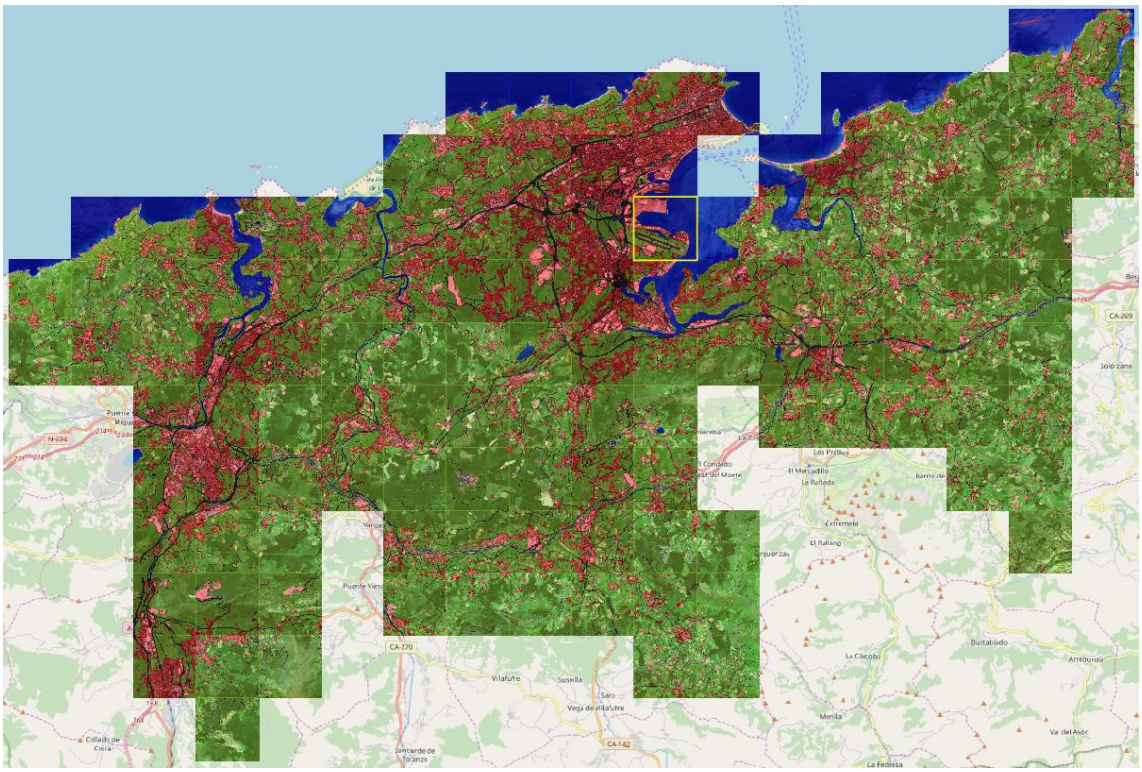
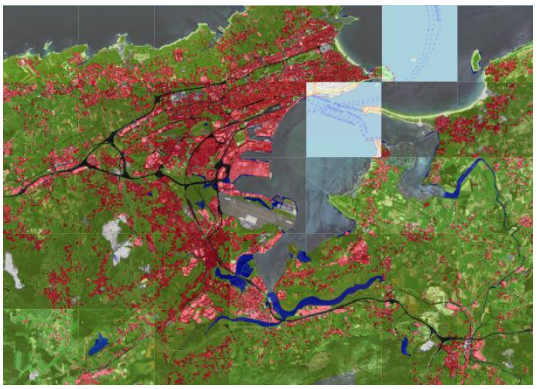


Figura 72 Predicción del mosaico de Santander

En la predicción se sigue corroborando todo lo comentado con las imágenes anteriores, por lo que podemos concluir que el modelo generaliza ya que funciona de forma aceptable en ciudades tan diversas de diferentes zonas geográficas. Al igual que con Pamplona, ampliamos en el centro urbano de Santander para apreciar mejor los detalles.

(a) Imagen Sentinel troceada



(b) *Ground truth*

Figura 73 Santander y alrededores formado con 35 imágenes de 256x256

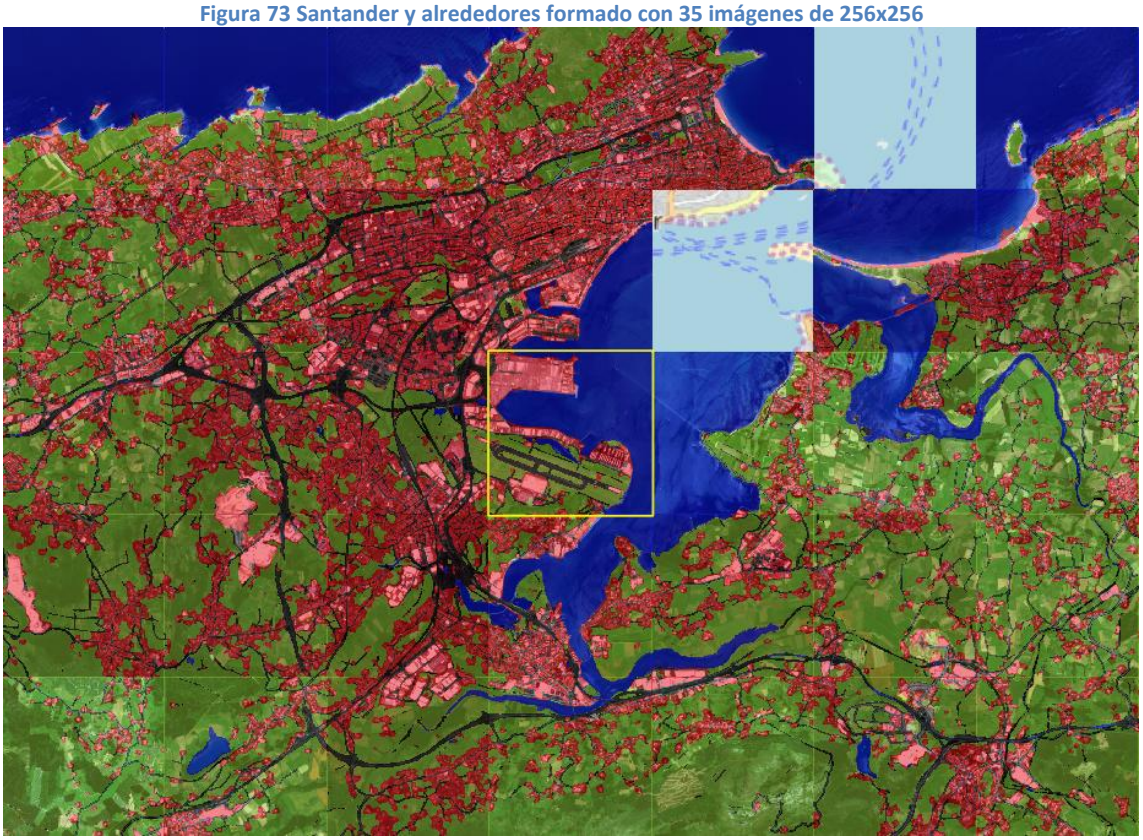


Figura 74 Predicción de Santander y alrededores

Al igual que en Alicante el modelo detecta perfectamente el mar a pesar de no estar incluido en la máscara. Los aeropuertos, con la reclasificación del *dataset*, han dejado de estar clasificados como urbanos completamente, y ahora conseguimos que se diferencien las diferentes partes, las edificaciones, las grandes carreteras y el suelo. Para apreciarlo mejor mostramos dicha zona ampliada.

(a) (b)

Figura 75 Zona de estudio de Santander (a) y su *ground truth* (b)

Figura 76 Predicción de la zona de estudio de Santander

Para finalizar y como curiosidad, en nuestro *dataset* no tenemos barcos etiquetados, pero el modelo es capaz de interpretarlos como urbanos. En la predicción lo podemos observar en el muelle junto al aeropuerto y si nos fijamos bien, en la parte superior izquierda, una pequeña embarcación navegando también la localiza con un solo pixel.

6 Conclusiones

En este proyecto hemos desarrollado un sistema para el etiquetado automático de *datasets* satelitales aprovechando los servicios y datos de *Copernicus*, en concreto el Urban Atlas. Además, hemos utilizado las etiquetas obtenidas para entrenar redes neuronales convolucionales y estudiar sus capacidades para la teledetección aplicando segmentación semántica. Como conclusiones principales podemos destacar los siguientes puntos:

1. Infrautilización de EOLearn

EOLearn es una colección de paquetes Python que permiten acceder y procesar sin problemas las secuencias de imágenes espacio-temporales adquiridas por satélites de manera oportuna y automática. El problema que nos hemos encontrado se debe a que sus principales utilidades para rellenar los datos satelitales del EOPatch se basan en servicios web WMS (*Web Map Service*) y WCS (*Web Coverage Service*) para obtener y procesar las imágenes. Estas utilidades están vinculadas al uso de una cuenta en Sentinel Hub (administrado por Sinergise) la cual está sujeta a ciertos costes (https://www.sentinel-hub.com/pricing-plans). EOLearn también permite el acceso a los datos satelitales mediante los servicios de almacenamiento de Amazon S3 pero, en ese caso Amazon también cobra a los usuarios según la cantidad de peticiones y datos descargados (https://aws.amazon.com/es/s3/pricing/).

Para familiarizarnos con el tratamiento de datos satelitales, mediante los paquetes sentinelsat, rasterio y GDAL nos las hemos arreglado para completar el EOPatch con los datos necesarios y así evitar el uso de estos servicios de pago, pero para ello necesitamos tres EOTask (DownloadSciHub, StackBands y Puzzle). En el caso concreto de SIWUAC, son las EOTask que hacen que se retrase el tiempo de ejecución del EOWorkflow y si fuera necesario podrían ser sustituidas por uno de los servicios de datos de Sinergise o Amazon.

Por otra parte no hemos aprovechado EOLearn al máximo ya que no hemos empleado sus propiedades multi-temporales y el estudio en con redes neuronales lo hemos realizado fuera de este *framework*.

2. Problemas de Urban Atlas

El principal problema que podemos achacar a Urban Atlas es que para crear máscaras de clasificación para diferenciar zonas urbanas de suelo libre de edificaciones, carreteras y agua, sus clases son demasiado burdas. Por ejemplo, las clases como la 14200 (Instalaciones deportivas y de ocio) o la 12400 (Aeropuertos) las hemos tenido que eliminar de nuestras máscaras debido a que no diferenciaban lo que nosotros estábamos interesados en clasificar.

Por otra parte otro de los problemas presentes, aunque no tan grave, ha sido la diferencia temporal entre los datos de las máscaras y los datos de las imágenes Sentinel. Urban Atlas se publicó en 2012 y los servicios de imágenes de Sentinel 2 no estuvieron operativos hasta 2016. Esto provoca que, por ejemplo en la clase de Urban Atlas Terrenos sin uso actual (CODE2012=13400) clasificada como suelo, pueden haber construido edificios entre 2012 y 2016 y lo hemos clasificado como urbano, pero esta revisión requeriría elevado tiempo para abordarlo en este proyecto.

3. Lecciones aprendidas con SIWUAC

Personalmente, este proyecto ha sido mi primera toma de contacto con los datos espaciales. Por este motivo, los principales conocimientos adquiridos han ido en la línea de la geolocalización de los datos. Durante la implementación hemos trabajado con datos representados en varios sistemas de coordenadas. Por un lado con los datos vectoriales hemos empleado WGS84 (EPSG 4326) para la solicitud de productos Sentinel con la API Sentinelsat y Urban Atlas viene en ETRS89 / LAEA (EPSG 3035). Ambos están definidos en términos de longitud y latitud pero con diferente datum. Por otro lado las imágenes Sentinel2, están georeferenciadas en UTM WGS84 el cual consta de un código EPSG en función de la zona para minimizar el error cometido con la proyección y se define en metros.

Los *datasets* finales, al ser creados a partir de datos vectoriales en ETRS89 LAEA que se proyectan al sistema UTM WGS84 de las imágenes Sentinel para formar las máscaras, son mosaicos georeferenciados en UTM con el correspondiente EPSG de su zona sus metadatos.

El tratamiento de datos espaciales (en nuestro caso satelitales) es una disciplina muy interesante, que actualmente está en auge, no solo por la cantidad de datos generados diariamente (satélites, drones, dispositivos GPS,…) sino por la gran utilidad de estos. Por este motivo y a pesar de existir multitud de herramientas para el geoprocesamiento de datos, cada día aparecen nuevas o se actualizan las ya existentes.

4. Desafíos de la implementación de SegNet

Las principales dificultades en la implementación de SegNet adaptada, no han estado en la implementación de la propia arquitectura *encoder-decoder* debido a que Keras es muy intuitivo para crear modelos redes neuronales. Para nosotros, la mayor dificultad ha estado en las modificaciones de las funciones de coste (*loss*) para aplicar los diferentes métodos de balanceo de clases y comprobar su correcto funcionamiento. De la misma forma la adaptación de las métricas teniendo en cuenta que las máscaras tienen zonas que no debemos evaluar su predicción.

5. Conclusiones finales

En los problemas de machine learning, hemos comprobado la importancia de invertir tiempo en el tratamiento del conjunto de datos en cuestión sobre el que se vaya a trabajar. En nuestro caso imágenes satelitales con máscaras de segmentación semántica creadas nosotros mismos. Además del tamaño de las imágenes, el porcentaje de píxeles clasificados, los grados de solapamiento espacial entre imágenes y el número de bandas de las imágenes entre otros, son parámetros que hubiésemos podido estudiar. Por suerte comenzando con el tamaño y una segunda reclasificación de las máscaras del *dataset* junto al estudio de los diferentes métodos de balanceo de clases han sido las claves para obtener mejoras significativas en los resultados respecto a las primeras pruebas realizadas empleando el mismo modelo.

En cuanto al rendimiento visual de los resultados, hemos decidido finalizar con lo expuesto en la sección anterior ya que cumple con las expectativas iniciales del proyecto. Hemos entrado y profundizado en el campo de los datos espaciales, en concreto los satelitales y además hemos

conseguido aplicar un modelo de segmentación semántica de vanguardia haciendo uso de la fuente de datos que es *Copernicus*.

6.1 Líneas futuras

Para finalizar, a continuación enumeramos las principales líneas de trabajo que abre este proyecto. Siguiendo la estructura general del proyecto, las primeras son referentes a SIWUAC y las siguientes orientadas a mejorar los modelos de segmentación semántica empleados.

1. Generalización de SIWUAC y mejora de máscaras

Urban Atlas nos limita la superficie que podemos cubrir con imágenes para entrenar modelos. Modificando únicamente la EOTask UrbanAtlasReclassify para que sea capaz de procesar y reclasificar no solo el *shapefile* de una ciudad de Urban Atlas, sino cualquier *shapefile* y combinarlos entre ellos, el EOWorkflow sería capaz de crear máscaras con más información extraída de otros *shapefiles*. Por ejemplo, Open Street Map ofrece información en el mismo formato que Urban Atlas pero con una clasificación muchísimo más detallada, dividida en capas temáticas con cobertura mundial y actualizada casi diariamente.

Siguiendo esta línea podríamos obtener máscaras de mayor calidad, precisión y variedad de clases. Además esto es sencillo de integrar en el *framework* desarrollado gracias a la filosofía de *workflow* ya que solo requiere generalizar la EOTask UrbanAtlasReclassify.

2. Mejora de los datos

Las imágenes Sentinel, tienen ciertas limitaciones, por ejemplo en cuanto a su resolución solo 4 de las 13 bandas que contienen son de 10m/px. Estas bandas son las que estamos empleando, pero podríamos incluir más aplicando enfoques de super-resolución. Por un lado podríamos simplemente redimensionar las bandas de menor resolución para emplearlas en nuestros *datasets* como bandas adicionales a las cuatro que ya tienen las imágenes, pero también podríamos utilizar herramientas como DSen2 [37]. Publicada en 2018, emplea modelos basados en CNN que super-resolucionan las bandas de menor resolución (20m/px y 60m/px) aprovechando las texturas presentes en las bandas de mayor resolución. Esta herramienta permite obtener las 13 bandas a 10m/px, por lo que podríamos llegar a emplear imágenes de 13 bandas. Lógicamente, luego quedaría evaluar la utilidad de cada una de estas bandas para su uso en la segmentación.

Otro enfoque de super-resolución, es aumentar la resolución de los datos de 10m Sentinel 2 y pasarlos a 5m. Para ello serían necesarias imágenes de muy alta resolución, que normalmente tienen costes elevados ya que suelen ser obtenidas por vuelos aéreos o satélites comerciales.

Por último, también se podría estudiar el rendimiento de los modelos incluyendo información obtenida por la instrumentación SAR de Sentinel-1 y adaptar SIWUAC para que no solamente pueda descargar datos Sentinel-1, sino incluirlos como bandas en las imágenes de los *datasets* generados.

3. Otras arquitecturas diferentes

Los resultados han mostrado que el *Deep learning* está más que capacitado para procesar datos satelitales pero las clasificaciones obtenidas no son todavía de excesiva calidad. Mejorando los datos y después de validarlos, nos quedaría considerar modificaciones de la arquitectura SegNet así como probar el rendimiento de otros modelos como DeepLab.

4. Modelo para generar segmentaciones de mayor resolución

Como hemos explicado en el punto 6 de la Sección 3.2 (EOTask Rasterize), gracias a SIWUAC y al detalle de los datos de Urban Atlas podemos generar máscaras de más resolución que las imágenes Sentinel empleando el *parámetro de inicialización* factor de máscara (FM). Por ejemplo, con este parámetro fijado a 2, las imágenes de entrenamiento de 1024x1024, tendrían máscaras de clasificación de 2048x2048. En este caso SegNet deberá predecir las etiquetas de 4 píxeles a partir de un único pixel de Sentinel. Esto permitiría obtener predicciones a 5m/px a partir de imágenes a 10m/px lo que podría ayudar a definir bordes o formas como las carreteras de mayor calidad.

5. Aplicación: EOWorkflow para segmentar imágenes en tiempo real

Con un modelo que sea capaz de clasificar imágenes Sentinel-2 de un tamaño determinado, el modelo puede ser incrustado en una EOTask y construir un EOWorkflow que monitorice zonas específicas. El flujo consistiría en descargar la última imagen Sentinel de una zona concreta, trocearla en imágenes del tamaño aceptado por el modelo entrenado y utilizar el modelo para obtener su segmentación en tiempo real. Una herramienta de este tipo puede ser el punto de partida de aplicaciones de detección de cambios o monitoreo de la expansión de las ciudades.

7 Bibliografía

[1] M. Bosch, G. A. Christie, and C. M. Gifford, “Sensor Adaptation for Improved Semantic Segmentation of Overhead Imagery.,” *arXiv Comput. Vis. Pattern Recognit.*

[2] T. Pohlen, A. Hermans, M. Mathias, and B. Leibe, “Full-resolution residual networks for semantic segmentation in street scenes,” *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*, vol. 2017–Janua, pp. 3309–3318, 2017.

[3] G. Lin, C. Shen, A. van dan Hengel, and I. Reid, “Efficient piecewise training of deep structured models for semantic segmentation,” 2015.

[4] L. C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, “Encoder-decoder with atrous separable convolution for semantic image segmentation,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 11211 LNCS, pp. 833–851, 2018.

[5] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, “The pascal visual object classes (VOC) challenge,” *Int. J. Comput. Vis.*, vol. 88, no. 2, pp. 303–338, 2010.

[6] T. Y. Lin *et al.*, “Microsoft COCO: Common objects in context,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2014.

[7] V. Badrinarayanan, A. Kendall, R. Cipolla, and S. Member, “SegNet : A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation,” pp. 1–14.

[8] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, “Rethinking Atrous Convolution for Semantic Image Segmentation,” 2017.

[9] F. Rosenblatt, “The perceptron: A probabilistic model for information storage and organization in the brain,” *Psychol. Rev.*, 1958.

[10] M. Minsky and S. Papert, “Perceptrons: expanded edition,” *MIT Press Cambridge MA*, vol. 522, p. 20, 1988.

[11] P. Werbos, “Beyond regression: New tools for prediction and analysis in the behavioral sciences,” *PhD.Thesis, Harvard Univ.*, 1974.

[12] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning internal representations by error propagation (No. ICS-8506).,” *Calif. Univ San Diego La Jolla Inst Cogn. Sci.*, vol. 1, pp. 318–362, 1986.

[13] K. Fukushima, “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position,” *Biol. Cybern.*, 1980.

[14] D. C. Cireşan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, “Flexible, high performance convolutional neural networks for image classification,” in *IJCAI International Joint Conference on Artificial Intelligence*, 2011.

[15] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proc. IEEE*, 1998.

[16] O. Russakovsky *et al.*, “ImageNet Large Scale Visual Recognition Challenge,” *Int. J. Comput. Vis.*, 2015.

[17] A. Krizhevsky and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” *Neural Inf. Process. Syst.*, 2012.

[18] M. Andriluka, L. Pishchulin, P. Gehler, and B. Schiele, “2D human pose estimation: New benchmark and state of the art analysis,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 3686–3693, 2014.

[19] E. Romera, J. M. Alvarez, L. M. Bergasa, and R. Arroyo, “Erfnet: Efficient residual factorized convnet for real-time semantic segmentation,” *Tits*, pp. 1–10, 2018.

[20] M. Siam, M. Gamal, M. Abdel-Razek, S. Yogamani, and M. Jagersand, “RTSeg: Real-time Semantic Segmentation Comparative Study,” no. 1, 2018.

[21] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello, “ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation,” pp. 1–10, 2016.

[22] E. Shelhamer, J. Long, and T. Darrell, “Fully Convolutional Networks for Semantic Segmentation,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 4, pp. 640–651, 2017.

[23] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9351, pp. 234–241, 2015.

[24] J. Yang, B. Price, S. Cohen, H. Lee, and M.-H. Yang, “Object Contour Detection with a Fully Convolutional Encoder-Decoder Network,” 2016.

[25] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, “Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs, Atrous Convolution, and Fully Connected CRFs,” *Iclr*, 2014.

[26] S. Zheng *et al.*, “Conditional random fields as recurrent neural networks,” *Proc. IEEE Int. Conf. Comput. Vis.*, vol. 2015 Inter, no. 6, pp. 1529–1537, 2015.

[27] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, “DeepLab,” *arXiv*, 2016.

[28] H. Kaiming, Z. Xiangyu, R. Shaoqing, and S. Jian, “Deep Residual Learning for Image Recognition,” *arXiv Comput. Vis. Pattern Recognit.*, 2015.

[29] G. Lin, A. Milan, C. Shen, and I. Reid, “RefineNet: Multi-path refinement networks for high-resolution semantic segmentation,” *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*, vol. 2017–Janua, pp. 5168–5177, 2017.

[30] G. Ghiasi and C. C. Fowlkes, “Laplacian pyramid reconstruction and refinement for semantic segmentation,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9907 LNCS, pp. 519–534, 2016.

[31] GoogleResearch, “TensorFlow: Large-scale machine learning on heterogeneous systems,” *Google Res.*, 2015.

[32] M. Cordts *et al.*, “The Cityscapes Dataset,” in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2016.

[33] G. J. Brostow, J. Fauqueur, and R. Cipolla, “Semantic object classes in video: A high-definition ground truth database,” *Pattern Recognit. Lett.*, 2009.

[34] S. Song, S. P. Lichtenberg, and J. Xiao, “SUN RGB-D: A RGB-D scene understanding benchmark suite,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2015.

[35] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv Prepr. arXiv1409.1556*, 2014.

[36] D. Eigen and R. Fergus, “Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2015.

[37] C. Lanaras, J. Bioucas-Dias, S. Galliani, E. Baltsavias, and K. Schindler, “Super-resolution of Sentinel-2 images: Learning a globally applicable deep neural network,” *ISPRS J. Photogramm. Remote Sens.*, vol. 146, pp. 305–319, 2018.

Anexo I - Tabla de figuras

FIGURA 1 IMAGEN SÉNTINEL-2 DEL PUERTO DE ALICANTE, JUNTO A UNA POSIBLE CLASIFICACIÓN GENERADA CON SIWUAC

FIGURA 2 LOGO TRACASA INSTRUMENTAL

FIGURA 3 VISOR WEB DEL URBAN ATLAS 2012 DE NAVARRA
[https://land.copernicus.eu/local/urban-atlas/urban-atlas-2012]

FIGURA 4 LOGO DEL PROGRAMA *COPERNICUS*
[https://insitu.copernicus.eu]

FIGURA 5 PILARES FUNDAMENTALES SOBRE LOS QUE SE SUSTENTA EL PROGRAMA *COPERNICUS*
[https://www.copernicus.eu/sites/default/files/documents/Copernicus_brochure_EN_web_Oct2017.pdf]

FIGURA 6 ICONOS DE LOS SERVICIOS DE *COPERNICUS*
[https://www.nceo.ac.uk/innovation/copernicus-relay]

FIGURA 7 INTERFAZ DE QGIS

FIGURA 8 REPRESENTACIÓN GEOGRÁFICA (A) Y PROYECTADA (B)
[http://resources.arcgis.com/es/help/getting-started/articles/026n0000000s000000.htm][https://es.wikipedia.org/wiki/Sistema_de_coordenadas_universal_transversal_de_Mercator]
[https://sentinels.copernicus.eu/web/sentinel/missions/sentinel-2/data-products]

FIGURA 9 SISTEMA DE COORDENADAS ESPACIALES Y COORDENADAS DE PÍXEL
[https://www.perrygeo.com/python-affine-transforms.html]

FIGURA 10 ESTRUCTURA DE CARPETAS DEL PRODUCTO SENTINEL-2
[https://sentinel.esa.int/web/sentinel/user-guides/sentinel-2-msi/data-formats]

FIGURA 11 ESCENA REPRESENTADA EN FORMATO VECTORIAL Y RÁSTER
[http://desktop.arcgis.com/es/arcmap/10.3/manage-data/raster-and-images/how-features-are-represented-in-a-raster.htm]

FIGURA 12 GEOMETRÍAS BÁSICAS DE LOS DATOS VECTORIALES
[https://docs.qgis.org/2.14/es/docs/gentle_gis_introduction/vector_data.html]

FIGURA 13 URBAN ATLAS DE ÁMSTERDAM SOBRE IMAGEN RGB DE SENTINEL-2 DE LOS PAÍSES BAJOS

FIGURA 14 CONTEXTUALIZACIÓN DEL *DEEP LEARNING*

FIGURA 15 ANALOGÍA ENTRE NEURONA BIOLÓGICA Y ARTIFICIAL

FIGURA 16 FUNCIONES DE ACTIVACIÓN

FIGURA 17 PERCEPTRÓN

FIGURA 18 ARQUITECTURA GENÉRICA POR CAPAS DE LAS REDES NEURONALES

FIGURA 19 TOPOLOGÍAS DE REDES NEURONALES
[https://www.asimovinstitute.org/author/fjodorvanveen/]

FIGURA 20 EXTRACCIÓN JERARQUIZADA DE CARACTERÍSTICAS EN CNNS

FIGURA 21 EJEMPLO DE FILTROS CONVOLUCIONALES PARA DETECCIÓN DE BORDES

FIGURA 22 EJEMPLO DE *MAX-POOLING*

FIGURA 23 ARQUITECTURA DE LENET-5

FIGURA 24 ARQUITECTURA DE ALEXNET

FIGURA 25 CLASIFICACIÓN (A), LOCALIZACIÓN DE OBJETOS (B) Y SEGMENTACIÓN SEMÁNTICA (C) [18]

FIGURA 26 FCN APLICADA A SEGMENTACIÓN SEMÁNTICA
[https://medium.com/nanonets/how-to-do-image-segmentation-using-deep-learning-]

FIGURA 27 ARQUITECTURA CLÁSICA DE RED *ENCODER-DECODER* [24]

FIGURA 28 EJEMPLO DE FLUJO DE TAREAS DE EOLEARN
[https://github.com/sentinel-hub/eo-learn]

FIGURA 29 ENTRADA Y SALIDA GENERAL DE SIWUAC

FIGURA 30 TOPOLOGÍA DEL EOWORKFLOW DE SIWUAC

FIGURA 31 EN VERDE LA SUPERFICIE QUE CUBRE URBAN ATLAS EN ALICANTE, Y LA BBOX EXTRAIDA

FIGURA 32 PRODUCTOS SÉNTINEL PARA CUBRIR LA ZONA DELIMITADA POR EL URBAN ATLAS DE ALICANTE

FIGURA 33 IMAGEN GEOTIFF GENERADA A PARTIR DE LAS DOS IMÁGENES SENTINEL DESCARGADAS

FIGURA 34 URBAN ATLAS DE ALICANTE
FIGURA 35 SHAPEFILE CREADO MEDIANTE LA AGRUPACIÓN DE CLASES DE URBAN ATLAS
FIGURA 36 RESULTADO DE LA RASTERIZACIÓN
FIGURA 37 *SHAPEFILE* (A) E IMAGEN SENTINEL (B)
FIGURA 38 MÁSCARAS DE CLASIFICACIÓN CON DIFERENTES VALORES DE FACTOR DE MÁSCARA
FIGURA 39 MOSAICO DE ALICANTE FORMADO POR 9 IMÁGENES 1024X1024X4, CON EL 25% DE PÍXELES SOLAPADOS
FIGURA 40 MOSAICO DE ALICANTE CON DIFERENTES GRADOS DE COBERTURA DE CLASES (MCC)
FIGURA 41 ARQUITECTURA DE SEGNET
FIGURA 42 A, B, C, D CORRESPONDEN A LOS VALORES DEL MAPA DE CARACTERÍSTICAS QUE SEGNET-BASIC EMPLEA PARA EL UPSAMPLING DEL MAPA DE CARACTERÍSTICAS Y CONVOLUCIONARLO SEGUIDAMENTE
FIGURA 43 *UPSAMPLING* EN SEGNET-BASIC
FIGURA 44 AZUL: INPUT; VERDE: OUTPUT; CONVOLUCIÓN (A) Y "DECONVOLUCIÓN" (B)
[https://www.cyberailab.com/home/segnet-an-image-segmentation-neural-network]
FIGURA 45 *UPSAMPLING* MODIFICADO
FIGURA 46 PÍXELES POR CLASE EN CADA *DATASET*
FIGURA 47 VERDADEROS POSITIVOS, LOS FALSOS POSITIVOS Y LOS FALSOS NEGATIVOS
[https://medium.com/datadriveninvestor/deep-learning-in-medical-imaging-3c1008431aaf]
FIGURA 48 ZONA DE ALICANTE: *GROUND TRUTH* (A) Y PRIMERA PREDICCIÓN (B)
FIGURA 49 AEROPUERTO DE BARAJAS CON GRANDES CARRETERAS SIN DEFINIR EN LA MÁSCARA COMO
FIGURA 50 REPRESENTACIÓN GRÁFICA DE LAS MÉTRICAS
FIGURA 51 DESGLOSE DE MIOU
FIGURA 52 ZONAS DE ESTUDIO DEL CONJUNTO DE *TEST*
FIGURA 53 RESULTADOS CUALITATIVOS DE CADA UNO DE LOS MODELOS EN LAS ZONAS DE ESTUDIO SELECCIONADAS
FIGURA 54 MOSAICO DE ALICANTE (A) Y SU GROUND TRUTH (B)
FIGURA 55 PREDICCIÓN DEL MOSAICO DE ALICANTE
FIGURA 56 ZONA DE ESTUDIO DE ALICANTE (A) Y SU *GROUND TRUTH* (B)
FIGURA 57 PREDICCIÓN DE LA ZONA DE ESTUDIO DE ALICANTE
FIGURA 58 PREDICCIÓN DEL MOSAICO DE MURCIA
FIGURA 59 ZONA DE ESTUDIO DE MURCIA (A) Y SU *GROUND TRUTH* (B)
FIGURA 60 PREDICCIÓN DE LA ZONA DE ESTUDIO DE MURCIA
FIGURA 61 PREDICCIÓN DEL MOSAICO DE PAMPLONA
FIGURA 62 PAMPLONA Y ALREDEDORES FORMADO CON 24 IMÁGENES DE 256X256
FIGURA 63 PREDICCIÓN DE PAMPLONA Y ALREDEDORES
FIGURA 64 PRIMERA ZONA DE ESTUDIO DE PAMPLONA (A) Y SU *GROUND TRUTH* (B)
FIGURA 65 ZONA DE ESTUDIO DE PAMPLONA
FIGURA 66 IMAGEN SENTINEL DE LA ZONA MEDIA DE NAVARRA CON 10 PARCHES DE 256X256
FIGURA 67 *GROUND TRUTH* DE LA ZONA MEDIA DE NAVARRA
FIGURA 68 PREDICCIÓN DE LA ZONA MEDIA DE NAVARRA
FIGURA 69 ZONA DE ESTUDIO EXTRA EN LA ZONA MEDIA DE NAVARRA (A) Y SU *GROUND TRUTH* (B)
FIGURA 70 PREDICCIÓN DEL ÁREA DE ESTUDIO EXTRA EN LA ZONA MEDIA DE NAVARRA
FIGURA 71 MOSAICO DE SANTANDER (A) Y *GROUND TRUTH* (B)
FIGURA 72 PREDICCIÓN DEL MOSAICO DE SANTANDER
FIGURA 73 SANTANDER Y ALREDEDORES FORMADO CON 35 IMÁGENES DE 256X256
FIGURA 74 PREDICCIÓN DE SANTANDER Y ALREDEDORES
FIGURA 75 ZONA DE ESTUDIO DE SANTANDER (A) Y SU *GROUND TRUTH* (B)
FIGURA 76 PREDICCIÓN DE LA ZONA DE ESTUDIO DE SANTANDER