



Escuela Técnica Superior de Ingeniería Informática

TELEDETECCIÓN

PROCESAMIENTO DE IMÁGENES DE SATÉLITES

Trabajo Segundo Parcial

Francisco Javier Gómez Pulido
5º Doble grado Matemáticas - Ingeniería Informática

30 de Diciembre de 2024

Índice

1 Introducción	3
1.1 Objetivos del trabajo	3
1.2 Contexto y relevancia del procesamiento de imágenes RAW	3
1.3 Herramientas utilizadas	4
2 Selección de imágenes	5
2.1 Descripción del Sentinel-2	7
2.2 Proceso de selección de imágenes	7
2.3 Información técnica de la imagen seleccionada	9
3 Apertura y procesamiento de imágenes	10
3.1 Importación y apertura de imágenes	11
3.2 Visualización inicial de las bandas	11
3.3 Validación y verificación de los metadatos	14
4 Identificación y corrección de errores	16
4.1 Tipos de errores posibles en imágenes RAW	16
4.2 Localización de errores	16
4.3 Métodos aplicados para la corrección de errores	18
4.3.1 Corrección de valores nodata	18
4.3.2 Corrección de valores fuera del rango esperado	21
4.3.3 Corrección de errores del sensor	22
5 Corrección geográfica	24
5.1 Conceptos de corrección geográfica	24
5.2 Verificación de la georreferenciación	24
6 Aplicación de índices de vegetación	28
6.1 Selección de índices a utilizar	28
6.2 Implementación de los índices y análisis de los resultados	29
6.2.1 RVI (Ratio Vegetation Index)	29
6.2.2 NDVI (Normalized Difference Vegetation Index)	31
6.2.3 SAVI (Soil Adjusted Vegetation Index)	33
6.2.4 TVI (Transformed Vegetation Index)	34
6.2.5 BAI (Burned Area Index)	36
7 Transformaciones	38
7.1 Histogramas	38
7.2 Filtros	47
7.3 Escalado	51
7.4 Clasificación	53
8 Conclusiones	57
Bibliografía	58

Índice de figuras

1	Primera imagen seleccionada	5
2	Segunda imagen seleccionada	6
3	Información de la primera imagen seleccionada	8
4	Información de la segunda imagen seleccionada	8
5	Estructura de directorios	10
6	Visualización inicial de las bandas en escala de grises	12
7	Visualización de la imagen en RGB	14
8	Análisis de los valores de la banda B04	18
9	Resultado tras corregir valores nodata	20
10	Resultado tras corregir valores fuera de rango	22
11	Resultado tras corregir errores del sensor	23
12	Corrección geográfica de la banda 1	26
13	Corrección geográfica de la banda 6	27
14	RVI para la primera imagen	30
15	RVI para la segunda imagen	30
16	NDVI para la primera imagen	32
17	NDVI para la segunda imagen	32
18	SAVI para la primera imagen	33
19	SAVI para la segunda imagen	34
20	TVI para la primera imagen	35
21	TVI para la segunda imagen	35
22	BAI para la primera imagen	36
23	BAI para la segunda imagen	37
24	Histograma	39
25	Histograma tras la expansión lineal	40
26	Expansión lineal	41
27	Histograma tras el corte de colas	43
28	Corte de colas	44
29	Histograma tras la ecualización del histograma	45
30	Ecualización del histograma	46
31	Filtro de suavizado	48
32	Filtro de paso de alta	49
33	Filtro de realce	49
34	Filtro sobel	50
35	Máscara de luces	52
36	Máscara de nubes	53
37	Clasificación de la imagen RGB	56

1. Introducción

En esta primera sección, hablaremos de los principales objetivos de este trabajo, el cual sustituye al examen del segundo parcial de la asignatura **Tele-detección**, presentaremos el tema del procesamiento de imágenes satelitales en formato RAW y mencionaremos las principales herramientas que usaremos en el trabajo.

1.1. Objetivos del trabajo

El principal objetivo del trabajo es hacer un **procesamiento de imágenes satelitales RAW**, obtenidas de *Copernicus Data Space Ecosystem* [5]. Esta plataforma nos permite acceder a imágenes de satélites de alta calidad.

Sin embargo, estas imágenes no están procesadas, por lo que nuestro objetivo será llevar a cabo un procesamiento, incluyendo detección de errores, aplicación de correcciones geográficas, aplicación de índices de vegetación... y más cosas que iremos viendo durante el trabajo.

1.2. Contexto y relevancia del procesamiento de imágenes RAW

Hablemos en esta sección sobre la importancia del procesamiento de imágenes RAW. Pero, en primer lugar, ¿qué es una imagen RAW? En fotografía digital, una imagen se suele procesar y comprimir antes de almacenarse en la tarjeta de memoria. No obstante, las cámaras también pueden almacenar una imagen sin procesarla o comprimirla, como **archivo RAW** [1].

Las principales **ventajas** de almacenar una imagen sin procesar en RAW son las siguientes: [2]

- Muchos más matices de color.
- Mayor rango dinámico y gama de colores.
- Mayor control y potencial de ajuste.
- Puede ajustar el espacio de color después de capturar la imagen.
- Mayor potencial de nitidez.
- Se puede utilizar para convertir a otros formatos RAW.
- Prueba de propiedad y autenticidad.

Las imágenes RAW son esenciales porque contienen la información en su estado más puro, sin transformaciones que puedan afectar su calidad o precisión. Pero, como veremos en este trabajo, esta falta de procesamiento implica que

estas imágenes pueden tener **errores** técnicos, como distorsiones. Por tanto, el procesamiento de estas imágenes es muy importante para garantizar su utilidad en análisis de teledetección y aplicaciones prácticas.

Por otra parte, el análisis de imágenes satelitales tiene una importancia particular en áreas como: [3]

- **Monitorización medioambiental:** Seguimiento de cambios en la deforestación y evaluación de la salud de los ecosistemas.
- **Agricultura:** Identificación de cultivos y evaluación de su estado a través de índices de vegetación como el NDVI.
- **Gestión de desastres naturales:** Evaluación del impacto de inundaciones, incendios o deslizamientos de tierra, permitiendo una respuesta rápida y efectiva.
- **Seguimiento de especies en peligro de extinción:** Mediante imágenes satelitales, podemos seguir los movimientos migratorios y proteger las especies en peligro de extinción de la caza furtiva, por ejemplo [17].
- **Hallazgo de restos arqueológicos:** Podemos descubrir restos de hace miles de años en áreas inaccesibles para los humanos mediante imágenes desde el espacio [16].

1.3. Herramientas utilizadas

A recomendación del profesor, para llevar a cabo este trabajo se intentó utilizar primero **JupyterLab** como entorno principal de desarrollo. Es una plataforma que nos permite combinar código, texto explicativo y visualizaciones en un único documento [18]. Sin embargo, no funcionaba correctamente e iba muy lento, por lo que finalmente se optó por **Google Colab**.

Usaremos **Python** como lenguaje de programación, puesto que es el que hemos estado usando a lo largo del curso y estamos muy familiarizados con él, debido a su sencillez de sintaxis y amplia versatilidad para el análisis de imágenes. Principalmente, usaremos las librerías **NumPy**, **Pandas**, **Matplotlib** y **Rasterio**.

2. Selección de imágenes

Para este trabajo, y con el fin de realizar comparaciones más adelante, escojeremos dos imágenes distintas, ambas del famoso satélite **Sentinel-2**. Este satélite fue desarrollado por la Agencia Espacial Europea, y es parte del programa Copernicus. Proporciona imágenes de alta resolución en varias bandas espectrales, fundamentales para el análisis medioambiental [8].

La primera imagen seleccionada nos muestra la costa de Grecia, destacando la ciudad de Atenas, capital de Grecia, en la parte inferior, y el mar Egeo visible a la derecha. La elección de esta imagen se debe a que podemos encontrar mar, tierra y nubes en la misma imagen, por lo que puede ser interesante para nuestro análisis.



Figura 1: Primera imagen seleccionada

La segunda imagen la seleccionamos de forma que sea relativamente distinta a esta. Hemos escogido una de la costa italiana, con el fin de analizar posteriormente los índices de vegetación de ambas imágenes:

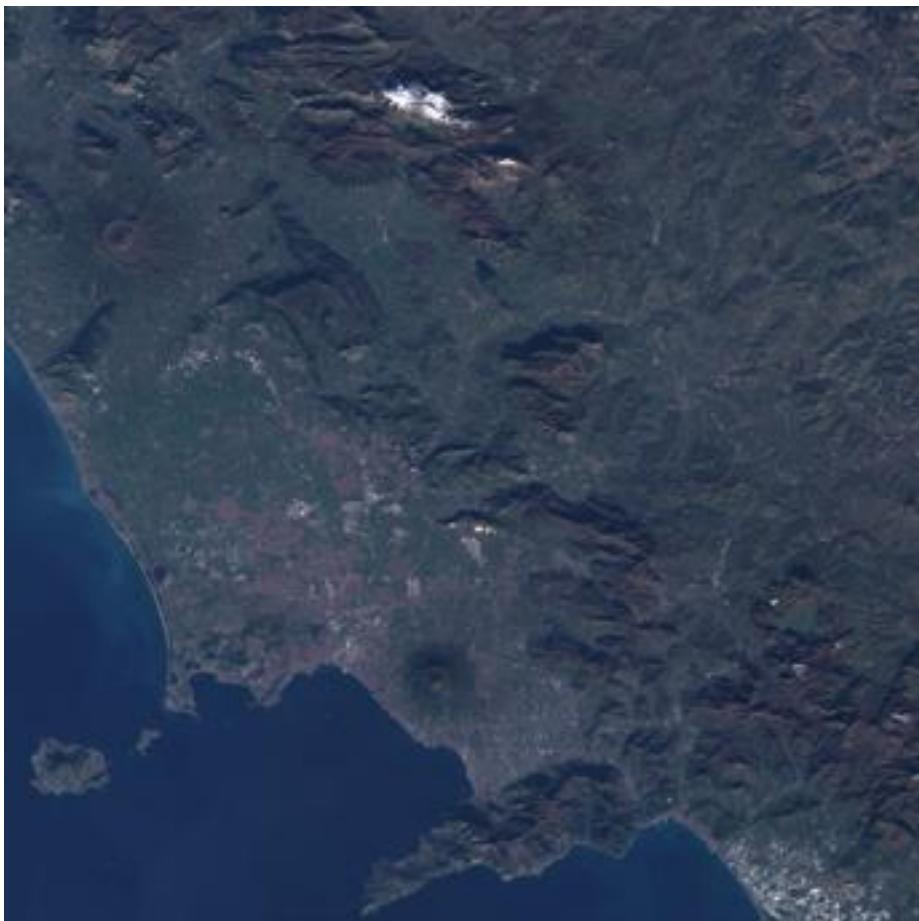


Figura 2: Segunda imagen seleccionada

Cuando descargamos una imagen de la web de Copernicus, además de la propia imagen, obtenemos los siguientes archivos y carpetas, que son relevantes para nuestro estudio:

- **MTD_MSIL2A.xml**: Archivo de metadatos principal.
- Carpeta **GRANULE**: Contiene subcarpetas con los datos de cada *granule* (unidad de imagen) procesados.
- Carpeta **IMG_DATA**: Contiene las imágenes de las distintas bandas especiales. Cada archivo está asociado a una resolución espacial: R10m, R20m, R60m.

- Carpeta HTML: Contiene un archivo `preview.html`, y otros, para visualizar una vista previa de la imagen y los metadatos.
- Carpeta `QI_DATA`: Contiene mapas de calidad y datos relacionados con condiciones atmosféricas (nubes, sombras...).

En los siguientes apartados, describiremos las características técnicas del satélite y la imagen seleccionada, además de los pasos necesarios para abrir la imagen y comenzar con su procesamiento.

2.1. Descripción del Sentinel-2

Brevemente, la información técnica del satélite seleccionado es la siguiente:

Característica	Detalle
Aplicaciones	Meteorológicas Monitoreo de la vegetación Calidad del agua Uso del suelo Áreas urbanas
Tipo de órbita	Polar heliosíncrona
Altura de la órbita	786 km
Resolución espacial	10 m para bandas visibles 20 m para bandas de infrarrojo cercano 60 m para bandas atmosféricas
Resolución temporal	Cada 5 días (para cada satélite) en una ubicación dada, con un intervalo de 10 días entre los dos satélites en órbita
Resolución radiométrica	12 bits
Resolución espectral	12 bandas espectrales: - 5 en el visible - 6 en el infrarrojo cercano y de onda corta - 2 en el infrarrojo - 1 en el blanco (B1) - 1 en el UV (B9)
Sensores	1 sensor principal: MSI, 12 bandas, barrido
Lanzamiento	23 de junio de 2015

Cuadro 1: Características técnicas del satélite Sentinel-2 [9]

2.2. Proceso de selección de imágenes

La selección de la imagen para este trabajo la realizamos a través de la plataforma **Copernicus Browser**, que proporciona acceso a las imágenes del satélite Sentinel-2, entre otros muchos. El proceso consiste en los siguientes pasos:

1. En primer lugar, seleccionamos el satélite y la zona del mapa de la que queremos la imagen satelital.
2. La página nos muestra el histórico de imágenes del satélite seleccionado en la zona seleccionada, por lo que elegimos también las últimas fotografías que nos muestra, una de las más recientes.
3. Comprobamos estas imágenes y seleccionamos la que mejor nos venga, teniendo en cuenta que no nos conviene que aparezcan muchas nubes en nuestra imagen, por ejemplo.
4. Finalmente, y una vez hemos tenido en cuenta todos estos pasos, descargamos el archivo, lo cual nos llevará algo de tiempo debido al enorme tamaño.

PRODUCT INFO

ATTRIBUTES

Summary

Name: S2A_MSIL1C_20241125T091321_N0511_R050_T34SGH_20241125T100022.SAFE
Size: 760MB
Sensing time: 2024-11-25T09:13:21.024000Z
Platform short name: SENTINEL-2
Instrument short name: MSI

Product

Instrument

Platform

Instrument short name: MSI
Platform short name: SENTINEL-2
Satellite platform: A

Download single files

PREVIEW

FOOTPRINT

Product id: [https://zipper.dataspace.copernicus.eu/odata/v1/Products\(b621109b-fee5-46aa-a0a1-bbd02c8337\)/\\$value](https://zipper.dataspace.copernicus.eu/odata/v1/Products(b621109b-fee5-46aa-a0a1-bbd02c8337)/$value)

Figura 3: Información de la primera imagen seleccionada

PRODUCT INFO

ATTRIBUTES

Summary

Name: S2C_MSIL1C_20241217T095451_N9905_R079_T33TVF_20241217T115452.SAFE
Size: 778MB
Sensing time: 2024-12-17T09:54:51.025000Z
Platform short name: SENTINEL-2
Instrument short name: MSI

Product

Instrument

Platform

Instrument short name: MSI
Platform short name: SENTINEL-2
Satellite platform: C

Download single files

PREVIEW

FOOTPRINT

Product id: [https://zipper.dataspace.copernicus.eu/odata/v1/Products\(e2c2e64a-2845-400b-b38d-293d3a63b2ff\)/\\$value](https://zipper.dataspace.copernicus.eu/odata/v1/Products(e2c2e64a-2845-400b-b38d-293d3a63b2ff)/$value)

Figura 4: Información de la segunda imagen seleccionada

2.3. Información técnica de la imagen seleccionada

La principal información técnica de nuestra primera imagen seleccionada para el trabajo es la siguiente:

- Número de la órbita absoluta: 49239
- Modo de adquisición: INS-NOBS
- Nivel de procesamiento: S2MSI1C, lo que significa que ha sido procesada para corregir errores radiométricos y geográficos básicos, aunque aun podría contener errores.
- Fecha de adquisición: 11 de noviembre de 2024 a las 08:22.

Respecto a la segunda imagen, encontramos que la principal información técnica es la siguiente:

- Número de la órbita absoluta: 40717
- Modo de adquisición: INS-NOBS
- Nivel de procesamiento: S2MSI2A, algo más avanzado que el anterior.
- Fecha de adquisición: 17 de diciembre de 2024 a las 09:54.

Las imágenes elegidas pueden verse en: [6] y [7].

3. Apertura y procesamiento de imágenes

Empecemos en este punto con el código como tal. Como mencionamos anteriormente, utilizaremos Google Colab. En primer lugar, lo abrimos en un navegador, y creamos una estructura de directorios muy sencilla: dos carpetas, una para los datos y otra para los notebooks. Únicamente necesitaremos los archivos de la carpeta **IMG_DATA**, que son los que nos ofrecen la información de las bandas espectrales. Por tanto, el primer paso será subirlos. Para ubicarnos más adelante, nos quedará la siguiente estructura de directorios:

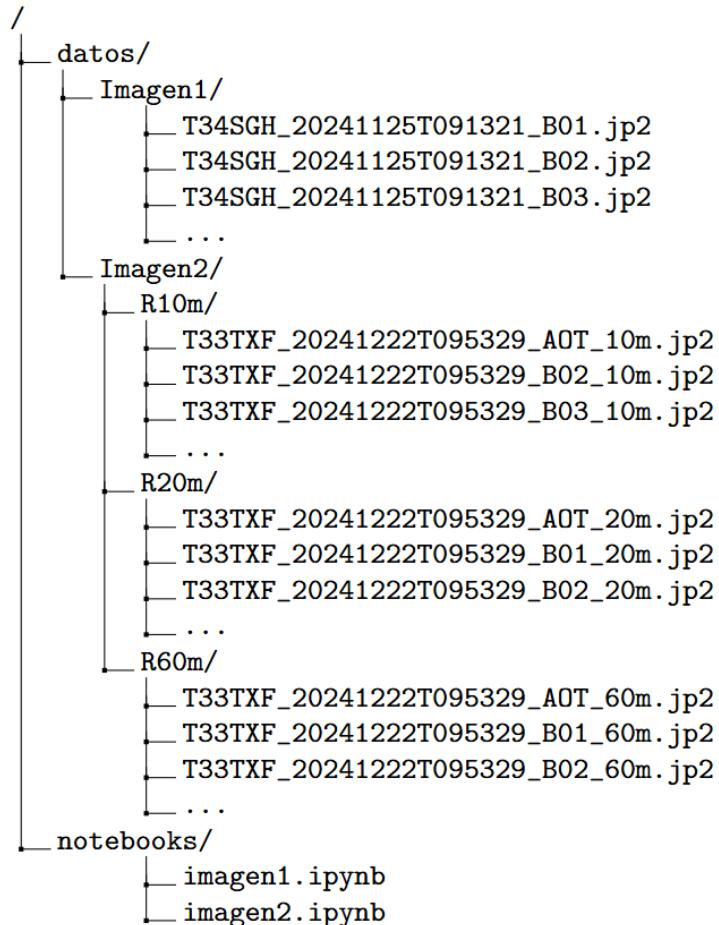


Figura 5: Estructura de directorios

Cabe destacar que únicamente incluiremos el código para la primera imagen, la segunda se hace de una forma muy similar.

3.1. Importación y apertura de imágenes

Una vez tenemos clara esta estructura, comenzamos importando las imágenes. Para ello, tenemos el siguiente código:

```
1 import rasterio
2 import matplotlib.pyplot as plt
3 import numpy as np
4 import os
5
6 path_imagen1 = "../datos/Imagen1/T34SGH_20241125T091321_"
7
8 B01 = rasterio.open(path_imagen1 + 'B01.jp2')
9 B02 = rasterio.open(path_imagen1 + 'B02.jp2')
10 B03 = rasterio.open(path_imagen1 + 'B03.jp2')
11 B04 = rasterio.open(path_imagen1 + 'B04.jp2')
12 B05 = rasterio.open(path_imagen1 + 'B05.jp2')
13 B06 = rasterio.open(path_imagen1 + 'B06.jp2')
14 B07 = rasterio.open(path_imagen1 + 'B07.jp2')
15 B08 = rasterio.open(path_imagen1 + 'B08.jp2')
16 B09 = rasterio.open(path_imagen1 + 'B09.jp2')
17 B10 = rasterio.open(path_imagen1 + 'B10.jp2')
18 B11 = rasterio.open(path_imagen1 + 'B11.jp2')
19 B12 = rasterio.open(path_imagen1 + 'B12.jp2')
```

3.2. Visualización inicial de las bandas

Pasemos ahora a visualizar cada uno de estos canales obtenidos:

```
1 bandas = {
2     "B01": B01,
3     "B02": B02,
4     "B03": B03,
5     "B04": B04,
6     "B05": B05,
7     "B06": B06,
8     "B07": B07,
9     "B08": B08,
10    "B09": B09,
11    "B10": B10,
12    "B11": B11,
13    "B12": B12
14 }
15
16 fig, axes = plt.subplots(3, 4, figsize=(16, 12))
17
18 for i, (nombre_banda, banda) in enumerate(bandas.items()):
19     ax = axes[i // 4, i % 4]
20     ax.imshow(banda.read(1), cmap='gray') # Mostrar la banda en escala de grises
21     ax.set_title(f"{nombre_banda}")
```

```

22     ax.axis('off')
23
24 plt.tight_layout()
25 plt.show()

```

Una vez ejecutado este código, obtenemos el siguiente resultado:

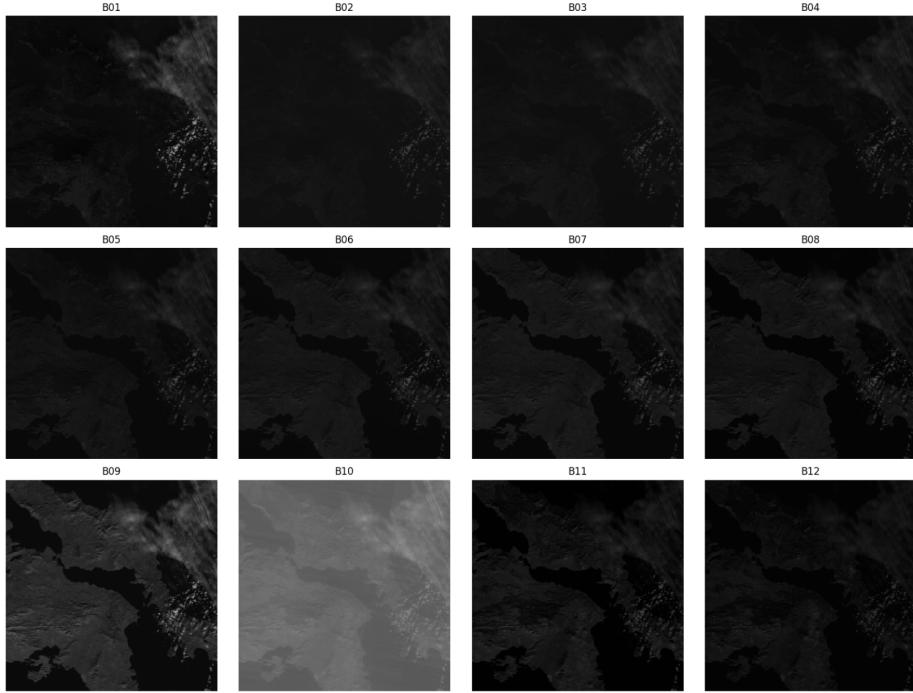


Figura 6: Visualización inicial de las bandas en escala de grises

Cada satélite tiene unas bandas en específico. En nuestro caso, el Sentinel-2 tiene las siguientes bandas: [15]

- **B01:** Banda de ultra violeta (correspondiente a 443 nm). Utilizada para el estudio de la atmósfera y la corrección de la imagen en función de la atmósfera.
- **B02:** Banda azul (correspondiente a 490 nm). Usada para la visualización en color, análisis de agua, y la detección de ciertos tipos de vegetación y suelos.
- **B03:** Banda verde (correspondiente a 560 nm). Usada para la visualización en color, estudios de vegetación y análisis de agua.
- **B04:** Banda roja (correspondiente a 665 nm). Fundamental para la creación de imágenes RGB, detección de vegetación y análisis de suelos.

- **B05:** Banda de 705 nm (banda roja estrecha). Utilizada principalmente en el análisis de vegetación y su respuesta espectral.
- **B06:** Banda de 740 nm (rojo estrecho). Similar a la B05, también usada para el análisis de vegetación y agua.
- **B07:** Banda de 783 nm (rojo estrecho). Utilizada en el análisis de vegetación, suelos y agua.
- **B08:** Banda de infrarrojo cercano (NIR, 842 nm). Muy importante para la evaluación de la vegetación, como en el cálculo del índice NDVI.
- **B09:** Banda de agua vaporizada (940 nm). Usada principalmente para la detección de vapor de agua y análisis atmosférico.
- **B10:** Banda de infrarrojo térmico (1375 nm). Utilizada para el análisis de la temperatura de la superficie terrestre y la humedad.
- **B11:** Banda de infrarrojo de onda corta (1610 nm). Empleada en el estudio de la vegetación, suelos y humedad superficial.
- **B12:** Banda de infrarrojo de onda corta (2190 nm). Similar a la B11, utilizada en el análisis de agua, vegetación y suelos.

Por ejemplo, si queremos mostrar una imagen RGB, el código y el resultado es el siguiente:

```

1 red = B04.read(1) / 10000
2 green = B03.read(1) / 10000
3 blue = B02.read(1) / 10000
4
5 imagen_rgb = np.dstack((red, green, blue))
6
7 plt.imshow(imagen_rgb)
8 plt.title("Imagen RGB (B04, B03, B02)")
9 plt.axis('off')
10 plt.show()

```

Imagen RGB (B04, B03, B02)

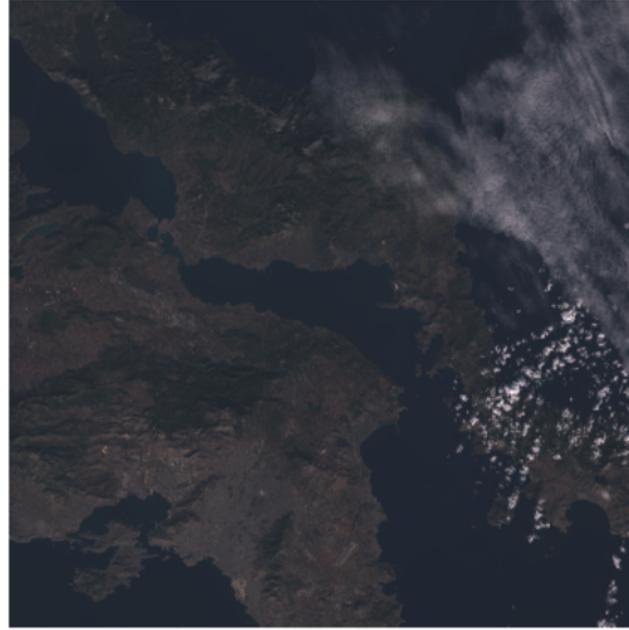


Figura 7: Visualización de la imagen en RGB

3.3. Validación y verificación de los metadatos

En esta sección, comprobaremos que los metadatos asociados a las imágenes que hemos descargado, como puede ser la resolución espacial, sean correctos y consistentes. Los metadatos permiten comprender cómo se debe interpretar la imagen y, por tanto, es muy importante verificar que estos se correspondan con los datos reales de la imagen. Para ello, abrimos una de las bandas; por ejemplo:

```
1 with rasterio.open(path_imagen1 + 'B04.jp2') as dataset:  
2     metadata = dataset.meta  
3     print(metadata)
```

Esto nos devuelve la siguiente información:

```
1 {  
2     "driver": "JP2OpenJPEG",  
3     "dtype": "uint16",  
4     "nodata": null,  
5     "width": 10980,  
6     "height": 10980,  
7     "count": 1,  
8     "crs": "PROJCS[\"WGS 84 / UTM zone 34N\", "  
9         "GEOGCS[\"WGS 84\", "  
10            "DATUM[\"WGS_1984\", "
```

```

11      "SPHEROID[\\"WGS 84\\",6378137,298.257223563,"
12      "AUTHORITY[\"EPSG\",\\"7030\\"]],""
13      "AUTHORITY[\"EPSG\",\\"6326\\"],"
14      "PRIMEM[\"Greenwich\",0,AUTHORITY[\"EPSG\",\\"8901\\"]],"
15      "UNIT[\"degree\",0.0174532925199433,AUTHORITY[\"EPSG\",\\"9122\\"]],"
16      "AUTHORITY[\"EPSG\",\\"4326\\"],"
17      "PROJECTION[\"Transverse_Mercator\"],"
18      "PARAMETER[\"latitude_of_origin\",0],"
19      "PARAMETER[\"central_meridian\",21],"
20      "PARAMETER[\"scale_factor\",0.9996],"
21      "PARAMETER[\"false_easting\",500000],"
22      "PARAMETER[\"false_northing\",0],"
23      "UNIT[\"metre\",1,AUTHORITY[\"EPSG\",\\"9001\\"]],"
24      "AXIS[\"Easting\",EAST],"
25      "AXIS[\"Northing\",NORTH],"
26      "AUTHORITY[\"EPSG\",\\"32634\\"]]",
27      "transform": "Affine(10.0, 0.0, 699960.0, 0.0, -10.0, 4300020.0)"
28 }

```

A partir de esta información, podemos verificar varios aspectos clave de la imagen:

- **Driver:** El controlador utilizado para leer la imagen es JP2OpenJPEG, lo cual es adecuado para imágenes en formato JPEG2000.
- **Tipo de datos:** La imagen tiene un tipo de datos `uint16`, lo que significa que cada píxel se representa con 16 bits sin signo.
- **Resolución espacial:** El `transform` muestra la resolución espacial de la imagen, con un valor de 10 metros por píxel en ambas direcciones.
- **Dimensiones:** La imagen tiene un ancho de 10.980 píxeles y una altura de 10.980 píxeles, lo que nos da un total de 120.536.400 píxeles.
- **Sistema de referencia de coordenadas (CRS):** El sistema de referencia de coordenadas es WGS 84 / UTM zone 34N, que es muy utilizado para representar datos geoespaciales en zonas del hemisferio norte.

Con esta información podemos asegurar que los metadatos son consistentes y correctos para la interpretación de la imagen. En particular, podemos verificar que la resolución espacial es la adecuada para los fines del análisis y que las dimensiones y el sistema de coordenadas coinciden con las expectativas del conjunto de datos.

4. Identificación y corrección de errores

En esta sección, nos centraremos en los errores más comunes que pueden presentarse en las imágenes RAW obtenidas del satélite y cómo podemos identificarlos y corregirlos.

4.1. Tipos de errores posibles en imágenes RAW

Los errores más comunes en las imágenes RAW son:

- **Valores nodata:** Áreas donde no hay datos disponibles. Estos valores pueden estar presentes por diversas razones, como áreas fuera del área de cobertura del satélite o problemas en la adquisición de datos.
- **Valores fuera del rango esperado:** Algunos píxeles pueden tener valores incorrectos debido a problemas de calibración o errores en la conversión de los datos del sensor a imágenes procesables. Esto puede incluir valores negativos en bandas que solo deberían contener valores positivos, o valores que exceden los límites de las bandas.
- **Distorsiones geométricas:** Se producen cuando los datos del satélite no se ajustan perfectamente a la realidad debido a factores como el movimiento del satélite, la curvatura de la Tierra o la distorsión provocada por la atmósfera.
- **Errores del sensor:** En algunos casos, es posible que el sensor del satélite falle al tomar la fotografía. Dependiendo del tipo de sensor, ésto puede verse como líneas verticales u horizontales negras en la imagen.

4.2. Localización de errores

Ahora localizamos los errores en las imágenes, que se encuentran en las carpetas DATASTRIP y QI_DATA. Es probable que las imágenes contenidas en estas carpetas contengan los errores mencionados en la sección anterior. Para localizarlos, podemos utilizar las siguientes estrategias:

- **Análisis visual de las imágenes:** Utilizamos herramientas como rasterio para abrir las imágenes y matplotlib para visualizarlas. Esto nos permite identificar errores visuales evidentes, como áreas vacías (valores nodata) o distorsiones.
- **Análisis estadístico:** Realizamos un análisis básico de los valores de los píxeles para identificar valores fuera del rango esperado o anómalos. Esto incluye verificar si existen valores negativos en bandas que deberían ser positivas o valores que exceden los límites establecidos para cada banda.
- **Máscaras de calidad:** En la carpeta QI_DATA, generalmente se incluyen las máscaras de calidad que indican la presencia de problemas como nubes,

sombra o valores nodata. Estas máscaras nos permiten identificar las áreas afectadas y gestionarlas adecuadamente.

El siguiente código carga una imagen de la carpeta DATASTRIP, la visualiza y realiza un análisis básico de los valores:

```
1 path_banda = "../datos/Imagen1/T34SGH_20241125T091321_B04.jp2"
2
3 with rasterio.open(path_banda) as dataset:
4     banda = dataset.read(1)
5
6     plt.imshow(banda, cmap='gray')
7     plt.title("Imagen Banda B04")
8     plt.colorbar()
9     plt.show()
10
11     nodata_value = dataset.nodata
12     if nodata_value is not None:
13         print(f"Valor nodata: {nodata_value}")
14         banda[banda == nodata_value] = np.nan
15     plt.imshow(banda, cmap='gray')
16     plt.title("Imagen Banda B04 (sin nodata)")
17     plt.colorbar()
18     plt.show()
19
20     print(f"Valor mínimo: {np.min(banda)}")
21     print(f"Valor máximo: {np.max(banda)}")
22     print(f"Media: {np.mean(banda)}")
```

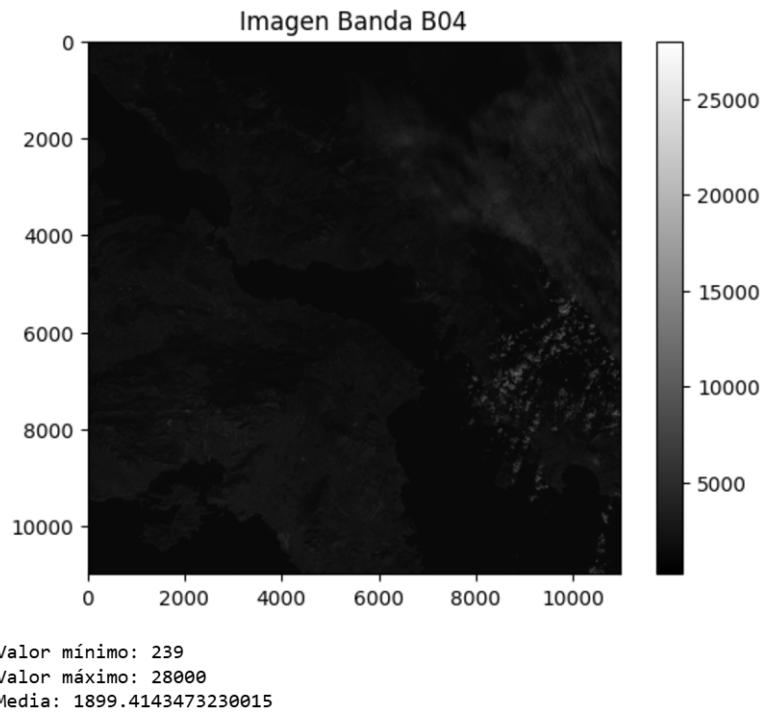


Figura 8: Análisis de los valores de la banda B04

4.3. Métodos aplicados para la corrección de errores

La corrección de errores es importantísimo para garantizar que las imágenes sean utilizables en el análisis posterior y reflejen con precisión la información que el satélite intenta capturar.

4.3.1. Corrección de valores nodata

Para cada una de las 12 bandas, comprobamos si existen valores nodata, y en ese caso los cambiamos por el valor medio. Las imágenes cambiadas las guardamos en otra carpeta. A modo de ejemplo:

```
1 banda_3 = B03.read(1)
2
3 nodata_value = B03.nodata
4 print("Valor nodata:", nodata_value)
5
6 if nodata_value is not None:
7     nodata_mask = banda_3 == nodata_value
8 else:
9     nodata_mask = np.zeros_like(banda_3, dtype=bool) # Si no hay valor nodata definido
10
```

```

11 plt.imshow(nodata_mask, cmap='gray')
12 plt.title('Máscara de nodata')
13 plt.colorbar()
14 plt.show()
15
16 # Rellenamos los valores nodata con el valor medio
17 media_banda_3 = np.nanmean(banda_3)
18 banda_3_corregida = np.where(nodata_mask, media_banda_3, banda_3)
19
20 # Visualizamos la banda corregida
21 plt.imshow(banda_3_corregida, cmap='gray')
22 plt.title('Banda 03 corregida')
23 plt.colorbar()
24 plt.show()
25
26 banda_3_corregida_uint16 = banda_3_corregida.astype(np.uint16)
27 output_dir = "../datos/Imagen1/bandasCorregidas"
28 os.makedirs(output_dir, exist_ok=True)
29 output_path = os.path.join(output_dir, "Banda_03_corregida.jp2")
30
31 with rasterio.open(output_path, 'w', driver='JP2OpenJPEG',
32                     height=banda_3.shape[0], width=banda_3.shape[1], count=1, dtype='uint16',
33                     crs=B03.crs, transform=B03.transform, nodata=nodata_value) as dst:
34     dst.write(banda_3_corregida_uint16, 1)
35
36 print(f"Imagen corregida guardada en: {output_path}")

```

Valor nodata: None

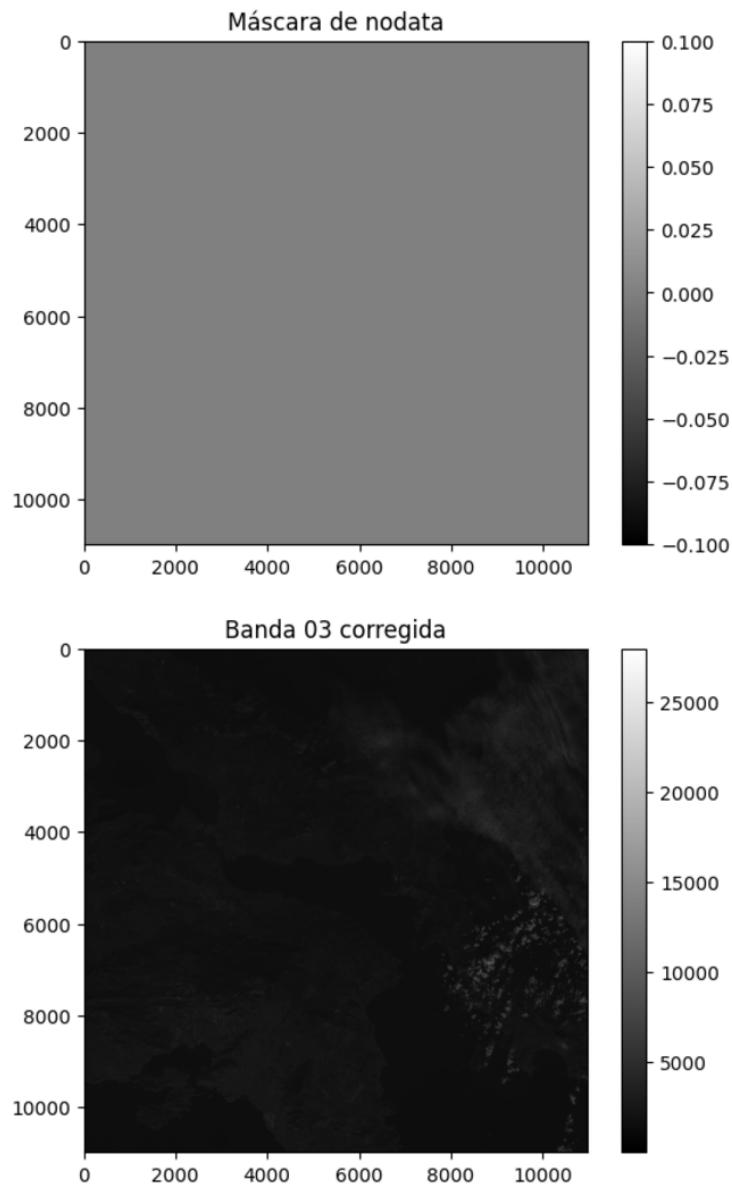


Imagen corregida guardada en: ../datos/Imagen1/bandasCorregidas\Banda_03_corregida.jp2

Figura 9: Resultado tras corregir valores nodata

En este caso, todas las bandas eran correctas, por lo que la imagen que obtenemos es la misma.

4.3.2. Corrección de valores fuera del rango esperado

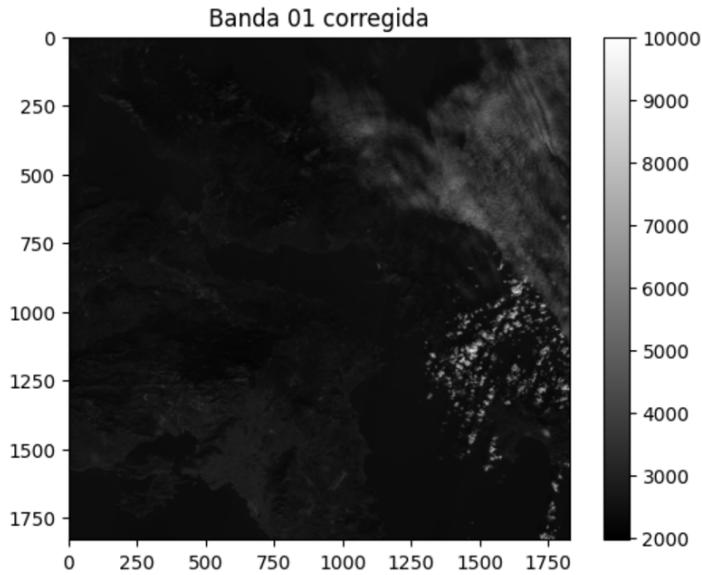
Ahora, comprobamos que todos los valores estén en el rango que queremos. Para ello, definimos el intervalo y los valores que se pasen por arriba los sustituimos por el máximo del intervalo, y los que se pasen por abajo por el mínimo del intervalo.

```
1 min_val = 0
2 max_val = 10000
3
4 banda_1 = B01.read(1)
5 banda_1_corregida = banda_1.copy()
6
7 # Corrección de valores fuera de rango:
8 banda_1_corregida[banda_1_corregida < min_val] = min_val
9 banda_1_corregida[banda_1_corregida > max_val] = max_val
10
11 print(f"Valor mínimo corregido: {banda_1_corregida.min()}")
12 print(f"Valor máximo corregido: {banda_1_corregida.max()}")
13 print(f"Media corregida: {banda_1_corregida.mean()}")
14
15 # Visualizar la banda corregida
16 plt.imshow(banda_1_corregida, cmap='gray')
17 plt.title('Banda 01 corregida')
18 plt.colorbar()
19 plt.show()
20
21 banda_1_corregida_uint16 = banda_1_corregida.astype(np.uint16)
22 output_dir = "../datos/Imagen1/bandasCorregidas"
23 os.makedirs(output_dir, exist_ok=True)
24 output_path = os.path.join(output_dir, "Banda_01_corregida.jp2")
25
26 with rasterio.open(output_path, 'w', driver='JP2OpenJPEG',
27                     height=banda_1.shape[0], width=banda_1.shape[1], count=1,
28                     dtype='uint16', crs=B01.crs, transform=B01.transform,
29                     nodata=nodata_value) as dst:
30     dst.write(banda_1_corregida_uint16, 1)
31
32 print(f"Imagen corregida guardada en: {output_path}")
```

```

Valor mínimo corregido: 1968
Valor máximo corregido: 10000
Media corregida: 2647.4990154976263

```



```
Imagen corregida guardada en: ../datos/Imagen1/bandasCorregidas\Banda_01_corregida.jp2
```

Figura 10: Resultado tras corregir valores fuera de rango

4.3.3. Corrección de errores del sensor

En primer lugar, comprobaremos si tuvimos errores de algún tipo en el sensor del satélite al tomar la fotografía. Como explicamos en una sección anterior, esto se vería reflejado como que algún componente del sensor falla durante la captura de información y por tanto no toma bien los datos, lo que se traduce en rallas negras, verticales u horizontales, en la imagen. Pueden incluso llegar a ser imperceptibles, por lo que es mejor comprobarlo nosotros mismos:

```

1 path_banda1 = "../datos/Imagen1/T34SGH_20241125T091321_B01.jp2"
2
3 with rasterio.open(path_banda1) as src:
4     banda = src.read(1)
5
6 suma_filas = np.sum(banda, axis=1)
7 filas_cero = np.where(suma_filas == 0)[0]
8
9 if len(filas_cero) > 0:
10     print(f"Las siguientes filas suman 0: {filas_cero}")
11 else:
12     print("Ninguna fila suma 0")
13

```

```

14 suma_columnas = np.sum(banda, axis=0)
15 columnas_cero = np.where(suma_columnas == 0)[0]
16
17 if len(columnas_cero) > 0:
18     print(f"Las siguientes columnas suman 0: {columnas_cero}")
19 else:
20     print("Ninguna columna suma 0")
21
22 if len(filas_cero)==0 & len(columnas_cero)==0:
23     print("El satélite no tuvo errores en el sensor al tomar la foto")

```

Al ejecutarlo sobre nuestra banda, obtenemos:

```

Ninguna fila suma 0
Ninguna columna suma 0
El satélite no tuvo errores en el sensor al tomar la foto

```

Figura 11: Resultado tras corregir errores del sensor

Cabe destacar que no incluiremos la comprobación y posterior corrección de todas las bandas, pues el código y el resultado son muy similares y no aportan al trabajo, pero si hacemos una breve comprobación, veremos que todas son correctas.

En caso de haber encontrado errores, tenemos múltiples soluciones, pero quizás la más sencilla sea la siguiente. Supongamos que tenemos una línea vertical negra. Entonces, cada píxel de la línea lo sustituimos por el **valor medio** de los píxeles de alrededor. De esta forma, damos una especie de continuidad y suavidad al error de la imagen, y desde fuera será imperceptible.

5. Corrección geográfica

La **corrección geográfica** es un proceso muy importante en el análisis de imágenes satelitales, ya que garantiza que los datos espaciales se alineen correctamente con las coordenadas del terreno [14]. Este proceso busca mejorar la precisión geográfica de la imagen, asegurando que las distorsiones o los desplazamientos causados por diversos factores (como la curvatura de la Tierra, errores del sensor o movimientos del satélite) sean corregidos [10].

La **georreferenciación** es un paso importante dentro de la corrección geográfica, en el que se asignan coordenadas reales a los píxeles de la imagen [4]. Una vez georreferenciada, la imagen puede ser alineada y comparada con otras fuentes de datos geoespaciales, como mapas topográficos, imágenes aéreas o datos de sistemas de información geográfica (SIG).

5.1. Conceptos de corrección geográfica

Los principales conceptos clave son:

- **Georreferenciación:** Es el proceso de asignar coordenadas geográficas (como latitud y longitud) a cada píxel de la imagen. Para ello, se utilizan **puntos de control geográficos** (GCP) que sirven como referencia [4].
- **Transformación de coordenadas:** Es el proceso mediante el cual se realiza el ajuste de la imagen para corregir las distorsiones geométricas.
- **Proyección cartográfica:** Durante la corrección geográfica, es crucial asegurar que la imagen esté en la proyección cartográfica correcta, ya que diferentes proyecciones afectan la representación de las coordenadas geográficas.

5.2. Verificación de la georreferenciación

La verificación de la calidad georreferenciada es un paso crucial después de realizar la corrección geográfica de una imagen satelital. Este proceso asegura que las coordenadas asignadas a los píxeles correspondan de manera precisa a su posición real en el terreno. Para realizar esta verificación, se emplean diversas técnicas y herramientas, que van desde métodos visuales hasta análisis cuantitativos con puntos de control geográficos.

Sin embargo, nuestra imagen podemos comprobar que está relativamente bien georreferenciada. Simplemente haciendo una breve comprobación con Google Maps podemos verlo. Si no, tendríamos que usar herramientas más avanzadas como QGIS.

En cualquier caso, vamos a aplicar el siguiente código para comprobarlo mejor, y no solo de manera visual:

```

1 from rasterio.warp import reproject, Resampling, calculate_default_transform
2
3 path_imagen_band1 = "../datos/Imagen1/T34SGH_20241125T091321_B01.jp2"
4 path_salida = "../datos/Imagen1/bandasCorregidas/"
5
6 with rasterio.open(path_imagen_band1) as src:
7     crs = src.crs
8     transform = src.transform
9     nodata_value = src.nodata
10    width = src.width
11    height = src.height
12
13    imagen_band1 = src.read(1)
14
15    destino_crs = "EPSG:4326"
16    transform_dest, width_dest, height_dest = calculate_default_transform(
17        crs, destino_crs, width, height, *src.bounds
18    )
19
20    imagen_corregida = np.empty((height_dest, width_dest),
21        dtype=imagen_band1.dtype)
22
23    reproject(
24        source = imagen_band1,
25        destination = imagen_corregida,
26        src_transform = transform,
27        src_crs = crs,
28        dst_transform = transform_dest,
29        dst_crs = destino_crs,
30        resampling = Resampling.bilinear
31    )
32
33    print(f"Rango de valores antes de re-proyección: {imagen_band1.min()} - {imagen_band1.max()}")
34    print(f"Rango de valores después de re-proyección: {imagen_corregida.min()} - {imagen_corregida.max()}")
35
36    imagen_corregida_display = imagen_corregida.astype(float)
37    imagen_corregida_display[imagen_corregida_display == nodata_value] = np.nan
38
39    max_val = np.nanmax(imagen_corregida_display)
40    if max_val > 0:
41        imagen_corregida_display /= max_val
42
43    plt.imshow(imagen_corregida_display, cmap='gray')
44    plt.title('Banda 1 Corregida Geográficamente')
45    plt.colorbar()
46    plt.show()
47
48    with rasterio.open(
49

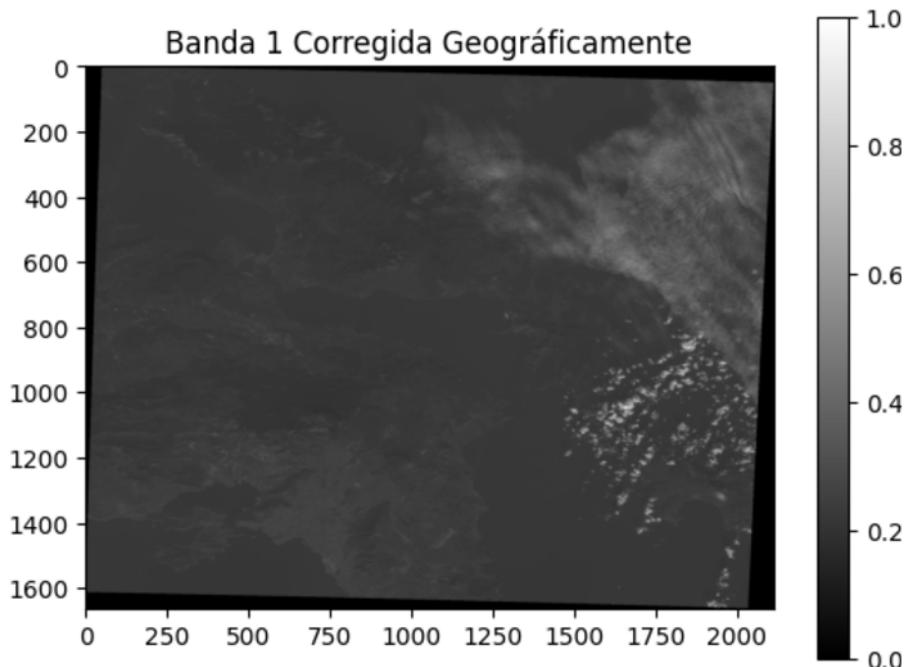
```

```

51         os.path.join(path_salida, "Banda_1_corregida.tif"),
52         'w', driver = 'GTiff',
53         height = height_dest, width = width_dest, count = 1,
54         dtype = imagen_banda1.dtype, crs = destino_crs,
55         transform = transform_dest, nodata = nodata_value
56     ) as dst:
57         dst.write(imagen_corregida, 1)
58
59     print("Corrección geográfica aplicada y guardada en 'Banda_1_corregida.tif'")

```

Rango de valores antes de re-proyección: 1968 - 11151
Rango de valores después de re-proyección: 0 - 10961



Corrección geográfica aplicada y guardada en 'Banda_1_corregida.tif'

Figura 12: Corrección geográfica de la banda 1

Por supuesto, habría que corregir todas las bandas. Sin embargo, el código para hacerlo es análogo y no nos aporta nada al trabajo posterior. En cualquier caso, y como veníamos comentando, la imagen ya estaba bastante bien georreferenciada, y en efecto podemos comprobar que apenas ha cambiado la posición respecto de la banda original. Por ello, seguiremos usando todas las bandas tal y como están, en lugar de corregirlas una a una (aunque en realidad deberíamos, pero no cambiará el resultado en secciones posteriores).

A modo de curiosidad, si queremos corregir las bandas de la segunda imagen, aplicando el mismo código obtenemos que la banda 6 ya está perfectamente georreferenciada:

Rango de valores antes de re-proyección: 0 - 28000
Rango de valores después de re-proyección: 0 - 28000

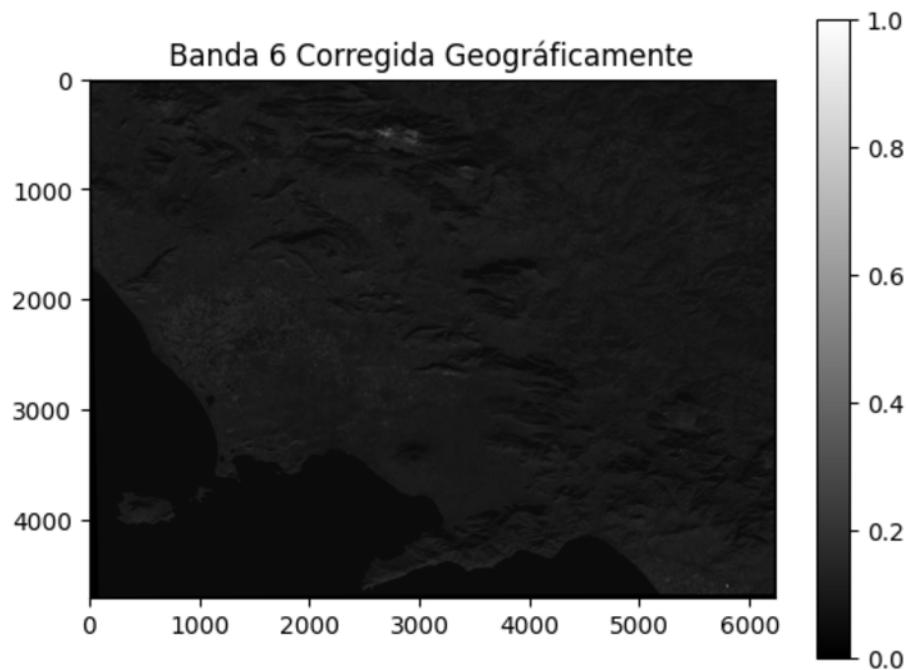


Figura 13: Corrección geográfica de la banda 6

6. Aplicación de índices de vegetación

Veamos en esta sección cómo aplicar algunos índices de vegetación a nuestras imágenes. En primer lugar, veremos qué índices hemos seleccionado, explicando brevemente para qué se utilizan. Posteriormente, los implementaremos en Python para las dos imágenes, estudiaremos los resultados y compararemos los índices obtenidos para cada imagen.

6.1. Selección de índices a utilizar

Utilizaremos los siguientes índices en cada imagen para comparar y analizar los resultados: [13]

1. **RVI (Ratio Vegetation Index)**: Es uno de los índices más simples, y fue diseñado para evaluar la proporción entre la reflectancia en el infrarrojo cercano (NIR) y el rojo.

$$RVI_{ij} = \frac{ND_{ij}(IR)}{ND_{ij}(R)} \quad (1)$$

Sirve para determinar la densidad de la vegetación, especialmente en áreas agrícolas, para la monitorización de cultivos, para la detección de estrés en plantas debido a sequías o enfermedades...

2. **NDVI (Normalized Difference Vegetation Index)**: Es el más común, y es utilizado para medir la cantidad de vegetación verde en una región. Compara la reflectancia en el NIR y en el rojo.

$$NDVI_{ij} = \frac{ND_{ij}(IR) - ND_{ij}(R)}{ND_{ij}(IR) + ND_{ij}(R)} \quad (2)$$

Sirve para la monitorización de la salud de los cultivos y el ecosistema, para los mapas de cobertura vegetal y áreas deforestadas, para la estimación de la biomasa...

3. **SAVI (Soil Adjusted Vegetation Index)**: Es una mejora del NDVI que corrige la influencia del brillo del suelo en áreas con vegetación escasa.

$$SAVI_{ij} = \frac{ND_{ij}(IR) - ND_{ij}(R)}{ND_{ij}(IR) + ND_{ij}(R)} (1 + L) \quad (3)$$

donde L es un factor de ajuste que depende de la cobertura vegetal. Sirve para la monitorización de áreas semiáridas o con vegetación escasa, para la evaluación de vegetación en suelos heterogéneos...

4. **TVI (Transformed Vegetation Index)**: Es un índice derivado del NDVI, que introduce una transformación matemática para realzar la sensibilidad a la vegetación, especialmente en áreas con valores intermedios de

NDVI. Su objetivo es simplificar el rango de valores del NDVI y mejorar la interpretación en escenarios de vegetación mixta.

$$TVI = \sqrt{|NDVI + 0,5|} \quad (4)$$

Sirve para la monitorización de cultivos y pastizales, donde los valores de NDVI pueden ser difíciles de interpretar, para las evaluaciones rápidas de cambios en la vegetación, al usar una escala menos dispersa que el NDVI, para los análisis de transición en áreas con mezcla de vegetación densa y suelos expuestos...

5. **BAI (Burned Area Index):** Es un índice utilizado para detectar áreas afectadas por incendios forestales mediante la comparación de la reflectancia en bandas específicas (rojo e infrarrojo cercano).

$$BAI_{ij} = \frac{1}{(ND_{ij(R)} - 0,1)^2 + (ND_{ij(IR)} - 0,06)^2} \quad (5)$$

Sirve para detectar áreas quemadas tras un incendio forestal, para evaluar la severidad de los incendios...

6.2. Implementación de los índices y análisis de los resultados

Para cada índice de vegetación, incluiremos únicamente el código para la primera imagen, pero el de la segunda es exactamente igual.

6.2.1. RVI (Ratio Vegetation Index)

El código para implementar el RVI es el siguiente:

```

1 red = B04.read(1).astype(float)
2 nir = B08.read(1).astype(float)
3
4 rvi = nir / red
5
6 rvi[np.isnan(rvi)] = -9999
7
8 plt.imshow(rvi, cmap='viridis', vmin=0, vmax=10)
9 plt.title("RVI")
10 plt.colorbar(label="Valor RVI")
11 plt.axis('off')
12 plt.show()

```

Obtenemos los siguientes resultados:

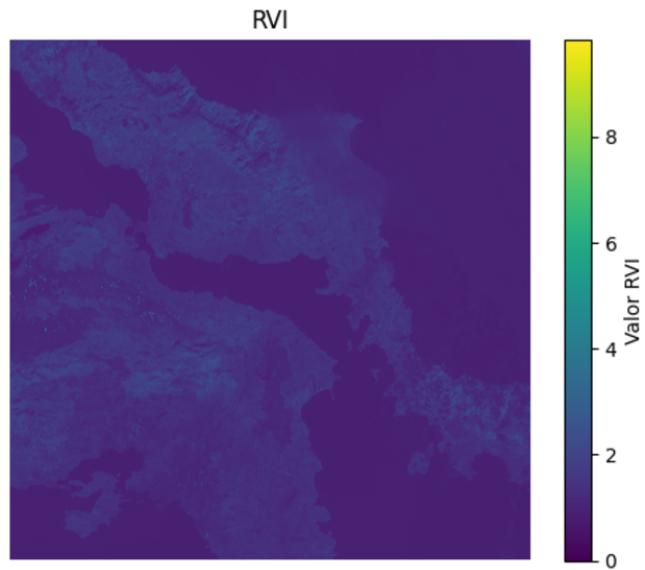


Figura 14: RVI para la primera imagen

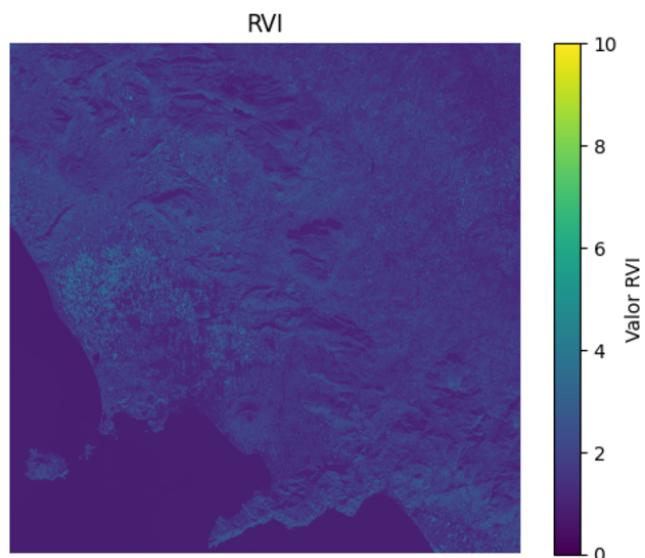


Figura 15: RVI para la segunda imagen

En ambas imágenes los valores del RVI se distribuyen entre 0 y 10 aproximadamente. En la primera, correspondiente a la costa de Grecia, las áreas con valores altos (verdes y amarillas) representan vegetación densa en zonas costeras o áreas montañosas, mientras que las zonas con valores más bajos (azules)

corresponden a agua y suelos desnudos. Esto tiene sentido si consideramos la geografía griega, caracterizada por costas rocosas. Respecto a la segunda imagen, correspondiente a la costa de Italia, dominan los valores intermedios, y hay algunas regiones con valores altos, lo cual es coherente con el paisaje italiano, donde hay más tierras agrícolas y vegetación uniforme.

Comparando ambas imágenes, la primera muestra una mayor heterogeneidad en el índice, lo que refleja una geografía más compleja con diversos tipos de cobertura. En contraste, la segunda imagen presenta un patrón más uniforme, lo que sugiere una cobertura vegetal más continua, posiblemente tierras agrícolas cultivadas. Los valores altos en ambas imágenes tienen sentido considerando su ubicación: en Grecia, podrían estar asociados a bosques o vegetación en áreas protegidas o montañosas, mientras que en Italia, podrían indicar campos cultivados intensivamente, como viñedos o huertos. En general, estas observaciones coinciden con las características físicas y de uso del suelo de ambas regiones, siendo Grecia más accidentada y diversa, e Italia más homogénea en términos de vegetación en algunas áreas.

6.2.2. NDVI (Normalized Difference Vegetation Index)

El código para implementar el NDVI es el siguiente:

```

1 red = B04.read(1).astype(float)
2 nir = B08.read(1).astype(float)
3
4 ndvi = (nir - red) / (nir + red)
5
6 ndvi[np.isnan(ndvi)] = -9999
7
8 plt.imshow(ndvi, cmap='RdYlGn', vmin=-1, vmax=1)
9 plt.title("NDVI")
10 plt.colorbar(label="Valor NDVI")
11 plt.axis('off')
12 plt.show()
```

Obtenemos los siguientes resultados:

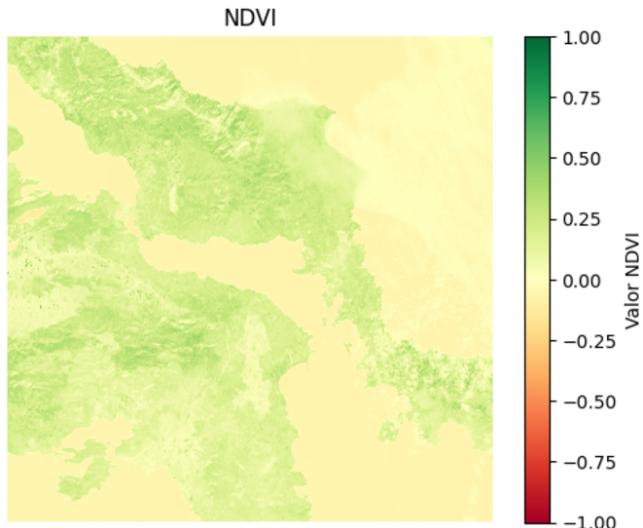


Figura 16: NDVI para la primera imagen

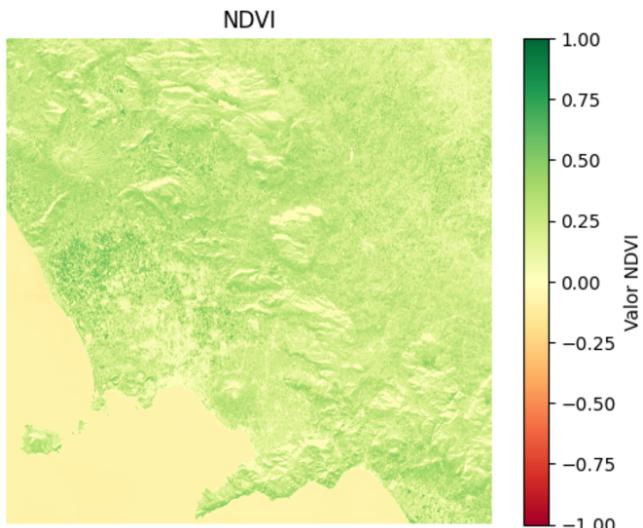


Figura 17: NDVI para la segunda imagen

Principalmente, ambas imágenes presentan valores de NDVI entre 0 y 0.5 aproximadamente, con algunas zonas alcanzando valores cercanos a 1 (verde oscuro). Esto indica áreas con vegetación moderada a densa en ciertas regiones, como montañas o zonas protegidas, mientras que los valores bajos (amarillo y rojo) corresponden a suelos desnudos, áreas urbanizadas o agua.

Pese a ello, encontramos diferencias. Por lo general, los valores de la primera imagen son más bajos que los de la segunda, lo cual tiene sentido por lo comentado anteriormente con el RVI. Además, en la segunda imagen, vuelve a destacar la zona que mencionábamos antes (verde oscuro), confirmando aun más que se trata de una zona de vegetación frondosa. Y es que si investigamos un poco en Google Maps, se trata de un Parque Regional de Italia. De hecho, toda esa zona, quitando algunas ciudades de la costa (como puede ser Nápoles) es de un verde bastante intenso, debido a que si nos alejamos de la costa lo que más encontramos son parques naturales.

6.2.3. SAVI (Soil Ajusted Vegetation Index)

El código para implementar el SAVI es el siguiente:

```

1 red = B04.read(1).astype('float32')
2 nir = B08.read(1).astype('float32')
3
4 L = 0.5
5 savi = ((nir - red) / (nir + red + L)) * (1 + L)
6
7 savi[np.isinf(savi)] = np.nan
8
9 plt.imshow(savi, cmap='YlGn')
10 plt.title('SAVI')
11 plt.colorbar(label='Valor SAVI')
12 plt.axis('off')
13 plt.show()

```

Obtenemos los siguientes resultados:

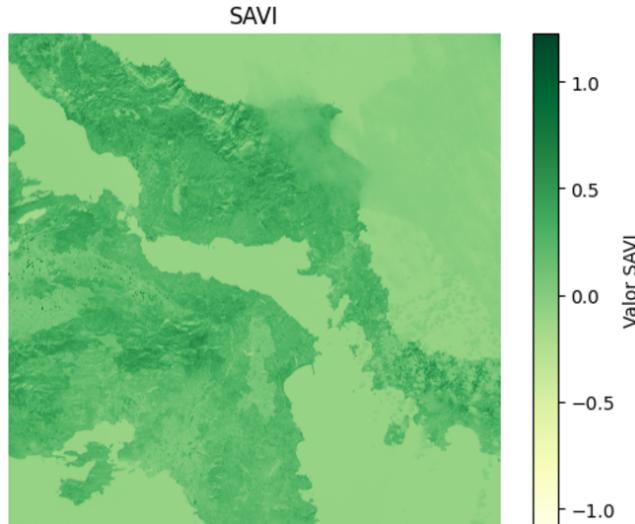


Figura 18: SAVI para la primera imagen

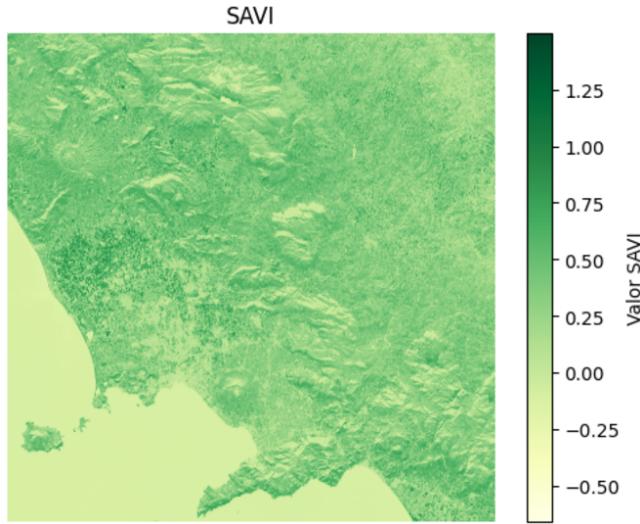


Figura 19: SAVI para la segunda imagen

La primera imagen (Grecia) presenta valores de SAVI que oscilan entre -1.0 y 1.0. Esto sugiere una cobertura vegetal moderada, con áreas donde la vegetación es menos densa o prácticamente inexistente, representadas por tonos más claros (amarillos). Las zonas de vegetación más densa se encuentran dispersas, aunque no son muchas. Por otro lado, la segunda imagen (Italia) tiene un rango de SAVI más amplio, llegando hasta 1.25. Esto indica la presencia de áreas con vegetación más densa y saludable, con tonos oscuros predominantes (verde intenso). Las áreas con baja vegetación o suelo desnudo son menos frecuentes, reflejando una cobertura vegetal más extensa y homogénea, posiblemente debido a un entorno más fértil y húmedo, típico del paisaje italiano.

6.2.4. TVI (Transformed Vegetation Index)

El código para implementar el TVI es el siguiente:

```

1 red = B04.read(1).astype('float32')
2 nir = B08.read(1).astype('float32')
3
4 ndvi = (nir - red) / (nir + red)
5 ndvi[np.isnan(ndvi)] = np.nan
6
7 tvi = np.sqrt((ndvi + 0.5))
8
9 tvi[np.isnan(tvi)] = 0
10
11 plt.imshow(tvi, cmap='BrBG')
12 plt.title('TVI')
13 plt.colorbar(label='Valor TVI')
```

```
14 plt.axis('off')
15 plt.show()
```

Obtenemos los siguientes resultados:

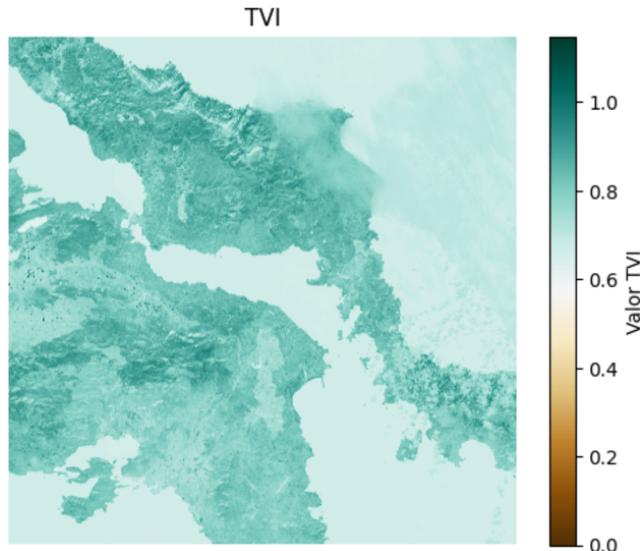


Figura 20: TVI para la primera imagen

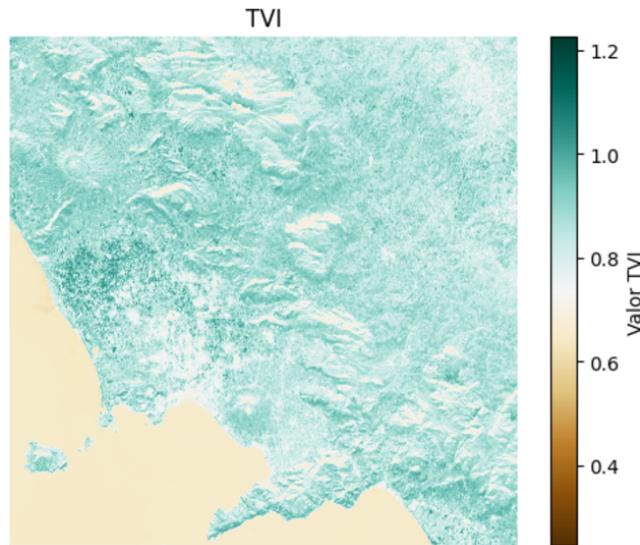


Figura 21: TVI para la segunda imagen

Ambas imágenes destacan la presencia de vegetación moderada a densa, con valores bajos en áreas costeras o descubiertas. Italia presenta valores más altos, lo que sugiere una vegetación más densa y uniforme, mientras que en Grecia los valores moderados dominan, probablemente debido a un entorno más árido. Estas variaciones reflejan diferencias en la fertilidad del suelo, el clima y el uso del territorio, características de las zonas mediterráneas.

6.2.5. BAI (Burned Area Index)

Los índices de vegetación también pueden utilizarse para medir otras cosas, no solo temas relacionados con los ecosistemas y las plantas, como los que hemos visto hasta ahora. Por ejemplo, veamos qué ocurre si aplicamos el BAI.

El código para implementar el BAI es el siguiente:

```

1 red = B04.read(1).astype('float32')
2 nir = B08.read(1).astype('float32')
3
4 bai = 1 / (((red - 0.06) ** 2) + ((nir - 0.1) ** 2))
5 bai[np.isnan(bai)] = np.nan
6
7 plt.imshow(bai, cmap='hot')
8 plt.title('BAI')
9 plt.colorbar(label='Valor BAI')
10 plt.axis('off')
11 plt.show()
```

Obtenemos los siguientes resultados:

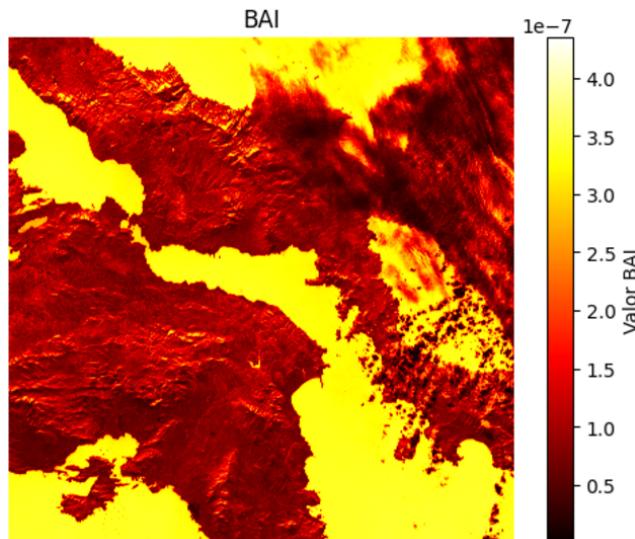


Figura 22: BAI para la primera imagen

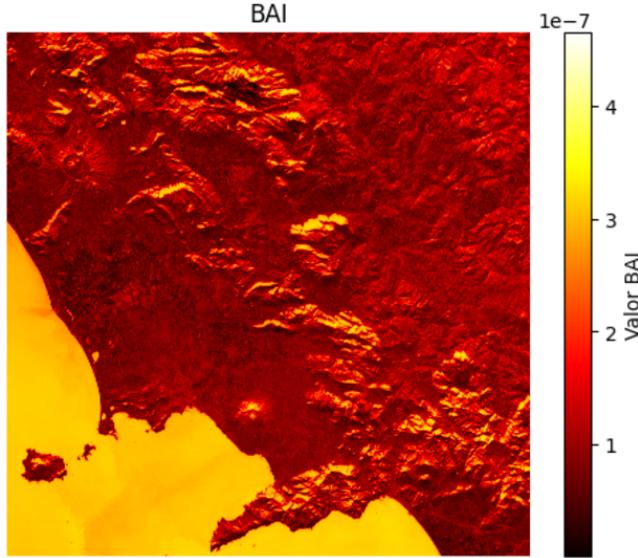


Figura 23: BAI para la segunda imagen

En la primera imagen predominan valores BAI intermedios y altos (rojos). Esto sugiere la presencia de extensas áreas afectadas por incendios recientes. Los tonos más claros (amarillos) indican áreas no quemadas o sin vegetación, que corresponden terrenos costeros o desnudos y al mar.

Los valores de BAI en Italia son similares, con predominancia de valores altos en tonos rojos intensos, especialmente en regiones interiores. Esto indica que también aquí se detectan áreas quemadas significativas. Las zonas más claras se ubican en regiones costeras o donde la vegetación no es densa.

Aunque el rango de valores es similar, Grecia parece tener una mayor concentración de áreas afectadas (valores altos más extendidos), mientras que en Italia los valores altos están más localizados, posiblemente debido a diferencias en el manejo del territorio o los tipos de vegetación afectada.

Si buscamos algo de información acerca de incendios en ambos países, encontramos que Grecia ha sufrido incendios significativos en estas zonas (Ática) en 2021 y 2023 [23], mientras que Italia también ha sufrido algunos, intensificados por olas de calor extremo. Estas condiciones explican la intensidad del índice en ambas imágenes y reflejan la vulnerabilidad de estos ecosistemas mediterráneos frente al cambio climático.

7. Transformaciones

En esta sección, y a modo de extra del trabajo, vamos a realizar una serie de transformaciones a nuestra primera imagen, las cuales hemos ido estudiando a lo largo del curso tanto en teoría como prácticas.

Cabe destacar que utilizaremos todo el tiempo la banda 4 correspondiente al rojo. Para ello, antes tenemos que normalizarla para que sus valores estén en el rango [0, 255]:

```
1 banda_min = np.min(red)
2 banda_max = np.max(red)
3 banda_normalizada = (red - banda_min) / (banda_max - banda_min) * 255
4 red_norm = banda_normalizada.astype(np.uint8)
```

Además de esto, y debido a que la calidad de la imagen es muy alta, para algunas funciones utilizaremos la banda anterior diezmada: [21]

```
1 def diezmado(img, R):
2     paso = int(1 / R)
3     resimg = img[:,::paso, ::paso]
4
5     return resimg
6
7 red_diez = diezmado(red_norm, 0.1)
```

7.1. Histogramas

En primer lugar, ¿qué es un histograma? [11]

Histograma: Representación gráfica del número de ocurrencias de cada nivel digital de una imagen.

El código para obtener un histograma a partir de una imagen es el siguiente: [19]

```
1 def histograma(img):
2     val, freq = np.unique(img, return_counts=True) # diccionario contador
3
4     plt.figure(figsize=(10, 6))
5     plt.bar(val, freq, color='gray', width=1.0)
6     plt.title('Histograma')
7     plt.xlabel('Pixel')
8     plt.ylabel('Frecuencia')
9     plt.grid(True)
10    plt.show()
11
12 histograma(red_norm)
```

El histograma que resulta de nuestra imagen es el siguiente:

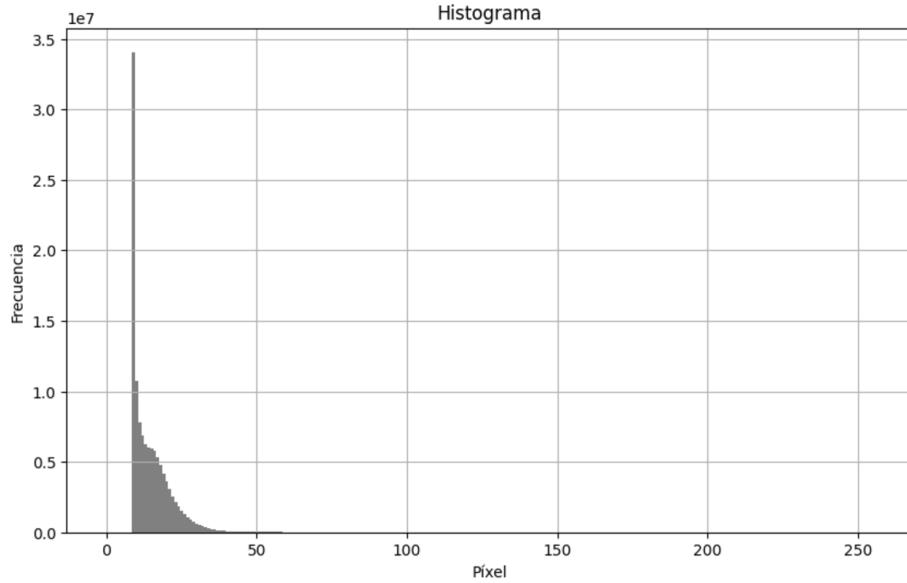


Figura 24: Histograma

Como podemos ver, la mayor parte de los valores de los píxeles están entre 0 y 50, por lo que la mayoría de los píxeles tienen intensidades bajas en la banda del rojo. Esto ya lo podíamos intuir, puesto que cuando mostramos todas las bandas en un primer lugar, la del rojo estaba muy apagada. Si mostrásemos por ejemplo el histograma de la banda 10, saldrían valores en un rango más alto (ver Figura 6).

También podemos realizar una expansión lineal de nuestra imagen.

Expansión lineal: Transformación lineal entre el rango de valores de los niveles digitales de la imagen y el rango de visualización de la nueva imagen.

El código necesario para llevar esto a cabo es el siguiente: [19]

```
1 def expand(img, cmin, cmax):
2     resimg = img.copy()
3
4     for i in range(img.shape[0]):
5         for j in range(img.shape[1]):
6             NDij = img[i, j]
7             if NDij <= cmin:
8                 resimg[i, j] = 0 # nuevo límite inferior
```

```

9         elif NDij >= cmax:
10            resimg[i, j] = 255 # nuevo límite superior
11        else:
12            resimg[i, j] = (NDij - cmin) * (255) / (cmax - cmin)
13
14    return resimg
15
16 def expan2(img, cmin, cmax):
17     resimg = img.copy().astype(np.float32)
18
19     resimg = np.clip(resimg, cmin, cmax)
20     resimg = (resimg - cmin) * 255 / (cmax - cmin)
21
22     resimg = resimg.astype(np.uint8)
23     return resimg
24
25 img_exp = expan(red_norm, 10, 200)

```

El histograma resultante de aplicar la expansión lineal es el siguiente:

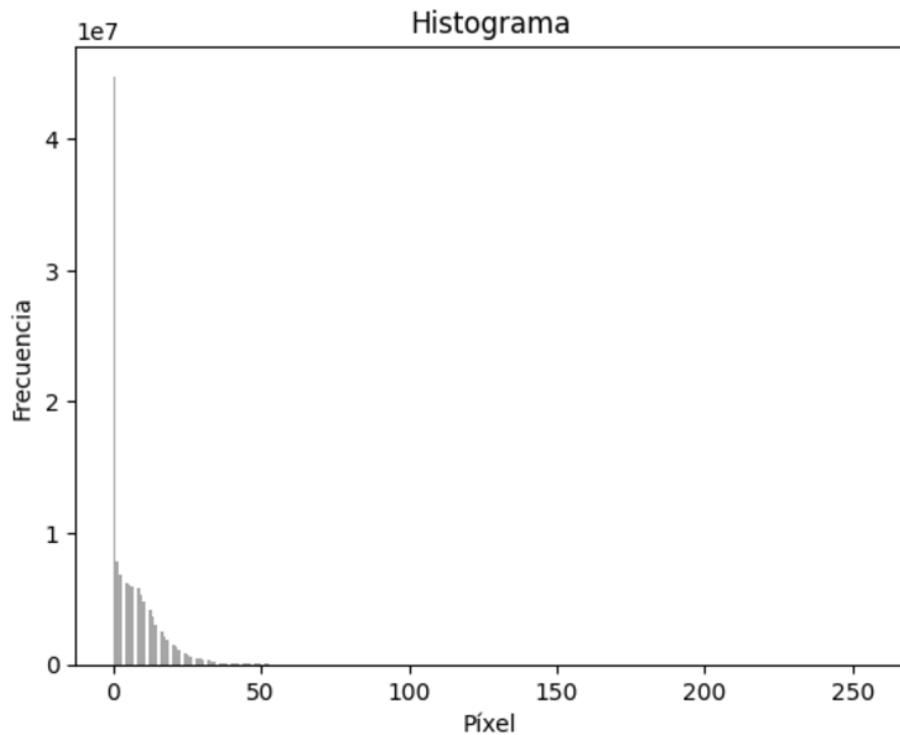


Figura 25: Histograma tras la expansión lineal

Es interesante comparar la imagen resultante con la original. Para ello: [19]

```
1 plt.figure(1)
2 plt.imshow(red_norm, cmap='gray')
3
4 plt.figure(2)
5 plt.imshow(red_expan, cmap='gray')
```

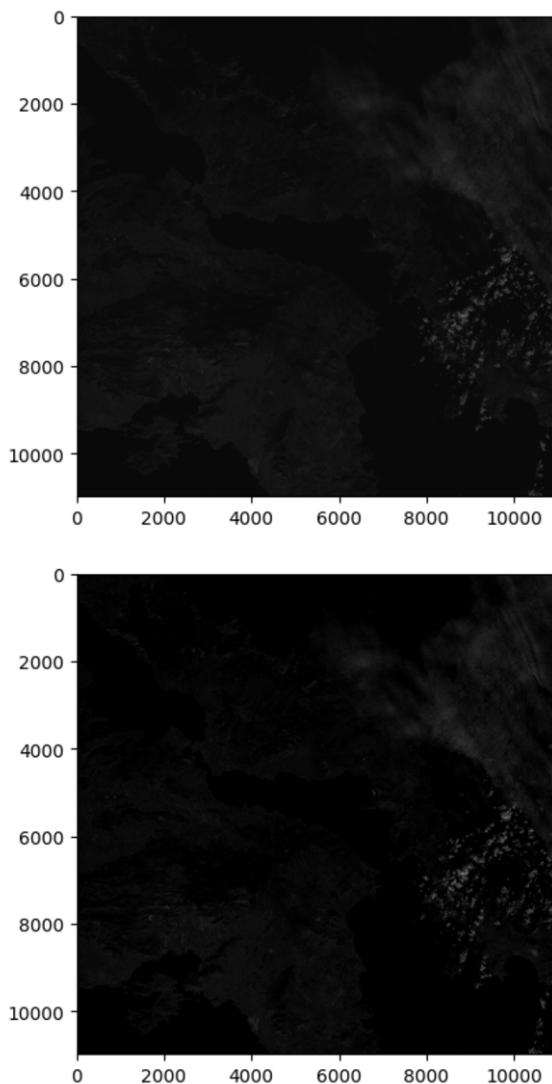


Figura 26: Expansión lineal

No hay demasiada diferencia entre las imágenes. Quizás, lo único apreciable es que ahora las nubes se ven algo más claras.

A fin de mejorar el contraste, podemos hacer también un corte de colas. [11]

Corte de colas: Técnica para eliminar los valores extremos de un conjunto de datos, ajustando un rango definido para mejorar el contraste y destacar detalles, especialmente en imágenes.

Debemos tener cuidado, puesto que un corte demasiado agresivo podría dejar fuera información importante. El corte de colas se haría de la siguiente forma: [19]

```
1 def corte(img, porc):
2     resimg = img.copy()
3     val, frec = histograma(img)
4     num_pixel = img.size
5     num_pixel_corte = int(porc * num_pixel / 100)
6
7     acc_inf = 0
8     acc_sup = 0
9
10    for valor, frecuencia in zip(val, frec):
11        acc_inf += frecuencia
12        if acc_inf >= num_pixel_corte:
13            NDcortemin = valor
14            break
15
16    for valor, frecuencia in zip(reversed(val), reversed(frec)):
17        acc_sup += frecuencia
18        if acc_sup >= num_pixel_corte:
19            NDcortemax = valor
20            break
21
22    resimg = expand(resimg, NDcortemin, NDcortemax)
23
24    return resimg
25
26 red_corte = corte(red_norm, 0.02)
```

En este caso, usaremos la imagen diezmada puesto que tarda mucho en ejecutarse con la banda original.

Mostramos a continuación el nuevo histograma y la comparación entre la imagen diezmada y la imagen tras aplicar el corte de colas.

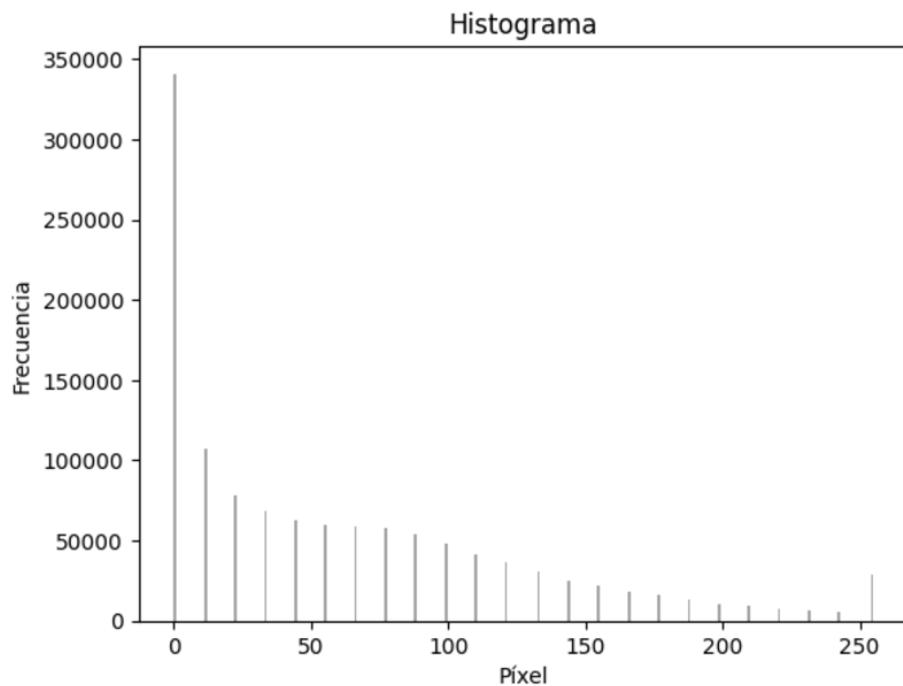


Figura 27: Histograma tras el corte de colas

Este resultado del histograma es esperable. Tenemos que tener en cuenta que cuando aplicamos un corte de colas a una imagen, estamos eliminando valores extremos, y nuestra imagen tenía muchos (por abajo), y redistribuyendo el resto de valores. Ahora, el histograma más disperso refleja que los valores de intensidad ahora ocupan un rango más amplio, abarcando más niveles de gris o color. Esto ocurre porque los valores extremos (que pueden ser ruido o valores atípicos) ya no comprimen el rango dinámico de la imagen.

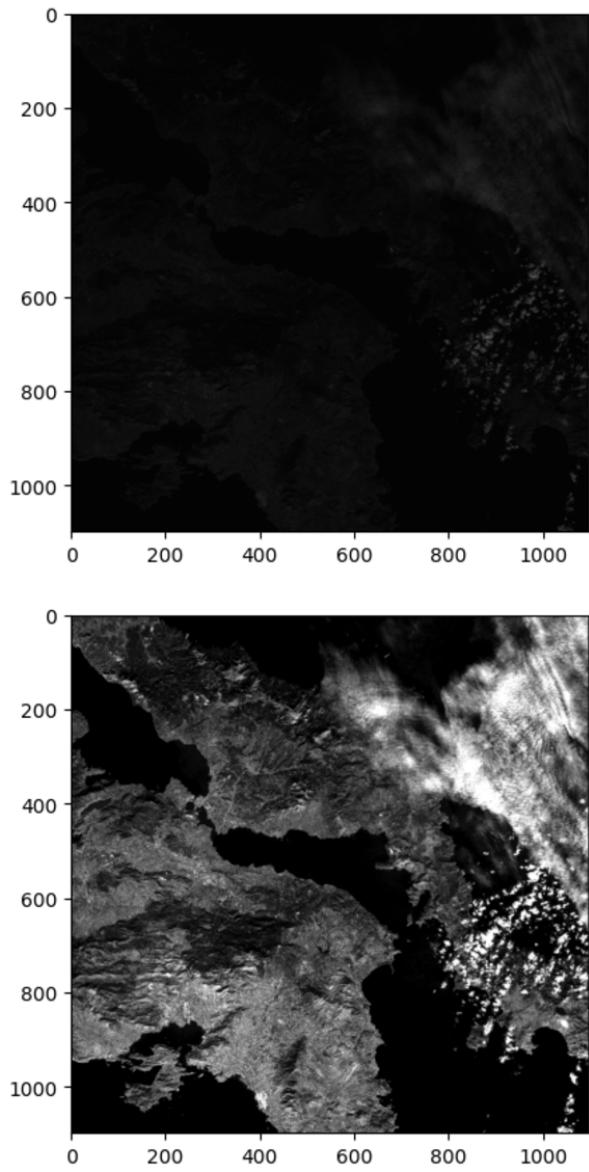


Figura 28: Corte de colas

La imagen se ve mucho más clara porque el corte de colas descarta los valores más oscuros y más claros (que pueden ser poco representativos) y luego escala los valores restantes a un rango más amplio. Esto aumenta el contraste general de la imagen, y de esta forma se ve algo mejor. De hecho, ya podemos empezar a intuir el país de Grecia, el mar, la ciudad de Atenas...

Por último en esta sección, realizaremos una ecualización del histograma.
[11]

Ecualización del histograma: Método para generar una nueva imagen cuyos niveles digitales sean proporcionales al valor y a su frecuencia de aparición en la imagen original.

```
1 def ecual(img):
2     resimg = img.copy()
3     val, freq = np.unique(img, return_counts=True)
4     frecuencia_acumulada = np.cumsum(freq)
5     num_pixeles = img.size
6     fe = 255 / num_pixeles
7     frecuencia_escalada = np.rint(fe * frecuencia_acumulada).astype(int)
8     dic = dict(zip(val, frecuencia_escalada))
9
10    for i in range(img.shape[0]):
11        for j in range(img.shape[1]):
12            resimg[i, j] = dic[img[i, j]]
13
14    return resimg
15
16 red_ecual = ecual(red_diez)
```

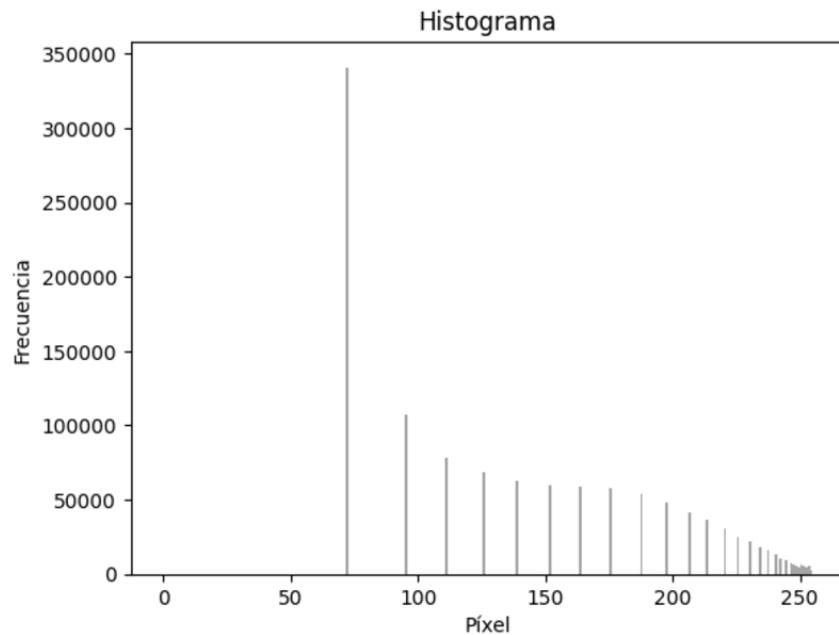


Figura 29: Histograma tras la ecualización del histograma

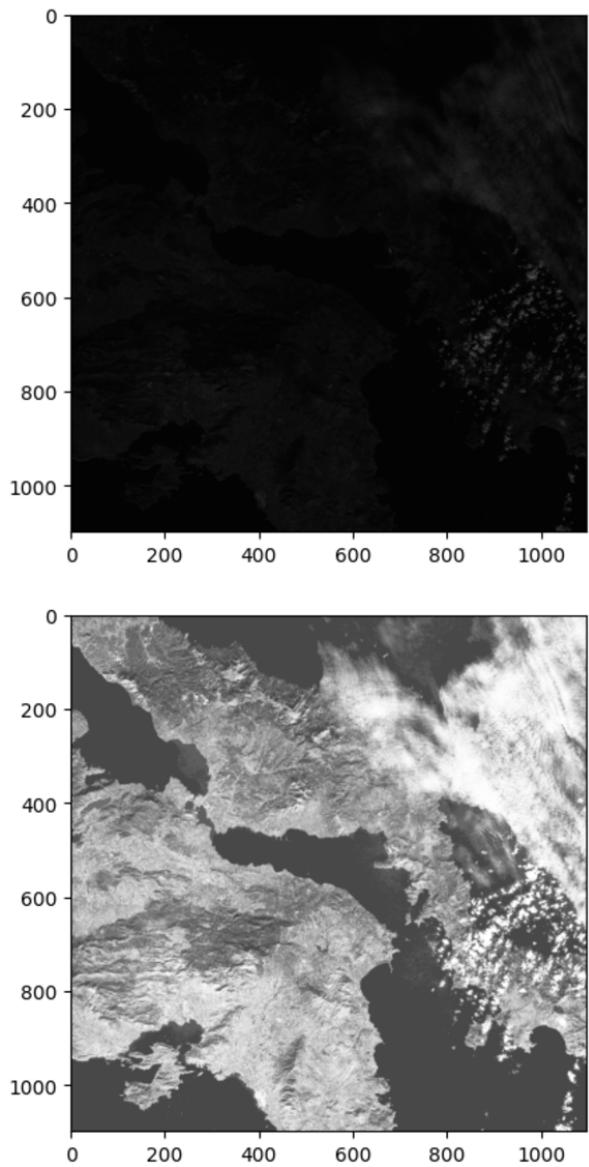


Figura 30: Ecualización del histograma

En primer lugar, obtenemos que la imagen resultante tiene mayor contraste. Esto se debe a que la ecualización expande rangos de intensidad que antes estaban comprimidos, y redistribuye las frecuencias para que todas las intensidades tengan una representación más uniforme [11]. El resultado es una mayor separación entre tonos similares, haciendo que las áreas más oscuras y las claras se distingan con mayor facilidad.

Además, recordemos que nuestra imagen tiene la mayoría de los píxeles en intensidades bajas, por lo que la ecualización incrementa estas intensidades para distribuirlas a lo largo del rango completo [0, 255].

7.2. Filtros

En esta sección, vamos a aplicar distintos filtros a nuestra imagen, con el fin de obtener más información y destacar aspectos relevantes de la misma. Utilizaremos la imagen tras el corte de colas, pues a partir de ella podremos obtener resultados más visuales que con la original, y en menor tiempo.

Comencemos definiendo qué es un filtro de una imagen: [12]

Filtro: Operación matemática aplicada a una imagen para resaltar, suavizar o modificar ciertas características, como bordes, texturas o ruido. En procesamiento de imágenes, los filtros se utilizan para transformar los valores de los píxeles en función de los valores de sus vecinos, siguiendo una matriz o kernel que define cómo se realiza esta transformación.

El código para aplicar un filtro a nuestra imagen es el siguiente: [20]

```
1 def filtro(img, cf):
2     if cf.shape[0] != cf.shape[1]:
3         raise ValueError("La matriz de filtro tiene que ser cuadrada")
4
5     resimg = img.copy()
6     num_filas, num_columnas = img.shape
7     resimg = np.zeros((num_filas, num_columnas))
8     cf_suma = np.sum(cf)
9
10    for i in range(1, num_filas - 1):
11        for j in range(1, num_columnas - 1):
12            suma = 0
13            for k in range(-1, 2):
14                for l in range(-1, 2):
15                    suma += img[i+k, j+l] * cf[k+1, l+1]
16            resimg[i, j] = round(suma/cf_suma)
17
18    plt.figure(figsize=(10, 5))
19    plt.subplot(1, 2, 1)
20    plt.title("Imagen original")
21    plt.imshow(img, cmap='gray')
22    plt.axis('off')
23
24    plt.subplot(1, 2, 2)
25    plt.title("Imagen después del filtro")
26    plt.imshow(resimg, cmap='gray')
```

```

27     plt.axis('off')
28
29     plt.show()
30
31     return resimg

```

Ahora iremos llamando a esta función con diferentes filtros sobre nuestra imagen:

```

1 # Filtro de suavizado:
2 cf1 = np.array([
3     [1/9, 1/9, 1/9],
4     [1/9, 1/9, 1/9],
5     [1/9, 1/9, 1/9]
6 ])
7
8 resimg_suavizado = filtro(red_corte, cf1)

```

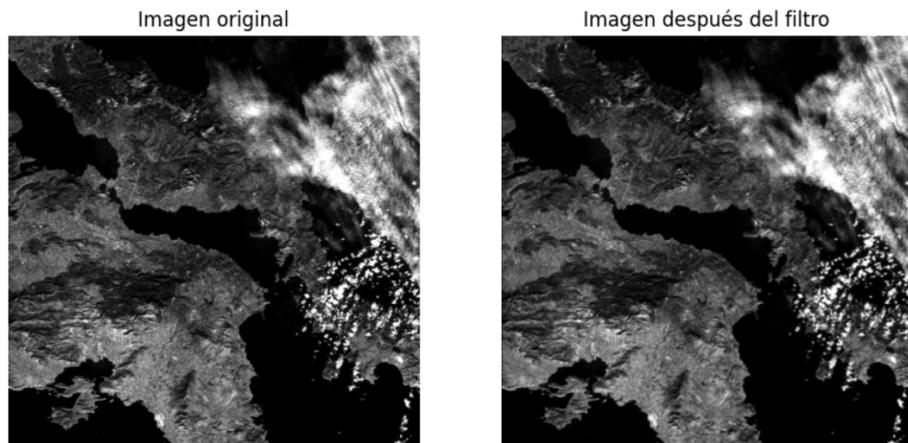


Figura 31: Filtro de suavizado

```

1 # Filtro de paso de alta:
2 cf2 = np.array([
3     [-1., -1., -1.],
4     [-1., 9, -1.],
5     [-1., -1., -1.]
6 ])
7
8 resimg_pasoalta = filtro(red_corte, cf2)

```

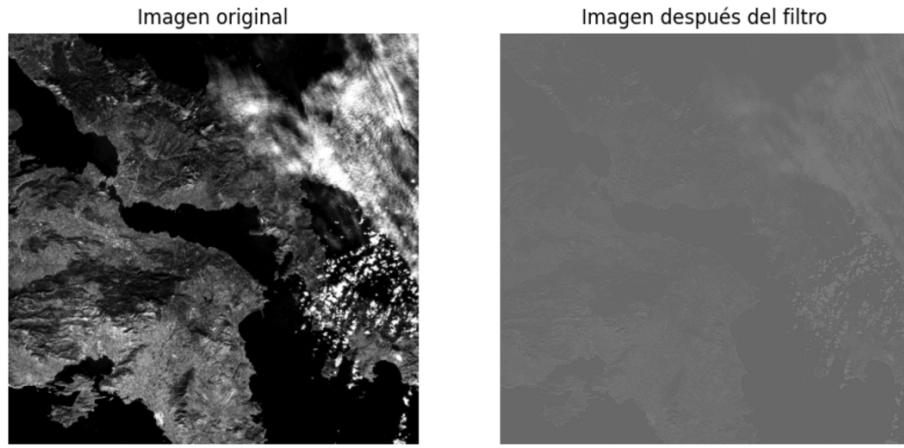


Figura 32: Filtro de paso de alta

```

1 # Filtro de realce:
2 cf3 = np.array([
3     [0, -1., 0],
4     [-1., 5., -1.],
5     [0, -1., 0]
6 ])
7
8 resimg_realce = filtro(red_corte, cf3)

```

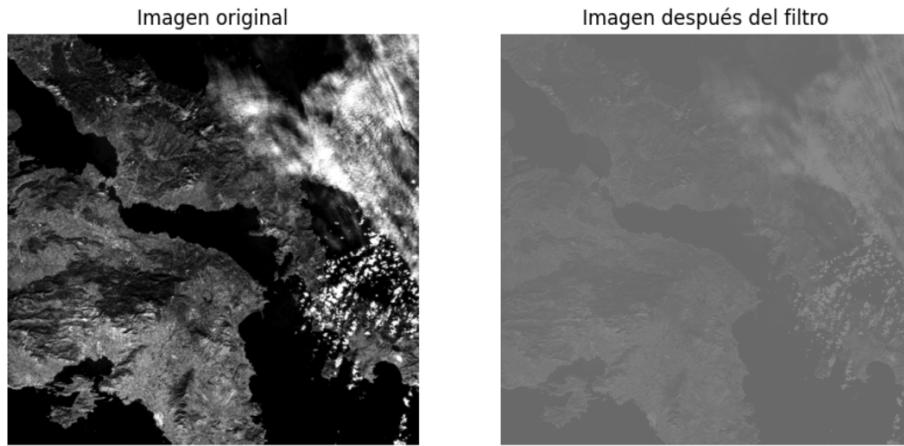


Figura 33: Filtro de realce

Cada uno de estos filtros tiene una funcionalidad diferente, en función de la característica de la imagen que queramos obtener.

También hay muchos filtros no lineales que podemos aplicar, pero quizás el más interesante sea el filtro detector de bordes sobel: [12]

Sobel: Mide la variación entre filas y columnas de la matriz de filtrado.

```
1 def sobel(img):
2     num_filas, num_columnas = img.shape
3     resimg = np.zeros((num_filas, num_columnas))
4     for i in range(1, num_filas - 1):
5         for j in range(1, num_columnas - 1):
6             C = (img[i-1, j+1] + 2 * img[i, j+1] + img[i+1, j+1])
7                 - (img[i-1, j-1] + 2 * img[i, j-1] + img[i+1, j-1])
8             F = (img[i-1, j-1] + 2 * img[i-1, j] + img[i-1, j+1])
9                 - (img[i+1, j-1] + 2 * img[i+1, j] + img[i+1, j+1])
10            resimg[i, j] = round(np.sqrt(C**2 + F**2))
11
12    plt.figure(figsize=(10, 5))
13    plt.subplot(1, 2, 1)
14    plt.title("Imagen original")
15    plt.imshow(img, cmap='gray')
16    plt.axis('off')
17
18    plt.subplot(1, 2, 2)
19    plt.title("Imagen después del filtro de Sobel")
20    plt.imshow(resimg, cmap='gray')
21    plt.axis('off')
22
23    plt.show()
24
25    return resimg
```

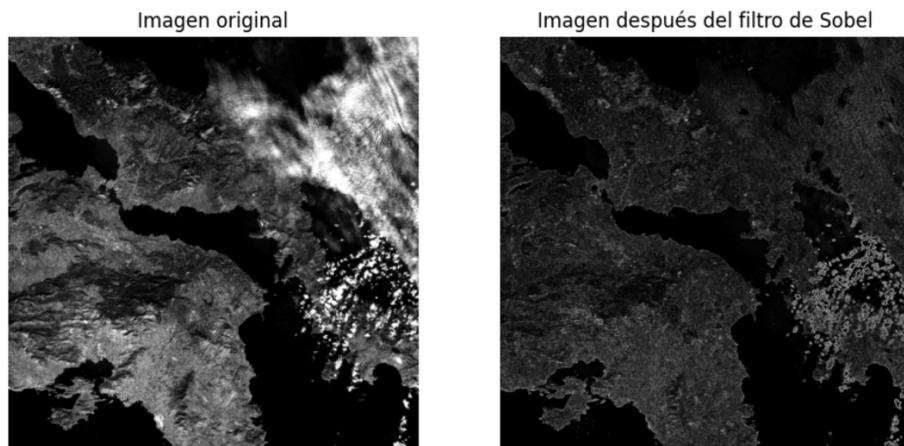


Figura 34: Filtro sobel

Con este filtro podemos ver mejor los bordes. Sin embargo, es cierto que no se aplica del todo bien. Pese a que parece resaltar algo más los bordes de la costa, también realza las nubes. Pero en cualquier caso es un filtro muy útil en teledetección que, quizás, aplicado a otras bandas o a la imagen original podrá ser de mucha ayuda.

7.3. Escalado

Lo primero que podemos hacer respecto al escalado es el **diezmado** de la imagen. Sin embargo, como comentamos antes, lo hicimos lo primero para poder usar imágenes de menor calidad y que tardase menos tiempo en ejecutar.

Otra función que podemos hacer es el efecto contrario: **ampliar** la imagen. No vamos a ejecutarlo pues tarda demasiado (y además, no nos es útil en estos momentos), pero el código Python para hacerlo es el siguiente: [21]

```

1 def amplia(img, R=2):
2     num_filas, num_columnas = img.shape
3     num_filas_res = R * num_filas
4     num_columnas_res = R * num_columnas
5
6     resimg = np.zeros((num_filas_res, num_columnas_res))
7
8     resimg[::R, ::R] = img # valores originales
9
10    # filas:
11    for i in range(1, num_filas_res-1, R):
12        for j in range(num_columnas_res):
13            resimg[i, j] = (resimg[i - 1, j] + resimg[i + 1, j]) / 2
14
15    # columnas:
16    for i in range(0, num_filas_res, R):
17        for j in range(1, num_columnas_res-1, R):
18            resimg[i, j] = (resimg[i, j - 1] + resimg[i, j + 1]) / 2
19
20    resimg[-1, :] = resimg[-2, :] # última fila
21    resimg[:, -1] = resimg[:, -2] # última columna
22
23    return resimg

```

Lo último que vamos a ver en esta sección es la aplicación de máscaras a nuestra imagen. Para ello, definiremos dos funciones en Python: **mascara** y **calculaMask**: [21]

```

1 def mascara(img, mask):
2     resimg = img * mask
3     return resimg

```

```

1 def calculaMask(img, NDmin, NDmax):
2     mask = np.zeros(img.shape, dtype=np.int16)
3     mask[(img > NDmin) & (img < NDmax)] = 1
4     return mask

```

Vamos a aplicar dos máscaras diferentes: una para realzar las luces y otra para las nubes:

```

1 mask_luces = calculaMask(red_corte, 100, 180)
2 msk_img_luces = mascara(red_corte, mask_luces)
3 plt.imshow(msk_img_luces, cmap='gray')

```

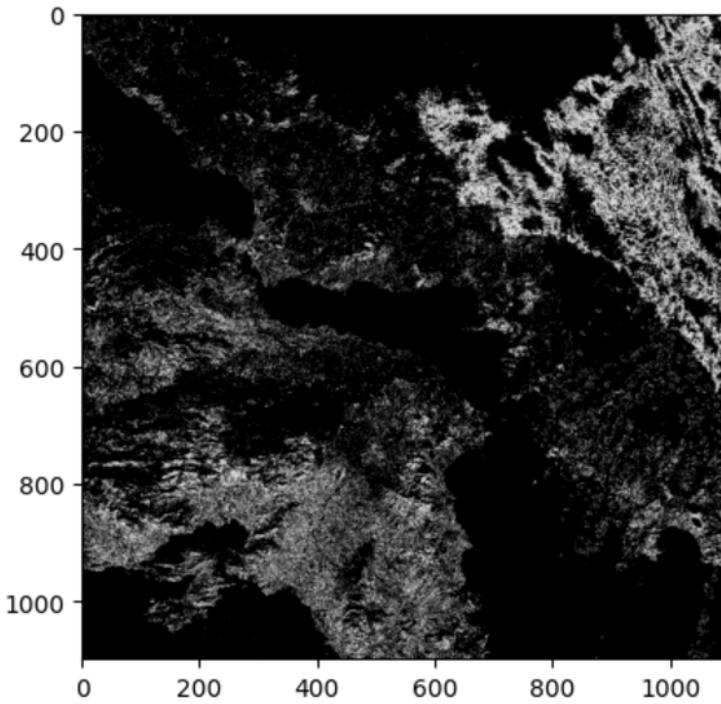


Figura 35: Máscara de luces

Podemos ver cómo se destacan los valores altos de la imagen, que corresponden a las luces. También se destacan mucho las nubes de la parte superior derecha. Si queremos destacar únicamente esta característica, podemos ampliar el rango de la máscara:

```

1 mask_nubes = calculaMask(red_corte, 200, 255)
2 msk_img_nubes = mascara(red_corte, mask_nubes)
3 plt.imshow(msk_img_nubes, cmap='gray')

```

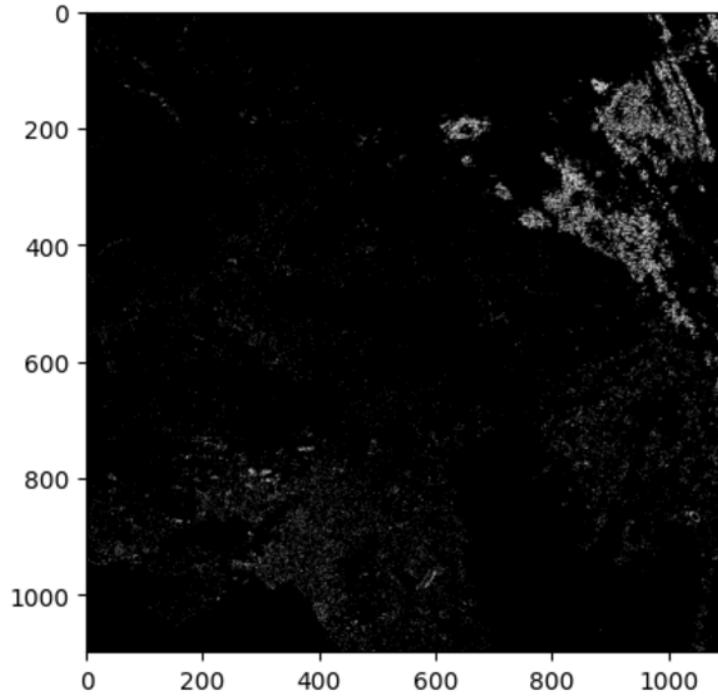


Figura 36: Máscara de nubes

De esta forma, con la simple aplicación de una máscara, podemos obtener dónde aparecen las nubes en nuestra banda.

7.4. Clasificación

Para esta última sección de transformaciones de una imagen, utilizaremos directamente la imagen RGB. Esta imagen podemos verla en la Figura 7.

Para la clasificación, usaremos las dos siguientes funciones:

- **Función isodata:** Clasifica los píxeles de una imagen en C clases de forma iterativa.
- **Función seudo:** Convierte una imagen clasificada en una representación visual pseudo-colorizada.

El código Python de estas funciones para llevar a cabo la clasificación de nuestra imagen es el siguiente: [22]

```

1 def isodata(img, C):
2     max_iter = 100
3     tol = 0.001
4

```

```

5   # Paso 1: Elegir aleatoriamente los centros iniciales de las clases
6   pixeles_no_negros = img[np.any(img > 0, axis=-1)].reshape(-1, img.shape[2])
7   indices = np.random.choice(pixeles_no_negros.shape[0], size=C, replace=False)
8   v = pixeles_no_negros[indices].astype(float) # Centros aleatorios
9
10  resimg = np.zeros((img.shape[0], img.shape[1]), dtype=int)
11
12  for it in range(max_iter):
13      # Paso 2: Clasificar cada píxel al centro más cercano
14      for i in range(img.shape[0]):
15          for j in range(img.shape[1]):
16              if np.all(img[i, j] == 0): # Píxel negro
17                  resimg[i, j] = 0
18              else:
19                  distancias = np.linalg.norm(img[i, j] - v, axis=1) # Distancia euclídea
20                  resimg[i, j] = np.argmin(distancias) + 1 # Clase c+1 para el resto
21
22      # Paso 3: Valor medio de los puntos asignados a cada clase
23      nuevos_centros = np.zeros_like(v) # Inicializamos los nuevos centros
24      conteo_pixeles = np.zeros(C, dtype=int) # Contadores para cada clase
25
26      for i in range(img.shape[0]):
27          for j in range(img.shape[1]):
28              if resimg[i, j] == 0:
29                  continue # Ignoramos píxel negro
30              cluster_actual = resimg[i, j] - 1 # Convertir clase c+1 a índice c
31              nuevos_centros[cluster_actual] += img[i, j]
32              conteo_pixeles[cluster_actual] += 1
33
34      # Actualizar los centros dividiendo por el número de píxeles en cada clase
35      for c in range(C):
36          if conteo_pixeles[c] > 0:
37              nuevos_centros[c] /= conteo_pixeles[c]
38          else:
39              # Si un cluster queda vacío, mantener su centro anterior
40              nuevos_centros[c] = v[c]
41
42      # Paso 4: Convergencia
43      if np.linalg.norm(nuevos_centros - v) < tol:
44          break
45
46      # Paso 5: Actualizar los centros para la siguiente iteración
47      v = nuevos_centros
48
49  return resimg

1 def seudo(img):
2     resimg = np.zeros((img.shape[0], img.shape[1], 3), np.uint8)
3     i = 0
4     while i < img.shape[0]:

```

```

5         j = 0
6     while j < img.shape[1]:
7         if img[i, j] == 0:
8             resimg[i, j, :] = [255, 0, 0]
9         elif img[i, j] == 1:
10            resimg[i, j, :] = [0, 255, 0]
11        elif img[i,j] == 2:
12            resimg[i, j, :] = [0, 0, 255]
13        elif img[i, j] == 3:
14            resimg[i, j, :] = [255, 255, 0]
15        elif img[i, j] == 4:
16            resimg[i, j, :] = [0, 255, 255]
17        elif img[i, j] == 5:
18            resimg[i, j, :] = [255, 0, 255]
19        else:
20            resimg[i, j, :] = [0, 0, 0]
21        j = j+1
22    i = i+1
23 return resimg

```

Veamos cómo se clasifica nuestra imagen en 3 clases:

```

1
2 imagen_rgb_diez = diezmado(imagen_rgb)
3
4 res = isodata(imagen_rgb_diez, 3)
5 res_seudo = seudo(res)
6 plt.figure(1)
7 plt.imshow(res_seudo)

```

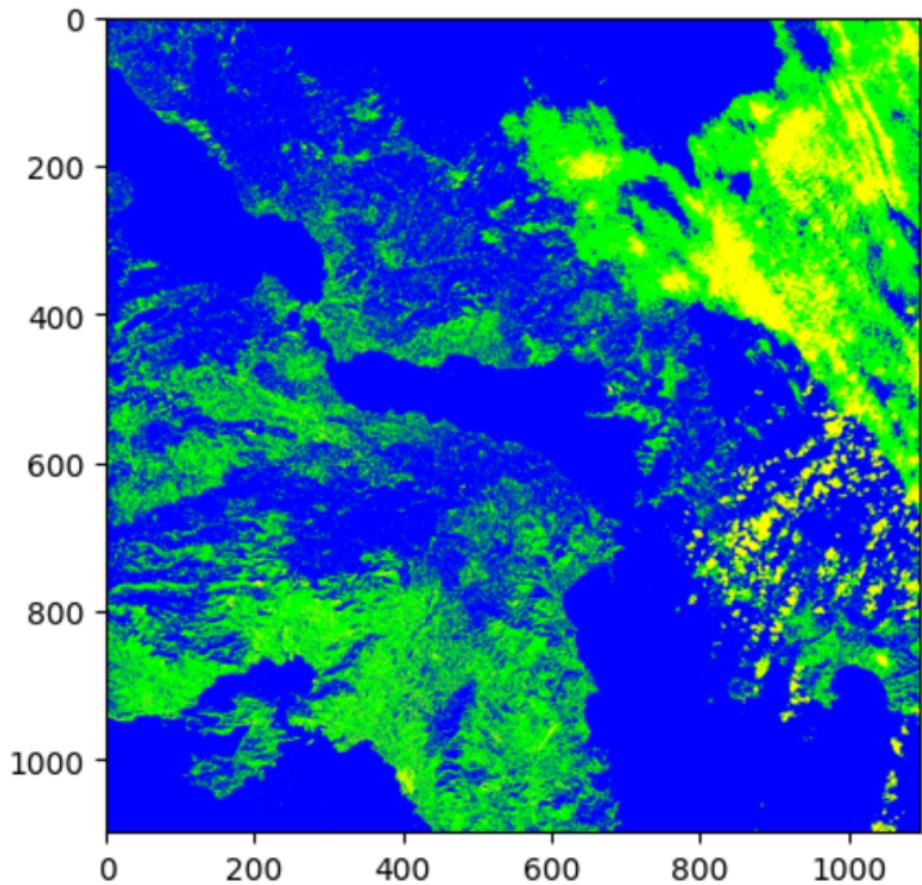


Figura 37: Clasificación de la imagen RGB

Podemos ver que clasifica la imagen en tres categorías aproximadamente: nubes, baja altitud y alta altitud. Es decir, en una clase representa las nubes (amarillo), en otra las zonas cercanas en altura al nivel del mar, mar incluido (azul), y en otra las zonas de mayor altura, es decir, los picos de las zonas montañosas (verde).

8. Conclusiones

Finalmente, vamos a dar una serie de conclusiones del trabajo y la asignatura en general.

En primer lugar, hemos podido comprobar que trabajar con imágenes satelitales no es fácil y requiere tiempo. Es importante entender en todo momento qué estamos haciendo, cómo funcionan las bandas, y no escribir código sin saber realmente qué hace. Pese a ello, una vez se entiende no es difícil y es muy entretenido. Hay bastante información en internet sobre cómo trabajar con este tipo de archivos, por lo que podemos recurrir a ella si estamos perdidos en alguna ocasión.

Por otra parte, trabajamos con imágenes de gran tamaño como hemos descrito en secciones anteriores. Esto dificulta y ralentiza el proceso, pues muchas funciones tardan demasiado en ejecutarse.

Los resultados obtenidos son buenos, hemos podido experimentar de primera mano cómo obtener mucha información a partir de una imagen satelital, y analizar a partir de ella diferentes aspectos.

Respecto a la asignatura en general, creo que es muy buena. Se aprende mucho y no requiere de tanto estudio como otras de la titulación, lo cual se agradece. Esto causa que se vaya a clase con más ganas de aprender de verdad, y no por obligación como otras veces pasa.

Bibliografía

- [1] Adobe. *Procesamiento de archivos de imagen RAW de cámara*. 2024. URL: <https://helpx.adobe.com/es/photoshop-elements/using/processing-camera-raw-image-files.html>.
- [2] Benowu. *¿Por qué deberías fotografiar en formato raw?* 2021. URL: <https://www.benowu.com/por%5C-que%5C-deberias%5C-fotografiar%5C-en%5C-formato%5C-raw/>.
- [3] Revista Contacto. *Imágenes satelitales, la importancia de una nueva lectura*. 2022. URL: <https://revistacontacto.uniandes.edu.co/contacto%5C-25%5C-cadena%5C-aeroespacial/imagenes%5C-satelitales%5C-la%5C-importancia%5C-de%5C-una%5C-nueva%5C-lectura/>.
- [4] Francisco Javier Dávila Martínez. «Georreferenciación de documentos cartográficos para la gestión de Archivos y Cartotecas». En: *Instituto Geográfico Nacional* (2012). Artículo de georreferenciación.
- [5] Programme of the European Union. *Copernicus Data Space Ecosystem*. 2024. URL: <https://dataspace.copernicus.eu/>.
- [6] Programme of the European Union. *Primera imagen seleccionada para el trabajo*. 2024. URL: [https://zipper.dataspace.copernicus.eu/odata/v1/Products%5C\(b621109b%5C-fee5%5C-46aa%5C-a0a1%5C-bb6b2ccb8337%5C\)/%5C\\$value](https://zipper.dataspace.copernicus.eu/odata/v1/Products%5C(b621109b%5C-fee5%5C-46aa%5C-a0a1%5C-bb6b2ccb8337%5C)/%5C$value).
- [7] Programme of the European Union. *Segunda imagen seleccionada para el trabajo*. 2024. URL: [https://zipper.dataspace.copernicus.eu/odata/v1/Products%5C\(e2c2e64a%5C-2845%5C-400b%5C-b38d%5C-293d3a63b26f%5C\)/%5C\\$value](https://zipper.dataspace.copernicus.eu/odata/v1/Products%5C(e2c2e64a%5C-2845%5C-400b%5C-b38d%5C-293d3a63b26f%5C)/%5C$value).
- [8] Programme of the European Union. *Sentinel-2*. 2024. URL: <https://dataspace.copernicus.eu/explore%5C-data/data%5C-collections/sentinel%5C-data/sentinel%5C-2>.
- [9] Juan Antonio Castro García. «Tema 3: Plataformas de teledetección espacial». En: *Universidad de Sevilla* (2024). Apuntes Teledetección.
- [10] Juan Antonio Castro García. «Tema 4: Procesado y corrección de la imagen». En: *Universidad de Sevilla* (2024). Apuntes Teledetección.
- [11] Juan Antonio Castro García. «Tema 5A: Realces y mejoras». En: *Universidad de Sevilla* (2024). Apuntes Teledetección.
- [12] Juan Antonio Castro García. «Tema 5B: Transformaciones analíticas de una imagen». En: *Universidad de Sevilla* (2024). Apuntes Teledetección.
- [13] Juan Antonio Castro García. «Tema 6A: Transformaciones de una imagen». En: *Universidad de Sevilla* (2024). Apuntes Teledetección.
- [14] Geoinnova. *Correcciones radiométrica y geográfica en imágenes Radar*. 2023. URL: <https://geoinnova.org/blog%5C-territorio/sentinel%5C-1/>.

- [15] Gis y Beers. *Todo lo que deberías saber sobre imágenes Sentinel 2*. 2016. URL: <https://www.gisandbeers.com/lo%5C-deberias%5C-saber%5C-imagenes%5C-sentinel%5C-2/>.
- [16] David Suárez Moreno Jairo Escánez García. «Teledetección para hallar restos arqueológicos». En: *Universidad de Sevilla* (2024). Trabajo voluntario Teledetección.
- [17] Francisco Javier Gómez Pulido José Ignacio Meseguer Gómez. «Seguimiento de especies en peligro de extinción con Teledetección». En: *Universidad de Sevilla* (2024). Trabajo voluntario Teledetección.
- [18] Jupyter Lab. *JupyterLab Documentation*. 2024. URL: <https://jupyterlab.readthedocs.io/en/latest/>.
- [19] Francisco Javier Gómez Pulido. «Práctica 2: Histogramas». En: *Universidad de Sevilla* (2024). Prácticas Teledetección.
- [20] Francisco Javier Gómez Pulido. «Práctica 3: Filtros». En: *Universidad de Sevilla* (2024). Prácticas Teledetección.
- [21] Francisco Javier Gómez Pulido. «Práctica 4: Escalado». En: *Universidad de Sevilla* (2024). Prácticas Teledetección.
- [22] Francisco Javier Gómez Pulido. «Práctica 6: Clasificación». En: *Universidad de Sevilla* (2024). Prácticas Teledetección.
- [23] Tourinews. *El 37 por ciento de los bosques de Ática (Atenas) se han quemado en los últimos 8 años*. 2024. URL: https://www.tourinews.es/destinos%5C-turismo/bosques%5C-atica%5C-atenas%5C-quemados%5C-ultimos%5C-8%5C-anos%5C_4482759%5C_102.html.