UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Geodésica, Cartográfica y Topográfica

Cartografiado de paneles solares mediante Deep Learning

Trabajo Fin de Máster

Máster Universitario en Ingeniería Geomática y Geoinformación

AUTOR/A: Li, Bofeng

Tutor/a: Recio Recio, Jorge Abel

Cotutor/a: Crespo Peremarch, Pablo

CURSO ACADÉMICO: 2023/2024

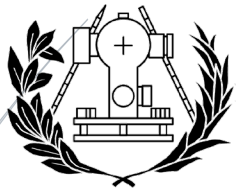
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

MUIGG 2023-2024

Cartografiado de panel solares mediante Deep Learning

Trabajo fin de máster.

Máster Universitario en Ingeniería Geomática y Geoinformación



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA GEODÉSICA CARTOGRÁFICA Y TOPOGRÁFICA

Autor: Li, Bofeng

Tutor: Recio Recio, Jorge Abel

Cotutor: Crespo Peremarch, Pablo

Curso Académico: 2023-2024

AGRADECIMIENTOS

En primer lugar, me gustaría agradecer a mi tutor D. Jorge Abel Recio Recio y cotutor D. Pablo Crespo Peremarch, quienes me han dado mucha ayuda en la realización de mi trabajo de fin de máster.

Como estudiante extranjero, también me gustaría agradecer a mis profesores de español, quienes también me brindaron mucha ayuda.

Además, quiero agradecer a mis compañeros y amigos, todos me dieron mucho apoyo, sin su ayuda no podría completar este trabajo.

COMPROMISO

"El presente documento ha sido realizado completamente por el firmante; no ha sido entregado como otro trabajo académico previo y todo el material tomado de otras fuentes ha citado su origen en el texto, así como referenciado en la bibliografía"

Resumen:

La energía solar es la fuente de energía renovable más disponible[1]. España, como el séptimo país del mundo en términos de generación de energía fotovoltaica, instala una gran cantidad de nuevos paneles solares cada año[2]. Por lo tanto, es necesario utilizar técnicas de teledetección para monitorear la distribución y el crecimiento de los paneles solares fotovoltaicos. Mediante la aplicación de redes neuronales convolucionales se creará un modelo clasificador que identifique las placas solares en ortofotografías de 25cm de resolución espacial. Para generar un conjunto de muestras de entrenamiento y evaluación se digitalizarán manualmente todas las instalaciones de placas solares en el término municipal de L'Eliana (Valencia). Mediante análisis espacial se determinarán los fragmentos de las ortofotografías en los que existan placas solares. A partir de este conjunto de muestras se compararán diversos parámetros que influyan en la calidad del modelo clasificador para obtener el modelo óptimo. Se evaluará el modelo obtenido analizando sus fortalezas y debilidades. Dicho modelo se podrá aplicar en otros municipios y/o en años distintos al analizado para conocer la evolución de las instalaciones de placas solares.

Palabras clave:

teledetección, deep learning, CNN, paneles solares, fotovoltaica, U-Net

[1] Gielen, D., Boshell, F., Saygin, D., Bazilian, M. D., Wagner, N., & Gorini, R. (2019). The role of renewable energy in the global energy transformation. Energy Strategy Reviews, 24, 38–50.

[2] Kruitwagen, L., Story, K. T., Friedrich, J., Byers, L., Skillman, S., & Hepburn, C. (2021). A global inventory of photovoltaic solar energy generating units. Nature, 598(7882), 604–610.

Abstract:

Solar energy is the most available renewable energy source. Spain, as the seventh country in the world in terms of photovoltaic energy generation, installs a large number of new solar panels each year. Therefore, it is necessary to use remote sensing techniques to monitor the distribution and growth of photovoltaic solar panels. Applying convolutional neural networks will create a classifier model to identify solar panels in orthoimages with a spatial resolution of 25 cm. All solar panel installations in the municipality of L'Eliana (Valencia) will be manually digitized to generate a training and evaluation sample set. Spatial analysis will determine the fragments of orthoimages where solar panels exist. Various parameters influencing the quality of the classifier model will be compared from this sample set to obtain the optimal model. The obtained model will be evaluated by analyzing its strengths and weaknesses. This model can be applied to other municipalities and/or different years from the one analyzed to understand the evolution of solar panel installations.

Keywords:

remote sensing, deep learning, CNN, solar panel, photovoltaic, U-Net

Resum:

L'energia solar és la font d'energia renovable més disponible. Espanya, com el setè país del món en termes de generació d'energia fotovoltaica, instal·la una gran quantitat de nous panells solars cada any. Per tant, és necessari utilitzar tècniques de teledetecció per monitoritzar la distribució i el creixement dels panells solars fotovoltaics. Mitjançant l'aplicació de xarxes neuronals convolucionals es crearà un model classificador que identifique les plaques solars en ortoimatges de 25 cm de resolución espacial. Per generar un conjunt de mostres d'entrenament i avaluació es digitalitzaran manualment totes les instal·lacions de plaques solars en el terme municipal de L'Eliana (València). Mitjançant anàlisi espacial es determinaran els fragments de les ortoimatges en els quals existisquen plaques solars. A partir d'aquest conjunt de mostres es compararan diversos paràmetres que influeixen en la qualitat del model classificador per obtindre el model òptim. S'avaluarà el model obtingut analitzant les seues fortaleses i debilitats. Aquest model es podrà aplicar en altres municipis i/o en anys diferents de l'analitzat per conéixer l'evolució de les instal·lacions de plaques solars.

Paraules clau:

teledetecció, deep learning, xarxes neuronals convolucionals, panells solars, fotovoltaica, U-Net

Índice de figuras:

Índice de tablas:

Índice

1 Introducción

La energía solar es la fuente de energía renovable más disponible. La capacidad de generación de energía solar fotovoltaica (PV) ha crecido un 41 por ciento por año desde 2009 [1]. Las proyecciones del sistema energético que mitigan el cambio climático y ayudan al acceso universal a la energía muestran un aumento de casi diez veces en la capacidad de generación de energía solar fotovoltaica para 2040 [2][3].

España, como el séptimo país del mundo en términos de generación de energía fotovoltaica, se instala una gran cantidad de nuevos paneles solares cada año[4]. Por lo tanto, es necesario utilizar técnicas de teledetección para monitorear la distribución y el crecimiento de los paneles solares fotovoltaicos.

La mayoría de los estudios relacionados con la distribución de paneles solares fotovoltaicos utilizan imágenes satelitales (como Sentinel, Landsat, etc.) mediante el aprendizaje profundo o el aprendizaje automático para determinar su distribución [5][6]. De esta manera, se puede identificar rápidamente los paneles solares fotovoltaicos en todo el mundo. Sin embargo, debido a la baja resolución espacial de las imágenes satelitales, los datos satelitales comunes, como los del satélite Sentinel-2, tienen una resolución espacial de 10 metros, mientras que el satélite Landsat-8 tiene una resolución espacial de 30 metros. El uso de estos datos generalmente solo permite detectar grandes áreas de plantas solares fotovoltaicas, que suelen estar ubicadas en campos y áreas agrícolas[4][5]. El uso de estos datos no permite detectar pequeños paneles solares domésticos instalados en los tejados de los edificios.

En España, existe el servicio PNOA (Plan Nacional de Ortofotografía Aérea) que proporciona ortofotografías de alta resolución actualizadas regularmente para satisfacer las necesidades de información geográfica a nivel nacional y regional. Este servicio puede ofrecer ortofotografías con una resolución espacial de al menos de 25 cm [7], lo cual es suficiente para identificar pequeños paneles solares domésticos en áreas urbanas. En la Comunidad Valenciana, el ICV (Institut Cartogràfic Valencià) también proporciona ortofotografías con una resolución espacial de 25 cm, pero estas se actualizan anualmente [8], a diferencia del plan nacional PNOA, que se actualiza cada tres años [7]. Por lo tanto, en este trabajo fin de máster (TFM) se ha optado por utilizar las ortofotografías proporcionadas por el ICV para la identificación de paneles solares en áreas urbanas.

En este TFM, se crearán dos modelos de redes neuronales convolucionales (CNN) para identificar paneles solares en ortofotografías. Para entrenar y evaluar los modelos, se ha realizado una vectorización manual de todos los paneles solares en la zona de L'Eliana (Valencia). A través del análisis y la evaluación del rendimiento de los modelos,

se ajustarán y analizarán varias veces las muestras utilizadas en el entrenamiento para obtener el mejor modelo posible. Además, se analizarán sus ventajas y desventajas. Se intentará utilizar el modelo generado para identificar la ubicación de los paneles solares en ortofotografías de otras áreas y años diferentes a los utilizados en el entrenamiento.

2 Objetivo

El objetivo de este trabajo final de master es profundizar en el aprendizaje de conceptos relacionados con el Deep Learning. Para ello, se entrenará un modelo CNN binario utilizando redes neuronales convolucionales para reconocimiento y un modelo U-Net para generar máscaras. Estos modelos se utilizarán para identificar automáticamente los paneles solares fotovoltaicos en la zona de estudio. Finalmente, se realizará un análisis de datos sobre los cambios en los paneles solares fotovoltaicos utilizando los resultados obtenidos.

3 Plataforma y software

3.1 Colab

3.1.1. Introducción de Colab

Google Colab (*Google Colaboratory*) es una plataforma de computación en la nube gratuita. Esta plataforma ofrece un entorno Jupyter Notebook, permitiendo así que todos los usuarios puedan escribir y ejecutar código Python directamente en la nube a través del navegador. Esto nos permite usar Colab en cualquier dispositivo. Además de en computadoras portátiles y de escritorio, también podemos usar Colab en tabletas y teléfonos móviles[9][10].

Colab es especialmente adecuado para el aprendizaje automático, el análisis de datos y fines educativos, ya que combina la conveniencia de la computación en la nube con la potente funcionalidad de Jupyter Notebook. Colab permite a los usuarios utilizar de forma gratuita las GPU (unidades de procesamiento gráfico) y TPU (unidades de procesamiento tensorial) de Google, lo que resulta especialmente beneficioso para entrenar modelos de aprendizaje profundo complejos. Realizar estas operaciones en la nube a través de Colab acelera significativamente la velocidad de cálculo y elimina la necesidad de adquirir equipos de hardware costosos [9][10].

Colab incluye muchas de las bibliotecas de Python más utilizadas, como TensorFlow, Keras, PyTorch, Pandas, NumPy, entre otras, lo que permite a los usuarios usarlas directamente sin necesidad de instalación manual. Esto proporciona una gran comodidad, ya que configurar el entorno para ejecutar TensorFlow y Keras suele ser una tarea compleja.

Los usuarios pueden integrar Colab con Google Drive para guardar y compartir fácilmente archivos de cuadernos (. ipynb). En Colab se puede importar directamente Google Drive para acceder a los datos o archivos almacenados allí y utilizarlos directamente para entrenar modelos y otras operaciones. Al mismo tiempo, los archivos generados por los scripts escritos en la nube también se pueden guardar directamente en Google Drive [11].

Además, en Colab se utilizan archivos en formato ". ipynb" (*Jupyter Notebook*). Es un formato que permite crear múltiples scripts en una sola página web y podemos insertar texto o imágenes al mismo archivo. Lo que facilita la comprensión rápida y el registro del propósito de los scripts. Al ejecutar este formato de archivo en la web, si tenemos múltiples scripts, los scripts posteriores pueden usar directamente las variables generadas por los scripts anteriores. Es porque se almacenan temporalmente en la memoria de la nube de Colab. Esto resulta muy conveniente para tareas repetitivas como la generación de gráficos. Por ejemplo, después de entrenar un modelo, al evaluarlo, se suele generar un gráfico de pérdida de entrenamiento y validación. Sin embargo, antes de ver el gráfico, puede ser difícil definir correctamente el rango del eje y. Con este formato de archivo, se puede añadir un script adicional a continuación sin necesidad de ejecutar todos los scripts desde el principio [12].

Además de poder usar TPU y GPU gratuitas, también se pueden comprar unidades de cómputo para utilizar TPU y GPU de mayor rendimiento. Las TPU y GPU de diferentes capacidades consumen distintas cantidades de unidades de cómputo, generalmente oscilando entre 1 y 10 unidades de cómputo por hora. Algunas TPU o GPU permiten la opción de usar un modo de alta memoria, que proporciona más memoria operativa, pero también consume más unidades de cómputo adicionales. En Europa, el costo de adquirir Colab individualmente es de 11,29 euros por cada 100 unidades de cómputo. Al suscribirse a Colab Pro o Colab Pro+, se obtienen 100 o 500 unidades de cómputo mensuales, respectivamente. Las unidades de cómputo compradas se mantendrán válidas por 3 meses [13].

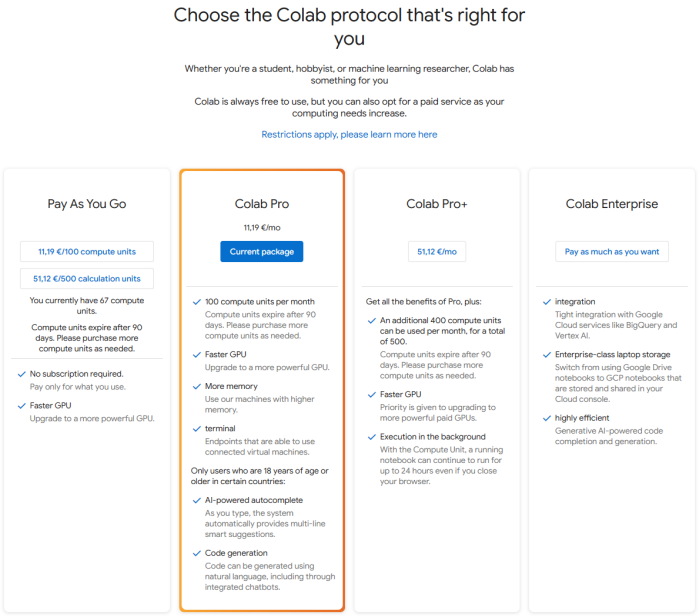


Figura 1 : Funciones y tarifas de la suscripción de pago de Colab

Fuente: Colab paid services pricing. (n.d.).

3.1.2. Trabajo en Colab

La mayoría de los scripts de este TFM se ejecutarán en Colab. Dado que necesitamos acceder a una gran cantidad de datos, hemos suscrito a Colab Pro para utilizar el modo de alta memoria y utilizar GPU para acelerar la velocidad de cálculo. Especialmente en el entrenamiento del modelo U-Net, el uso de GPU puede reducir significativamente el tiempo de entrenamiento.

Sin embargo, al ejecutar los scripts en Colab, también enfrentamos algunas dificultades, principalmente debido a las limitaciones de Google Drive. Si un directorio de Google Drive contiene una gran cantidad de archivos, esto puede provocar un tiempo de espera al cargar archivos en Colab. En nuestro caso, las ortofotografías de 25 cm de resolución espacial de nuestra área de estudio se han dividido en aproximadamente 500,000 subimágenes. La escritura y lectura de estas imágenes impacta negativamente en Google Drive. Aunque hemos probado usar un script para crear una subcarpeta para cada 5000 imágenes al escribirlas, esto sólo retrasa que se produzca el tiempo de espera de Google Drive, ya que no se resuelve el problema. Este método fue eficaz al escribir las imágenes, pero no fue suficiente cuando utilizamos nuestro modelo entrenado para identificar los paneles solares en el área de estudio, ya que el script generalmente se detendría o se quedaría en tiempo de espera después de procesar aproximadamente una quinta parte de las imágenes.

Por lo tanto, después de intentar muchas soluciones, decidimos desplegar el entorno

de Python para el entrenamiento localmente y volver a entrenar y aplicar el modelo. Sin embargo, esto no significa que nuestros scripts no puedan ejecutarse en Colab. Cuando calculamos un área más pequeña, el Colab puede proporcionar un buen soporte de computación en la nube.

3.2 PyCharm

Debido a las limitaciones de Google Drive al ejecutar scripts en Colab, optamos por desplegar el entorno necesario para entrenar el modelo de manera local. Al ejecutar los scripts localmente, utilizamos PyCharm como la plataforma para escribir y ejecutar los scripts en Python, e instalamos paquetes necesarios para el aprendizaje profundo como TensorFlow, Fiona, etc. Keras está incluido en el paquete de TensorFlow. En este TFM, utilizamos *PyCharm Professional Edition*.

PyCharm es un entorno de desarrollo integrado (IDE) profesional desarrollado por *JetBrains*, especializado en programación en Python. Ofrece funciones como autocompletado de código, integración de depuración y pruebas de scripts. En particular, la herramienta de depuración de PyCharm es muy útil al encontrar errores, ya que permite revisar paso a paso las llamadas y referencias de variables en los scripts, y proporciona detalles sobre ellas. Esta herramienta mejora significativamente la eficiencia del desarrollo [14].

Al ejecutar localmente los modelos entrenados en Colab, enfrentamos problemas de incompatibilidad de versiones, principalmente debido a diferencias en las versiones de TensorFlow que afectaban la estructura del modelo. Para resolver estos problemas de compatibilidad, utilizamos el gestor de paquetes Anaconda para instalar TensorFlow, Fiona y otros paquetes necesarios para el aprendizaje profundo, y volvimos a entrenar nuestros modelos. Esto garantizó que todos los problemas de compatibilidad fueran resueltos.

3.3 Entorno de ejecución

Al ejecutar localmente, si queremos usar Python para entrenar modelos, primero debemos configurar su entorno virtual para soportar la ejecución de nuestros scripts. Este proceso incluye la configuración de la aceleración de GPU, como CUDA y cuDNN, así como la instalación de paquetes de Python, como TensorFlow, Keras, Fiona, entre otros. Durante el proceso de instalación, también puede ser necesario modificar o ajustar manualmente las variables de entorno.

3.3.1. CUDA

CUDA (*Compute Unified Device Architecture*) es una plataforma de computación paralela y un modelo de programación desarrollado por NVIDIA, que permite a los desarrolladores escribir programas en el lenguaje CUDA (basado en C/C++) para ejecutarlos en las GPU de NVIDIA. CUDA juega un papel importante al utilizar la GPU para entrenar modelos de aprendizaje profundo y aprendizaje automático. CUDA admite la computación paralela, lo que permite ejecutar una gran cantidad de tareas de computación paralela en miles de núcleos de GPU simultáneamente. Esto mejora significativamente la velocidad de cálculo, especialmente para el entrenamiento y la inferencia de modelos de aprendizaje profundo.
En nuestro trabajo, también hemos instalado este complemento. Es importante tener en cuenta que se debe disponer de una GPU NVIDIA que sea compatible con CUDA para poder instalar este software [15].

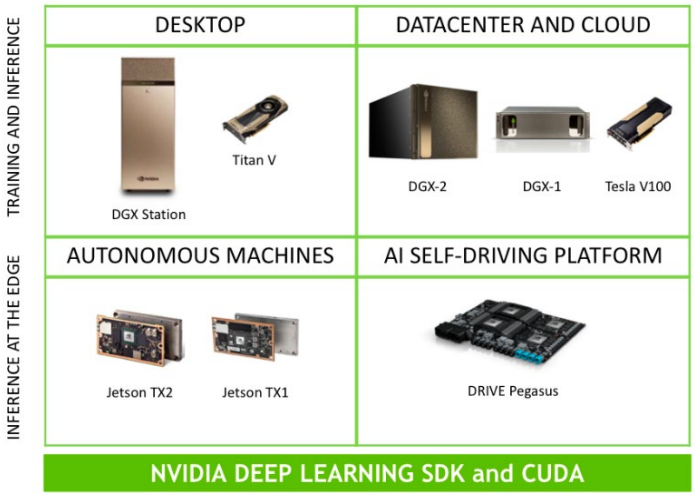


Figura 2 : Vista general de la plataforma de aprendizaje profundo de NVIDIA
Fuente: Deep learning. (n.d.). NVIDIA Developer.

3.3.2. cuDNN

cuDNN (*CUDA Deep Neural Network library*) es una biblioteca de aceleración GPU optimizada para redes neuronales profundas, construida sobre CUDA. Proporciona operaciones de alto rendimiento para convoluciones, normalización, funciones de activación, operaciones de agrupamiento (*pooling*), entre otras. cuDNN está ampliamente integrado en los principales marcos de aprendizaje profundo, como TensorFlow, PyTorch, MXNet, etc. Estos marcos llaman a la biblioteca cuDNN en el nivel

subyacente, mejorando significativamente el rendimiento del entrenamiento y la inferencia. Además, cuDNN incluye muchas estrategias de optimización que pueden seleccionar automáticamente la ruta de implementación óptima según la arquitectura de la GPU y las características del modelo, mejorando aún más el rendimiento [16].
Para nuestro trabajo, usamos el marco de aprendizaje profundo TensorFlow. Por lo tanto, la instalación de cuDNN puede mejorar significativamente el rendimiento durante el entrenamiento y la inferencia.

3.3.3. TensorFlow

TensorFlow es un marco de trabajo de aprendizaje automático y aprendizaje profundo de código abierto desarrollado por Google. Inicialmente fue creado por el equipo de *Google Brain* para satisfacer las necesidades de investigación y producción interna de *Google* y se lanzó como código abierto en 2015, convirtiéndose rápidamente en uno de los marcos de aprendizaje profundo más populares del mundo. Este marco es muy flexible y permite el despliegue de cálculos en dispositivos de escritorio, servidores, dispositivos móviles y dispositivos de borde utilizando una única API. Además, admite aceleración mediante CPU, GPU y TPU.
TensorFlow es adecuado para una variedad de tareas como el reconocimiento de imágenes, el procesamiento del lenguaje natural, el reconocimiento de voz, los sistemas de recomendación, entre otras. También ofrece API de alto nivel fáciles de usar, como Keras, lo que hace que la construcción y el entrenamiento de modelos de aprendizaje profundo sean más sencillos e intuitivos [17]. Además, cuenta con herramientas robustas de canalización de entrada de datos, como “tf.data”, que ayudan a procesar y cargar conjuntos de datos a gran escala, mejorando la eficiencia del entrenamiento.

3.3.4. Keras

Keras es una API de alto nivel para redes neuronales desarrollada por François Chollet, diseñada para hacer que la construcción y el entrenamiento de modelos de aprendizaje profundo sean más simples e intuitivos. Keras se lanzó inicialmente como una biblioteca independiente en 2015, pero a partir de TensorFlow 2.0, se integró en TensorFlow como su API de alto nivel. Keras es conocida por su diseño de API simple y coherente, fácil de aprender y usar, adecuada para el desarrollo rápido de prototipos. Su interfaz intuitiva permite a los usuarios construir, entrenar y evaluar modelos de aprendizaje profundo con facilidad. Keras fue diseñada originalmente para simplificar

y acelerar el desarrollo de modelos de aprendizaje profundo, y tiene ventajas significativas al trabajar con redes neuronales recurrentes (RNNs) y redes neuronales convolucionales (CNNs) [18].

3.3.5. Fiona

Fiona es una biblioteca de Python para leer y escribir archivos de datos geoespaciales, especialmente para interactuar con formatos de datos vectoriales como Shapefile y GeoJSON. Fiona se construye sobre la GDAL (*Geospatial Data Abstraction Library*) y proporciona una interfaz más Pythonic, simplificando las complejas operaciones de GDAL. Se utiliza principalmente para la lectura, escritura y conversión eficiente de datos vectoriales, y es compatible con otras bibliotecas geoespaciales como geopandas [19].

En nuestro trabajo, necesitamos convertir información geoespacial con frecuencia. Cada subimagen en formato ".tiff" contiene información geoespacial, y cuando el modelo UNet identifica paneles solares, es necesario crear archivos Shapefile a partir de las máscaras generadas. Para ello, debemos leer la información geoespacial del archivo ".tiff" y escribirla en el archivo Shapefile. La biblioteca Fiona permite realizar esta operación de manera eficiente y reducir el uso de memoria, lo cual es crucial cuando se trabaja con un gran número de imágenes.

3.4 QGIS

QGIS (Quantum GIS) es una aplicación de sistema de información geográfica (GIS) de código abierto, ampliamente utilizada para la creación, edición, visualización, análisis y publicación de datos geográficos. QGIS es compatible con los sistemas operativos Windows, macOS y Linux, y admite diversos formatos de datos vectoriales y ráster, incluyendo Shapefile, GeoJSON, KML, GPKG, TIFF, entre otros. QGIS cuenta con una comunidad activa que proporciona numerosos complementos, ampliando las capacidades de análisis espacial, procesamiento de teledetección y análisis de redes. Ofrece herramientas poderosas de análisis geográfico y cartografía, capaces de generar mapas de alta calidad, y también admite el acceso y uso de servicios de mapas en línea como WMS, WMTS y WFS [20].

En nuestro trabajo, QGIS se utiliza principalmente para las siguientes tareas: vectorización manual de datos ráster, recorte y combinación de ortofotografías según sea necesario, e importación de datos vectoriales finales para su análisis y modificación del modelo.

4 Datos

4.1 Zona de estudio

En este trabajo, elegimos la población de L'Eliana para realizar el estudio. Digitalizaremos manualmente los paneles solares fotovoltaicos de esta área, es decir, vectorizaremos los datos ráster. Utilizaremos los paneles solares fotovoltaicos domésticos pequeños de esta región como muestras para la posterior parte del entrenamiento de los modelos binarios de CNN y U-Net.

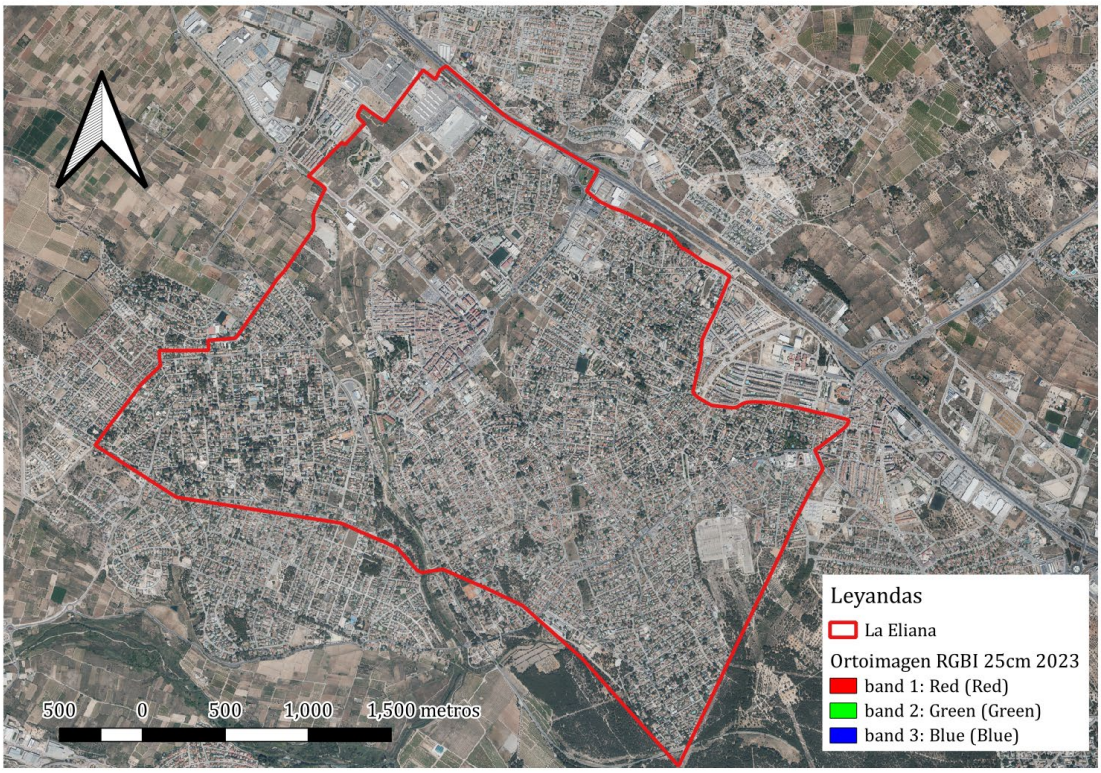


Figura 3 : Zona de estudio: término municipal de L'Eliana.

4.2 Ortofoto ICV RGB 25cm

Una ortofotografía es una fotografía con escala constante y propiedades de proyección ortogonal. Los objetos en el suelo ocupan sus posiciones horizontales reales. El Institut Cartogràfic Valencià (ICV) ofrece un producto de ortofotografía que cubre anualmente la Comunitat Valenciana con un mosaico de ortofotografías de 25 cm/píxel [21].
Esta cobertura se realiza a partir de un vuelo fotogramétrico digital RGBI con GSD de

22cm. Las imágenes son ortorectificadas a partir de orientaciones obtenidas por procesos de aerotriangulación y del modelo digital del terreno (MDT) revisado y actualizado a la fecha del vuelo para una correcta obtención de la ortofotografía. Como último paso se realiza el mosaico y ajuste final radiométrico del conjunto del trabajo, obteniendo una imagen continua. Otras características: Profundidad de color a 8 bits por banda, sistema de referencia geodésico ETRS89 y proyección UTM en el huso 30, consultable como servicio de mapas WMS, distribuido por hojas 1:5.000, siendo accesible actualmente en formato ECW (RGB 3 bandas) y TIFF (RGBI 4 bandas). En años anteriores no existe el modo de color RGBI para su descarga, ofreciéndose siempre una colección RGB y cuando exista otra IRG [21].

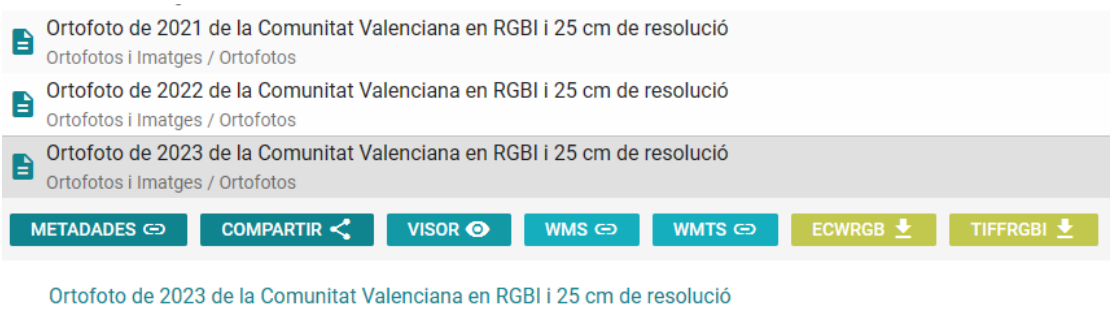


Figura 4 : Descarga de ortofoto en la web del ICV

Fuente: Ortofotos - ICV - Generalitat Valenciana. (n.d.-b). ICV.

En nuestro estudio, aunque utilizamos las ortofotografías proporcionadas por el ICV, es probable que se obtengan resultados similares utilizando las ortofotografías del PNOA. La principal diferencia es que el PNOA se actualiza cada tres años, y el ICV se actualiza anualmente. El PNOA cubre todo el territorio nacional de España, y el ICV solo proporciona ortofotografías del ámbito de la Comunidad Valenciana [7][8]. Por lo tanto, para nuestro estudio en la Comunidad Valenciana, el uso del ICV ofrece más años disponibles para elegir, lo que mejora la flexibilidad de la investigación.

4.3 Preproceso de datos

4.3.1. Combinar imágenes ráster ICV

Para facilitar la lectura y división de imágenes mediante scripts, primero necesitamos combinar las imágenes. Para las imágenes en modo de color RGB, el formato proporcionado por el ICV es ECW, que ofrece una alta tasa de compresión. Sin embargo, no podemos usar este método de compresión para volver a comprimir los resultados, lo que provoca que las imágenes combinadas en modo de color RGB sean más grandes que las originales. En cambio, las imágenes en modo de color RGBI, que incluyen la

banda infrarroja y tienen un total de cuatro bandas, ya están en tipo TIFF y usan compresión JPEG [21]. Por lo tanto, las imágenes combinadas en modo de color RGBI no aumentarán significativamente de tamaño.

Esta operación se ha realizado en QGIS, utilizando la herramienta "Combinar" de las herramientas de ráster. En esta herramienta, agregamos las cuatro imágenes descargadas del ICV que cubren el área de estudio. Realizamos esta operación tanto en las imágenes del formato ECW de tres canales RGB del año 2023, como en las imágenes del formato TIFF de cuatro canales RGBI. A continuación, se muestra el comando de la consola GDAL utilizado para esta operación.

"gdal_merge.bat -ot Byte -of GTiff -co COMPRESS=JPEG -co JPEG_QUALITY=75"

Como se puede observar, utilizamos el tipo de salida "Byte", que es el mismo tipo de datos "8 bits" utilizado en las imágenes originales. Este tipo indica que el rango de valores de píxeles de la imagen de salida es de 0 a 255. Usar este tipo de datos ayuda a reducir el tamaño de la imagen generada, minimizando el uso del espacio en el disco duro. Además, utilizamos la compresión JPEG. La variable "JPEG_QUALITY" establece el parámetro de calidad de compresión JPEG. Este valor varía de 0 a 100, representando la calidad de la imagen comprimida. Un valor más alto indica una mayor calidad de imagen y una menor tasa de compresión (archivo más grande), mientras que un valor más bajo indica una menor calidad de imagen y una mayor tasa de compresión (archivo más pequeño). Utilizamos el parámetro 75, ya que es un punto de equilibrio común que proporciona una buena calidad de imagen con una compresión moderada.

Cuando volvemos a cargar la imagen combinada en QGIS, es posible que notemos cierta transparencia en la imagen, lo que permite ver otras capas detrás de ella. Esto se debe a que, durante la combinación, el paquete GDAL escribió la banda infrarroja en la banda alfa, y ésta afecta la transparencia de la imagen. Si deseamos ver la imagen normalmente, podemos modificar manualmente los atributos de la capa de la imagen combinada. En el panel de propiedades, seleccionamos transparencia y luego en la sección de banda de transparencia, elegimos "Opaco".

4.3.2. Muestra digitalizada

En esta parte, digitalizamos manualmente casi todos los paneles solares en el área de estudio, convirtiéndolos en archivos de polígonos en formato *shapefile*. En la imagen siguiente (Figura 5), las áreas azules representan los paneles solares marcados manualmente. Hemos marcado prácticamente todos los paneles solares o áreas similares a paneles solares. Sin embargo, debido a la observación visual sobre

ortofotografías con una resolución espacial de 25 cm, puede ser difícil distinguir claramente entre algunos materiales de techo negro y los paneles solares, por lo que algunos marcajes pueden contener errores.

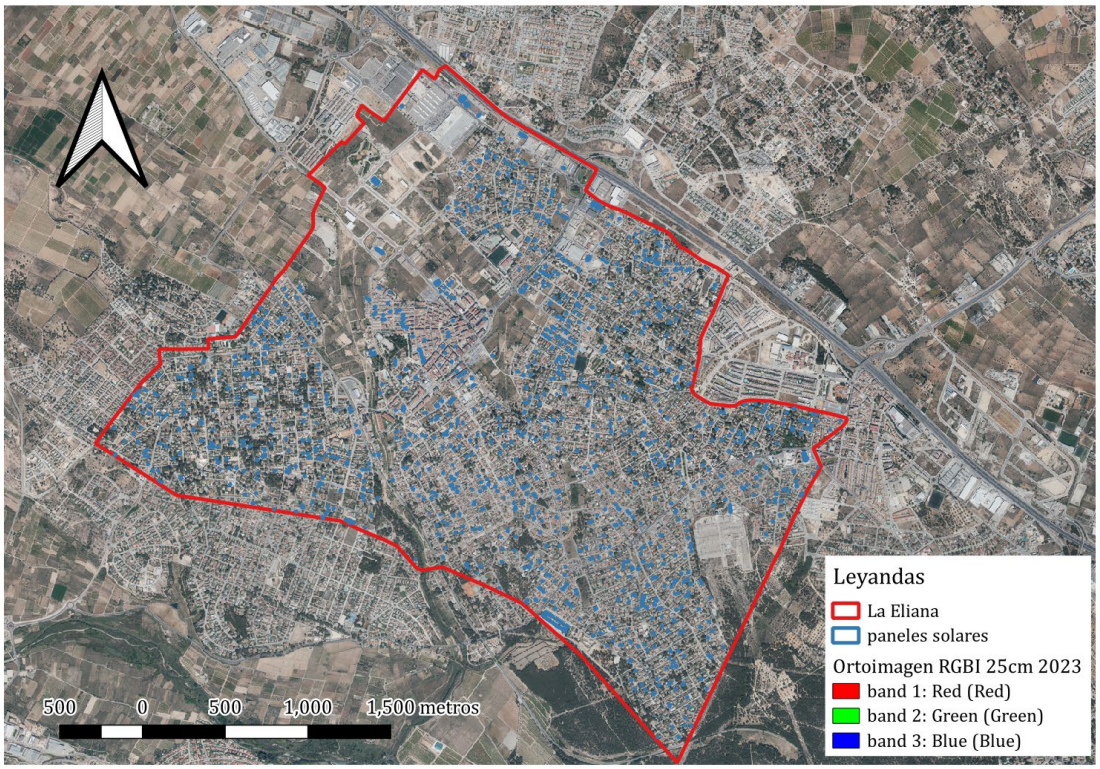


Figura 5 : Mapa de los paneles solares

Debido a problemas de iluminación, algunos paneles solares reflejan la luz y aparecen en blanco. En los pasos posteriores, eliminaremos estas partes que no están claras.

4.3.3. Segmentar imagen

Utilizamos un script en Python (Anexo 11.1.1). para dividir las imágenes combinadas que contienen el área de estudio en fragmentos de 32x32 píxeles para el entrenamiento. El script leerá las imágenes ráster en formato TIFF que combinamos anteriormente y que contienen el área de estudio. Dado que solo hemos digitalizado manualmente los paneles solares dentro del término municipal de L'Eliana, usamos un archivo poligonal en formato SHP que contiene los límites de esta área. De esta manera, el script solo generará subimágenes dentro de los límites del área de estudio.

Para las imágenes en modo de color RGBI, al actualizar los metadatos de salida, agregamos manualmente *"count": src.count*. Esto garantiza que se escriban todas las bandas, incluida la banda infrarroja.

Aunque utilizamos compresión JPEG al comprimir las imágenes combinadas

anteriormente, al dividirlas en subimágenes, utilizamos compresión LZW. Esto se debe a que nuestras imágenes se dividen en fragmentos de 32x32 píxeles, que son muy pequeños. En este caso, la compresión JPEG puede no mejorar significativamente la tasa de compresión y la compresión con pérdida podría afectar innecesariamente la calidad de la imagen. Por lo tanto, la compresión LZW es más adecuada para esta situación, ya que puede comprimir manteniendo la calidad de la imagen. Dado el pequeño tamaño de las imágenes, el tiempo de compresión y el tamaño del archivo generado con LZW no deberían ser un problema.

4.3.4. Determinar muestras

En este paso, para crear un conjunto de muestras de entrenamiento, necesitamos seleccionar de las imágenes segmentadas cuáles de ellas contienen paneles, a las que asignamos el valor “yes” y a las que no representan paneles en su interior les asignamos el valor “no”. Luego, asignaremos aleatoriamente las muestras de entrenamiento "yes" y "no" a los directorios de entrenamiento (*train*) y prueba (*test*). El total de muestras se dividirá en 80% para entrenamiento y 20% para evaluación. Para completar este paso, utilizamos varios scripts en Python.

Primero, un script permite filtrar todas las imágenes en una carpeta usando un archivo poligonal (Anexo 11.1.2). En este script, utilizamos las subimágenes de 32x32 píxeles obtenidas en el paso anterior (4.3.3). Recorremos todas las subimágenes en la carpeta y si sus límites se superponen con el archivo poligonal en formato SHP, las movemos al directorio "yes"; de lo contrario, las movemos al directorio "no".

A continuación, usamos dos scripts para procesar el contenido del directorio "yes". Las imágenes en este directorio incluyen, en mayor o menor medida, una parte de paneles solares. Sin embargo, algunas imágenes pueden contener solo una pequeña parte de los paneles solares, lo que podría afectar negativamente el entrenamiento del modelo CNN. Por lo tanto, primero calculamos el área del archivo poligonal en formato SHP en cada subimagen. Si el área es mayor al 15% de la subimagen, la movemos a un nuevo conjunto de muestras "yes", que luego se utilizará para entrenar el modelo CNN.

El primer script (Anexo 11.1.3) calcula el área de intersección entre cada subimagen y el archivo poligonal. Este script crea temporalmente un polígono del mismo tamaño que la subimagen, calcula la intersección con el archivo en formato SHP, y guarda el área en un archivo CSV. Este archivo CSV contiene el nombre del archivo, el ID del polígono y el porcentaje. Luego, podemos filtrar manualmente los porcentajes, dejando solo los archivos que tienen un área mayor al 15%. A continuación, utilizamos otro script (Anexo 11.1.4) para leer el archivo CSV y, según los nombres de archivo

incluidos, mover las imágenes a un nuevo conjunto de muestras "yes" que se utilizará para entrenar el modelo CNN.
Finalmente, contamos un total de 1086 subimágenes que se movieron al conjunto de muestras "yes". Para garantizar que los resultados del entrenamiento del modelo CNN binario no se sesguen debido a un desequilibrio en las muestras de entrenamiento, seleccionamos aleatoriamente 1086 muestras del directorio "no" para el conjunto final de muestras "no". Este proceso se llevó a cabo mediante un sencillo script en Python (Anexo 11.1.5), que puede mover aleatoriamente la cantidad especificada de imágenes de un directorio a otro.

4.3.5. Crear mascara para U-Net

Para entrenar el modelo U-Net, necesitamos crear una máscara para cada subimagen. La máscara debe contener dos valores, 0 y 1, donde todos los valores positivos indican áreas donde existen paneles solares. Para ello, utilizamos un script en Python para completar este paso (anexo 11.1.6). El script leerá las subimágenes en formato TIFF desde un directorio especificado. Y luego, basándose en su información geográfica de subimagen, creará un bloque de subimagen para encontrar las áreas que se superponen con el archivo poligonal en formato SHP. Luego, el script creará una máscara completamente en ceros y rasterizará las áreas superpuestas, asignando el valor 1 a esas áreas. Finalmente, el script convertirá la máscara a formato PNG, donde todos los valores 1 se convertirán en 255. Se puede ver algunos ejemplos de las máscaras generadas abajo (Figura 6), mostrando un total de 5 pares de subimágenes que contienen paneles solares. En cada par de ejemplos, la imagen de la izquierda es la imagen original en color verdadero RGB, y la de la derecha es la máscara generada mediante el script. Los píxeles positivos de la máscara indican el alcance de los paneles solares.

Figura 6 : Ejemplos de las máscaras generadas con sus imágenes originales

4.3.6. Convertir a PNG

Cuando entrenamos nuestro modelo de CNN, se ha utilizado el método *“keras.preprocessing.image_dataset_from_directory”* para cargar los datos de

imágenes. Este método utiliza esencialmente la API “tf.data.Dataset” de TensorFlow para cargar datos de imágenes desde un directorio. La función “image_dataset_from_directory” admite de forma predeterminada formatos de imagen comunes como PNG y JPEG, pero no admite directamente el formato TIFF. Por lo tanto, necesitamos convertir las imágenes en formato TIFF a formato PNG. Para ello, utilizamos un sencillo script (anexo 11.1.8).

Para las imágenes en modo de color RGBA, ya que al combinar las imágenes del área de estudio escribimos la banda infrarroja en el canal alfa, solo necesitamos realizar la conversión directamente sin operaciones adicionales. A continuación, se muestra un ejemplo de los cuatro canales de una subimagen RGBA (RGBI) (Figura 7), donde el canal alfa contiene la información de la banda infrarroja.

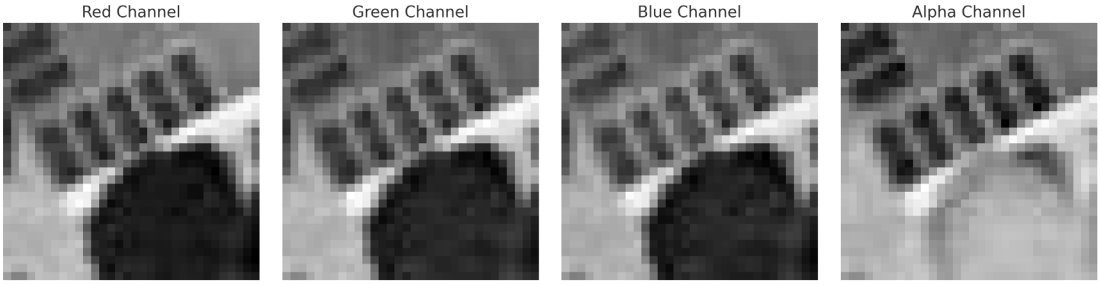


Figura 7 : Ejemplo de los 4 canales de una subimagen de RGBA(RGBI) en formato PNG

5 Metodología

En este trabajo, utilizamos el enfoque de aprendizaje profundo para identificar paneles solares fotovoltaicos, entrenando un modelo CNN (redes neuronales convolucionales) binario y un modelo U-Net (Redes Convolucionales para la Segmentación de Imágenes Biomédicas).

Desde sus inicios en los años 1950, la inteligencia artificial ha evolucionado significativamente. En los años 1990, el aprendizaje automático comenzó a prosperar, aprovechando grandes volúmenes de datos y el aumento en la capacidad de procesamiento. En la década de 2010, los avances en el aprendizaje profundo, una subdisciplina del aprendizaje automático, impulsaron un auge en la inteligencia artificial. El aprendizaje profundo, basado en redes neuronales artificiales, ha permitido logros extraordinarios en áreas como el reconocimiento de imágenes y el procesamiento del lenguaje natural, transformando diversas industrias y nuestra vida cotidiana [22].

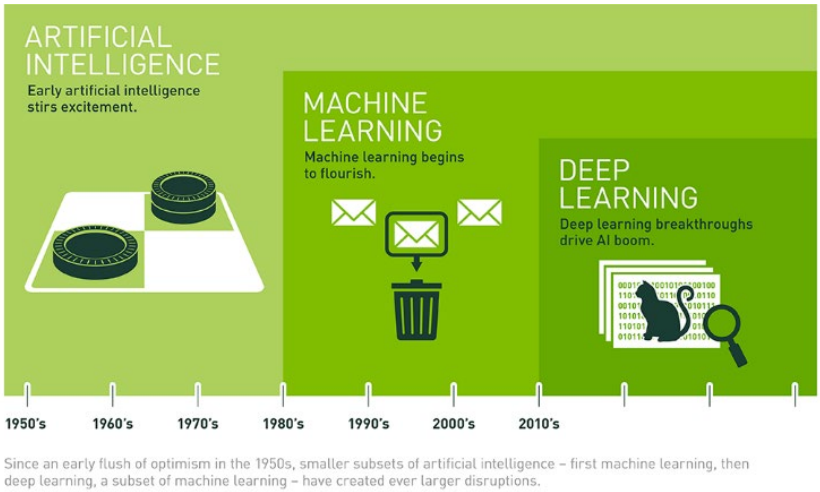


Figura 8 : Evolución de la inteligencia artificial

Fuente: Deep learning. (n.d.-b). NVIDIA Developer

La mayor parte del trabajo de entrenamiento en esta tarea se realizó en Colab. Posteriormente, debido a las limitaciones de Google Drive, trasladamos la ejecución a nivel local. Sin embargo, todo el código puede ejecutarse en Colab con ligeras modificaciones (importación y montaje de Google Drive). El uso de la computación en la nube puede mejorar significativamente la conveniencia del trabajo y facilitar la revisión repetida de los resultados (3.1.2).

5.1 Diagrama de flujo

5.1.1. Diagrama de flujo de trabajo

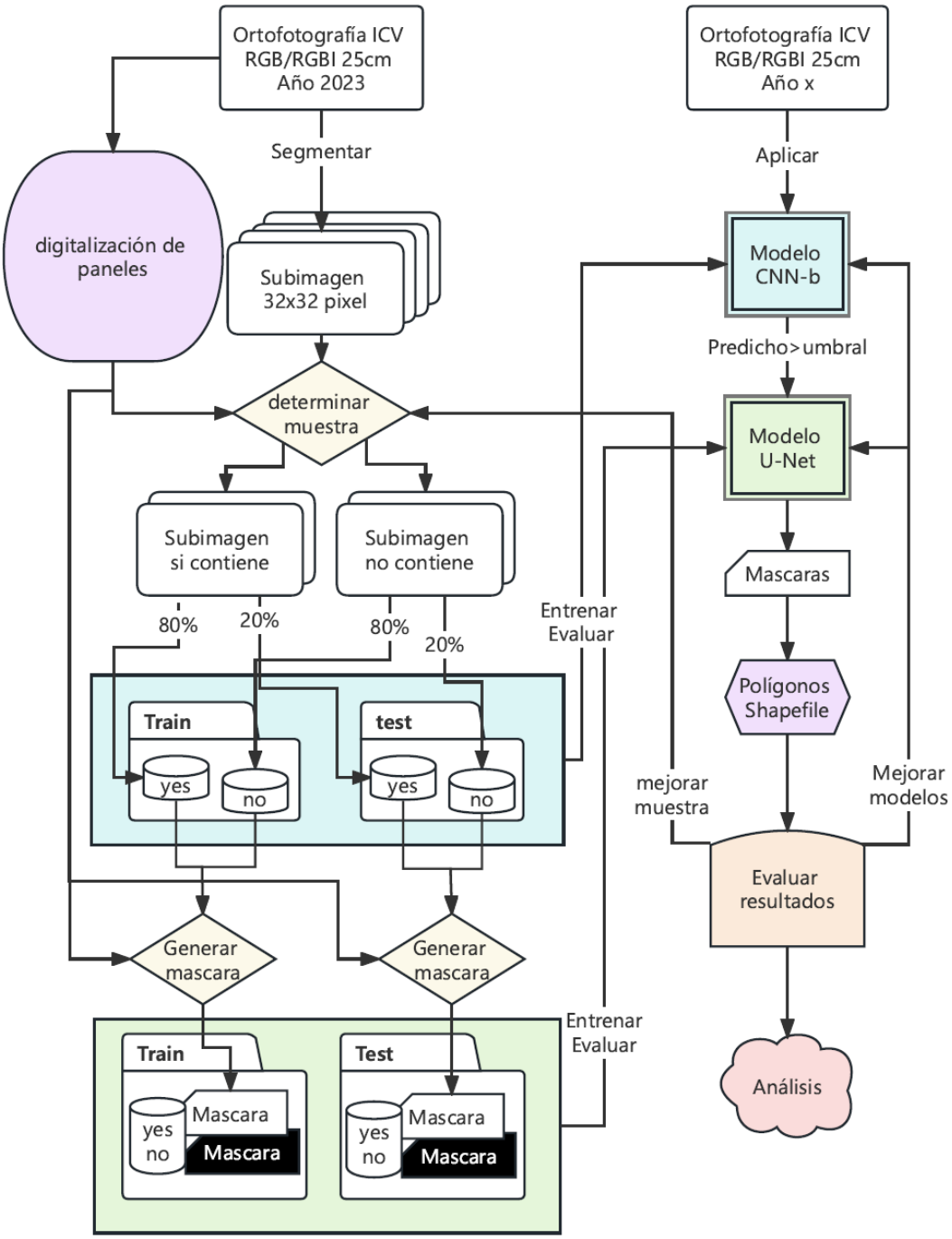


Figura 9 : Diagrama de trabajo

Este diagrama de flujo (Figura 9) describe brevemente una serie de operaciones desde la selección de muestras hasta la obtención de los resultados finales. A través del diagrama de flujo, se puede comprender que primero utilizamos las ortofotografías de 25 cm de resolución espacial de 2023 para digitalizar manualmente el área de estudio y dividirla en varias subimágenes. Luego, determinamos los conjuntos de entrenamiento y test para los modelos CNN-b y U-Net (también necesita generar máscaras para modelo U-Net). Después de entrenar los modelos, utilizamos estos dos modelos para aplicar el modelo CNN-b que hace un primer filtrado y luego el modelo U-Net que generar una máscara con la zona de las placas solares a las imágenes ortográficas de 25 cm de resolución espacial de cualquier año. Finalmente, generamos archivos *shapefile* a partir de las máscaras creadas por el modelo para el análisis de datos final. Según los resultados de estos datos, también optimizamos continuamente los dos modelos. El proceso de optimización y determinación del modelo final se puede ver en el siguiente diagrama (Figura 10).

5.1.2. Diagrama de flujo de modelo

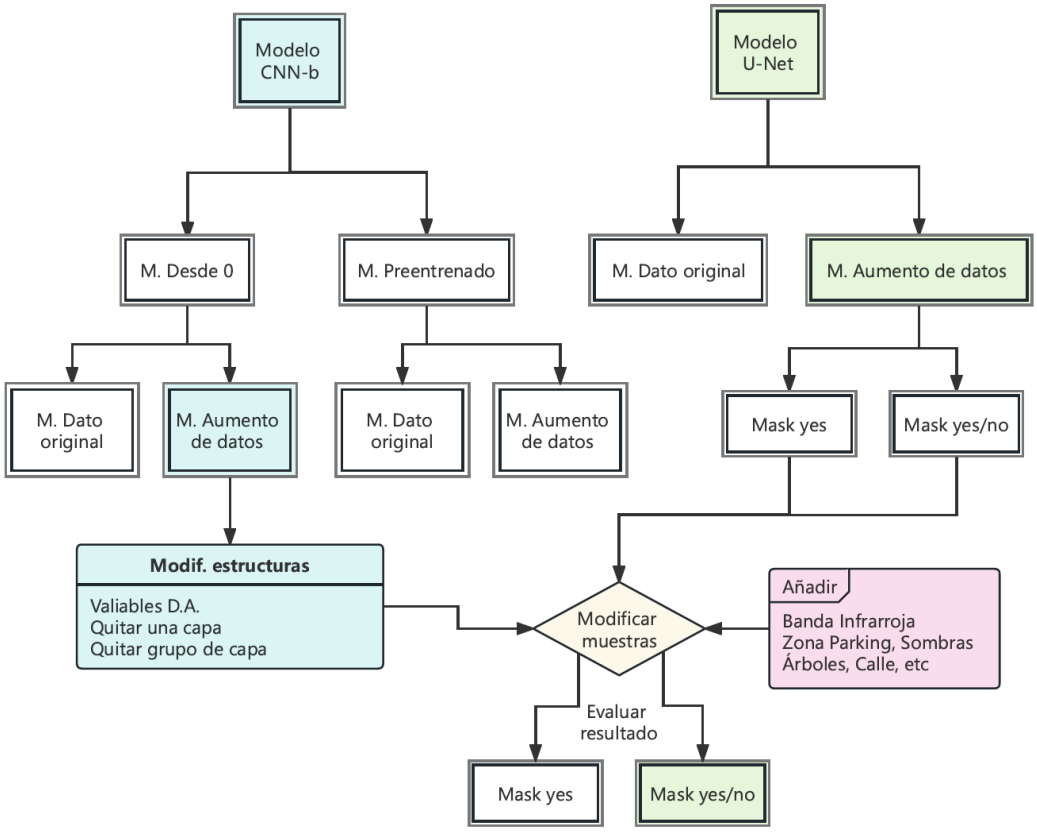


Figura 10 : Diagrama de flujo de modelo

A partir del diagrama de flujo del modelo, podemos observar que primero determinamos la estructura básica de los modelos CNN-b y U-Net a través de una serie de intentos. Luego, evaluamos los resultados y realizamos modificaciones continuas y reentrenamientos del modelo. Al entrenar el modelo CNN-b, probamos dos enfoques: entrenar desde cero y mejorar un modelo preentrenado. Para ambos enfoques, probamos si usar o no el aumento de datos. Finalmente, descubrimos que el modelo entrenado desde cero con aumento de datos producía los mejores resultados. Luego, continuamos optimizando el modelo CNN-b modificando los parámetros de aumento de datos y eliminando algunas capas.
Para el modelo U-Net, probamos el uso del aumento de datos y encontramos que la precisión era elevada. Luego, consideramos que la inclusión de muestras negativas (las muestras que no contiene las zonas de las placas solares) durante el entrenamiento debería tener un efecto positivo en el modelo, aunque no pudimos confirmarlo solo con el conjunto de validación. Por lo tanto, después de aplicar el modelo, realizamos ajustes y optimizaciones continúas basándonos en los resultados finales. En este proceso, añadimos la banda infrarroja y manualmente agregamos algunas muestras negativas que eran propensas a errores de identificación. Finalmente, descubrimos que el modelo U-Net entrenado con la inclusión de muestras negativas producía mejores resultados. Mientras ajustábamos las muestras negativas, también reentrenamos y optimizamos la capacidad de identificación del modelo CNN-b.

5.2 Entrenamiento del modelo CNN binario

Una red neuronal convolucional (*Convolutional Neural Network*, CNN) es un modelo de aprendizaje profundo ampliamente utilizado en el procesamiento de imágenes y vídeos. Su diseño se inspira en el sistema visual biológico, especialmente en el mecanismo de respuesta a características visuales locales en la corteza visual de los gatos. En nuestro trabajo, entrenamos y evaluamos un modelo CNN especializado en clasificación binaria (CNN-b). Finalmente, encontramos un modelo CNN con el mejor rendimiento teórico. El proceso de intentar diferentes modelos CNN-b y encontrar el más adecuado se puede ilustrar en el diagrama de flujo mencionado anteriormente (Figura 10 : Diagrama de flujo de modelo).

5.2.1. Arquitectura de modelo CNN

La estructura de un modelo CNN está compuesta por multitud de capas. Normalmente, una CNN incluye las siguientes: capa de convolución, función de activación, capa de

pooling, capa totalmente conectada y capa de salida. Por supuesto, para optimizar el rendimiento del modelo, podemos usar otras capas que no son específicas de los CNN, como la capa Dropout y la capa de normalización por lotes [23].

En nuestro trabajo, creamos un modelo CNN que entrena desde cero que incluye las capas mencionadas anteriormente. A continuación, se detallará la función de cada capa.

Capa de convolución

La capa de convolución (*Convolutional Layer*) es el componente central de una red neuronal convolucional (CNN), utilizada para extraer características de los datos de entrada, especialmente los datos de imagen. Esta capa aplica un núcleo de convolución (filtro o *kernel*) que se desplaza sobre los datos de entrada, calculando la suma ponderada de las regiones locales para extraer características en diferentes niveles. Está compuesta principalmente por el núcleo de convolución, la longitud del paso, el relleno y la función de activación [23].

Ej.1.: x1 = Conv2D(32, (3, 3), padding="same", activation="relu")(inputs)

En nuestra capa de convolución (Ej.1), utilizamos 32 filtros, es decir, el número de canales en el mapa de características de salida. El tamaño de cada filtro es de 3x3, es decir, una altura y un ancho de 3. Utilizamos el modo de relleno "*same*", lo que significa que el mapa de características de salida tiene las mismas dimensiones espaciales (altura y ancho) que el mapa de características de entrada, rellenando con ceros alrededor de la entrada para mantener las dimensiones. Después, utilizamos la función de activación ReLU (*Rectified Linear Unit*).

Función de activación

La función de activación es un componente crucial en las redes neuronales, ya que introduce no linealidad, permitiendo que la red neuronal aprenda y represente patrones complejos. La función de activación convierte la suma ponderada de las neuronas (la combinación lineal) en una salida no lineal, lo que permite que el modelo ajuste datos no lineales [23]. Las funciones de activación comunes incluyen *Sigmoid*, *Tanh* y *ReLU*.

Ej.2.: predictions = Dense(classes, activation="sigmoid")(xfc)

En nuestro trabajo, utilizamos las funciones de activación *ReLU* y *sigmoid*. En el ejemplo anterior (Ej.1), ReLU ya se ha añadido a la capa de convolución. Además, la función de activación *sigmoid* se ha añadido a la capa totalmente conectada (Ej.2), ya que esta función es más adecuada para representar probabilidades.

Capa de pooling

La capa de *pooling* (*Pooling Layer*) es un componente importante en las redes neuronales *convolucionales* (CNN), y se utiliza principalmente para realizar un

muestreo a la baja de los mapas de características generados por la capa de convolución. La función principal de la capa de pooling es reducir el tamaño de los mapas de características, disminuir la carga computacional y el consumo de memoria, al mismo tiempo que conserva la información de las características importantes [23].

Ej.3. : x1 = MaxPooling2D(pool_size=(2, 2))(x1)

En nuestro trabajo, utilizamos una capa de *pooling* con una ventana de tamaño 2x2 (Ej.3) para realizar la operación de *max pooling*.

Capa totalmente conectada

La capa totalmente conectada (*Fully Connected Layer*), también conocida como capa densa (Dense Layer), es un tipo de capa básica en las redes neuronales. Cada neurona de la capa totalmente conectada está conectada a todas las neuronas de la capa anterior, permitiendo la integración de características y la representación de alto nivel [23].

Ej.4. : xfc = Flatten()(x1)
xfc = Dense(512, activation="relu")(xfc)

En nuestro trabajo, utilizamos una capa de aplanamiento (*Flatten Layer*) para convertir los mapas de características multidimensionales en un vector unidimensional, facilitando su entrada a la capa totalmente conectada. Luego, se realiza la integración de características de alto nivel y la transformación no lineal. En el ejemplo (Ej.4), se utilizan 512 neuronas, lo que significa que la dimensión del vector de características de salida (xfc) es 512.

Capa dropout

La capa Dropout es una técnica de regularización utilizada para prevenir el sobreajuste en las redes neuronales. La idea principal es que, durante cada iteración del entrenamiento, se descarta aleatoriamente un porcentaje de neuronas, lo que impide que la red dependa demasiado de ciertas características locales, mejorando así la capacidad de generalización del modelo [23][24].

Ej.5. : x1 = Dropout(0.25)(x1)

En nuestro trabajo, debido a que los paneles solares son muy simples en forma y estructura, los modelos entrenados tienden a sobreajustarse. Por lo tanto, la incorporación de la capa Dropout es muy necesaria. En el ejemplo (Ej.5), la tasa de abandono de la capa Dropout se establece en 0.25. Lo que significa que, durante el entrenamiento, esta capa eliminará aleatoriamente las neuronas de entrada con una probabilidad del 25% (establecerá sus valores de salida a cero).

Capa de normalización por lote

La normalización por lotes (*Batch Normalization*) es una técnica de regularización utilizada para acelerar el entrenamiento de redes neuronales profundas y mejorar su

rendimiento y estabilidad. Esta capa normaliza las entradas en cada *mini*-lote, reduciendo el desplazamiento interno de covariables (internal covariate shift), lo que estabiliza la distribución de los datos de entrada para cada capa [23][25].

Ej.6.: if batchNorm: x1 = BatchNormalization()(x1)

En nuestro trabajo, por defecto (batchNorm = True), utilizamos una capa de normalización por lotes después de cada capa de convolución y capa totalmente conectada.

5.2.2. Modelos CNN para clasificación binaria

Al crear el modelo CNN para la clasificación binaria (CNN-b), realizamos muchos intentos. Por ejemplo, definimos un modelo CNN que entrena desde cero, utilizamos modelos preentrenados, empleamos conjuntos de datos aumentados y realizamos ajustes sobre modelos. A continuación, presentaremos los modelos CNN utilizados para la clasificación binaria, describiendo su estructura y componentes básicos. En la parte posterior de esta memoria, se seleccionará el modelo CNN que mejor se adapte a nuestro trabajo según los resultados de la evaluación.

Modelo que entrena desde cero

La arquitectura de la red del modelo CNN-b que entrena desde cero se puede consultar en el siguiente diagrama esquemático:

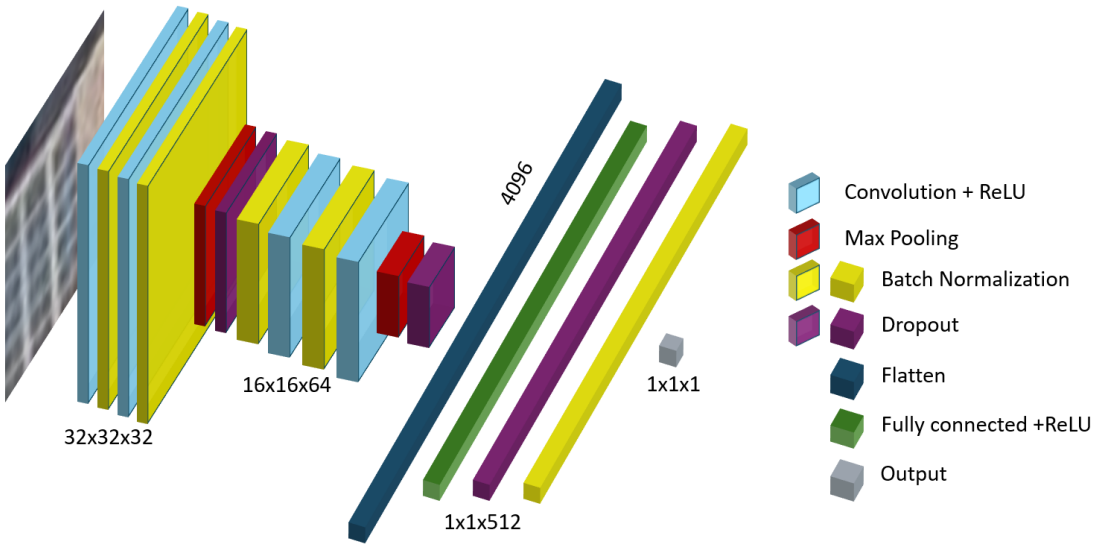


Figura 11 : Esquema de arquitectura de la red del modelo CNN-b

Primero utilizamos dos bloques de capas de convolución, cada uno con dos capas de convolución. El primer grupo de capas de convolución tiene dos capas con 32 filtros cada una, con núcleos de convolución de 3x3 y la función de activación ReLU. El

segundo grupo de capas de convolución tiene dos capas con 64 filtros cada una, el resto de los parámetros igual que en el primer grupo. Cada capa de convolución va seguida de una capa de normalización por lotes, usando un total de 4 capas de normalización por lotes. Después de cada grupo de capas de convolución, se añade una capa de *max pooling* de 2x2, usando un total de 2 capas de *max pooling*. Luego, cada capa de *pooling* va seguida de una capa Dropout con una tasa de abandono del 25%, usando un total de 2 capas Dropout. A continuación, se usa una capa completamente conectada para convertir las características multidimensionales en un vector unidimensional. Luego, se añade una capa totalmente conectada con 512 neuronas y la función de activación ReLU, seguida de una capa de normalización por lotes y una capa Dropout con una tasa de abandono del 50%. Finalmente, la capa de salida tiene tantas neuronas como clases, y utiliza la función de activación "*sigmoid*" para la clasificación. La función de activación "*sigmoid*" puede mapear la entrada a un rango entre 0 y 1, por lo que a menudo se utiliza para la estimación de probabilidades. Para nuestra tarea de clasificación binaria, la salida de esta función puede representar de manera intuitiva la probabilidad de que una muestra pertenezca a la clase positiva (por ejemplo, que contenga paneles solares).

Modelo empleando modelo preentrenado (fine tuning)

También intentamos usar el modelo preentrenado VGG16 con tecnología *fine tuning* para entrenar nuestro modelo. Primero, importamos el modelo preentrenado VGG16 desde tensorflow.keras.applications y especificamos los pesos preentrenados de *ImageNet*, el tamaño de entrada de 32x32x3, y excluimos la parte superior de clasificación. Luego, congelamos las capas de los primeros cuatro bloques de convolución del modelo VGG16, de modo que estas capas no actualicen sus pesos durante el entrenamiento, preservando así sus características preentrenadas. Es decir, el script recorre todas las capas del modelo y deja de congelar al encontrar la capa llamada 'block5_conv1'; las capas anteriores se configuran como no entrenables. Después, construimos un nuevo modelo secuencial. Este modelo incluye primero la base del VGG16 preentrenado, seguido de una capa de aplanamiento (*Flatten Layer*) para convertir las características multidimensionales en un vector unidimensional. A continuación, agregamos dos capas totalmente conectadas: la primera con 1000 neuronas y la función de activación ReLU, y la segunda con 200 neuronas y también la función de activación ReLU. Finalmente, agregamos una capa de salida, cuya cantidad de neuronas es igual al número de clases (*classes*) y utiliza la función de activación *sigmoid* para realizar la tarea de clasificación multietiqueta. Esta estructura combina la capacidad de extracción de características del modelo preentrenado con la funcionalidad de clasificación personalizada de las nuevas capas añadidas.

```
# Cargar el modelo VGG16 pre-entrenado con dimensiones de entrada específicas
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(32, 32, 3))
# Mostrar la arquitectura del modelo
base_model.summary()

# Congelar las capas de los primeros cuatro bloques de convolución,
#     el quinto bloque se reentrenará
for layer in base_model.layers:
    if layer.name == 'block5_conv1':
        break
    layer.trainable = False
    print('Layer ' + layer.name + ' frozen...')

# Construir un modelo secuencial que incluya:
# Añadir el modelo pre-entrenado con bloques congelados menos el último bloque
# Añadir nuestro propio clasificador, incluyendo una capa oculta con 1000 neuronas y
#     activación relu, un dropout de 0,3, otra capa oculta con 200 neuronas y
#     activación relu, y una capa de salida con activación softmax
model = Sequential()
model.add(base_model)
model.add(Flatten())
model.add(Dense(1000, activation='relu', name='fc1'))
model.add(Dropout(0.3))
model.add(Dense(200, activation='relu', name='fc2'))
model.add(Dense(n_classes, activation='sigmoid', name='predictions'))
```

Figura 12 : Empleando con modelo preentrenado VGG16

Además de VGG16, también probamos otros modelos preentrenados como ResNet50. Solo es necesario importar el modelo desde "tensorflow.keras.applications" sin necesidad de cambiar otros parámetros básicos.

Aumento de datos

Para mejorar la capacidad de generalización del modelo, utilizamos la aumentación de datos en tiempo real (*real-time data augmentation*) para ajustar el conjunto de entrenamiento de entrada [23]. En el ejemplo (Figura 13), utilizamos "ImageDataGenerator" para realizar la aumentación de datos en tiempo real. Al llamar a esta función, se introdujeron varios parámetros:

- El parámetro "rotation_range=40" indica que las imágenes se rotarán aleatoriamente hasta 40 grados durante el entrenamiento. Esto ayuda a que el modelo sea menos sensible a las rotaciones, mejorando así su robustez.
- El parámetro "horizontal_flip=True" da a las imágenes un 50% de probabilidad de ser volteadas horizontalmente, lo cual es útil para tareas que no son sensibles a la simetría horizontal.
- El parámetro "vertical_flip=True" significa que las imágenes tienen un 50% de probabilidad de ser volteadas verticalmente, lo que es útil en ciertos escenarios donde los objetos en las imágenes son simétricos en vertical.
- El parámetro "rescale=1./255" escala los valores de los píxeles de la imagen al rango de 0 a 1, dividiendo cada valor de píxel por 255. Esto normaliza los datos de la imagen,

ayudando a acelerar el proceso de entrenamiento del modelo y a mantener la estabilidad numérica.

- Finalmente, El parámetro “validation_split=0.2” divide los datos en conjuntos de entrenamiento y validación, usando el 20% de los datos para validar el rendimiento del modelo. Esto permite monitorear el sobreajuste del modelo durante el entrenamiento.

```
datagen = ImageDataGenerator(
    rotation_range=40,
    horizontal_flip=True,
    vertical_flip=True,
    rescale = 1./255,
    validation_split=0.2,
    )
```

Figura 13 : Utilizando aumentos de datos en tiempo real

Para maximizar la capacidad de generalización del modelo, también intentamos introducir más parámetros al añadir la aumentación de datos en tiempo real. Como se muestra en la imagen (Figura 14), añadimos 5 nuevos parámetros adicionales:

- Los parámetros "width_shift_range=0.2" y "height_shift_range=0.2"indican que las imágenes se desplazarán aleatoriamente en la dirección horizontal y dirección vertical hasta un 20% de su ancho. Estos ajustes ayudan a mejorar la capacidad del modelo para manejar la invariancia a desplazamientos.
- El parámetro "shear_range=0.2" aplica una transformación de cizallamiento en un rango de hasta el 20%, lo que aumenta la diversidad de las deformaciones en las imágenes, ayudando al modelo a aprender mejor las características de diferentes deformaciones.
- El parámetro "zoom_range=0.1" indica que las imágenes se escalarán aleatoriamente dentro de un rango del 10%, lo que ayuda al modelo a manejar objetos a diferentes escalas.
- El parámetro "fill_mode='nearest'" asegura que, después de las transformaciones de la imagen, las áreas vacías se llenen con el valor del píxel más cercano, evitando así que las áreas vacías afecten al entrenamiento.

```
datagen = ImageDataGenerator(
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.1,
    horizontal_flip=True,
    vertical_flip=True,
    fill_mode='nearest',
    rescale=1. / 255,
    validation_split=0.2
)
```

Figura 14 : Utilizando aumentos de datos con más parámetros

5.2.3. Entrenar modelo CNN binario

Para entrenar nuestro modelo CNN para clasificación binaria, creamos un script en Python para entrenamiento y evaluación (Anexo 11.2.1). Este script puede ejecutarse tanto en Colab como en PC localmente. A lo largo del uso de este script, modificamos varias veces la parte de definición del modelo para encontrar los mejores parámetros. También intentamos utilizar modelos preentrenados, como se mencionó en la sección anterior.

En este script, primero importamos las bibliotecas necesarias. Entre ellas, “tensorflow” y “keras” se utilizan para construir y entrenar modelos de aprendizaje profundo, “ImageDataGenerator” para la aumentación de datos, “sklearn” para calcular el informe de clasificación y las métricas de evaluación, y “matplotlib” para la visualización de imágenes. Si se ejecuta en Colab, además del script en el apéndice, es necesario importar y montar Google Drive para acceder a los datos almacenados en Google Drive.

Cuando se ejecuta localmente, usamos CUDA y cuDNN de NVIDIA para acelerar el cálculo mediante GPU, por lo que es necesario tener instalados CUDA y cuDNN previamente. Si se ejecuta en Colab, no es necesario configurar esto, solo importar las bibliotecas necesarias y ejecutar el script. Esta es una de las principales ventajas de Colab.

A continuación, configuramos las rutas de los archivos. El script necesita leer dos directorios: uno para el conjunto de entrenamiento (*train*) y otro para el conjunto de prueba (test). En cada uno de estos directorios, debe haber dos carpetas, yes y no, que definen las muestras de clase positiva y negativa respectivamente.

Al leer los archivos, nuestro generador de datos (aumento de datos) de imágenes

define el 20% del conjunto de entrenamiento como el conjunto de validación. El conjunto de validación juega un papel crucial en la evaluación del modelo. A través del rendimiento del modelo en el conjunto de validación, podemos entender su capacidad de generalización y detectar rápidamente si el modelo está experimentando un sobreajuste.

Luego, creamos dos funciones: una para definir la arquitectura del modelo CNN (función deep_CNN) y otra para realizar la normalización de las imágenes (función normalize). Después, cargamos los generadores de datos para realizar la aumentación de datos en tiempo real y preparamos los conjuntos de entrenamiento y validación.

Usamos la variable "*class_weight*" para definir los pesos de las clases y abordar el problema del desbalance de datos. Si los datos "yes" y "no" están equilibrados, no es necesario configurar este valor. Esta variable es un diccionario en el que 0 y 1 representan la clase positiva y negativa, respectivamente. Los valores se calculan según la proporción de cada clase, lo que implica que cada clase tiene un peso diferente durante el entrenamiento. Un valor más alto significa un mayor peso. El cálculo se realiza sumando el total de muestras y dividiendo por la cantidad de muestras de cada clase. Por ejemplo, si hay 1000 muestras de la clase yes y 2000 muestras de la clase no, entonces el peso de la clase positiva es (2000+1000)/1000=3 y el peso de la clase negativa es 1.5.

Finalmente, compilamos y entrenamos el modelo utilizando la biblioteca "keras", y guardamos el modelo entrenado. Para todos los modelos, casi siempre entrenamos durante 1000 épocas. Esto puede aumentar la precisión hasta cierto punto, pero un entrenamiento excesivo también puede incrementar el riesgo de sobreajuste. Al entrenar en Colab, ocasionalmente el entrenamiento puede detenerse después de un cierto número de ciclos debido a algunos parámetros, lo que hace que deje de ejecutarse. Por lo tanto, es posible que reduzcamos el número de ciclos cuando sea necesario. Luego, evaluamos el modelo. Durante la evaluación, generamos un informe usando la biblioteca "sklearn.metrics", que proporciona el F1 Score, la precisión y el valor de pérdida para cada clase. También podemos imprimir manualmente estos valores u otros parámetros útiles para el análisis, como el coeficiente de correlación de Matthews (MCC). Después, generamos tres gráficos basados en la precisión y la pérdida. Los primeros dos gráficos muestran la curva de pérdida a lo largo de los ciclos de entrenamiento: el primero con el eje Y ajustado automáticamente y el segundo con el eje Y limitado entre 0 y 1 o entre 0 y 1.5. Esto se debe a que las fluctuaciones en el conjunto de validación pueden ser bastante grandes, lo que no facilita el análisis. Estas fluctuaciones se deben a que pequeños ajustes en el modelo pueden tener un gran impacto en los resultados, lo que puede llevar a un aumento o disminución abrupta en la precisión, y también puede indicar sobreajuste. Usar tanto un rango automático

como un rango fijo puede ayudarnos a analizar y determinar visualmente si ha ocurrido sobreajuste. El tercer gráfico muestra la curva de precisión a lo largo de los ciclos de entrenamiento.

5.2.4. Evaluación y comparación de los modelos

Al evaluar el modelo, generamos un informe que incluye exactitud (*accuracy*), precisión (*precision*), tasa de pérdida (*loss*), tasa de *recall* y valor F1.
Estos indicadores utilizados para evaluar el modelo se calculan a partir de la matriz de confusión. La matriz de confusión se compone de 4 valores clave:
TP (Verdaderos Positivos): Muestras correctamente identificadas como positivas.
FP (Falsos Positivos): Muestras incorrectamente identificadas como positivas.
TN (Verdaderos Negativos): Muestras correctamente identificadas como negativas.
FN (Falsos Negativos): Muestras incorrectamente identificadas como negativas.
A continuación, se describen sus significados y funciones [23] [26] [27]:

- La precisión o exactitud (*Accuracy*) indica la proporción de muestras correctamente predichas por el modelo en relación con el total de muestras. Es un indicador intuitivo que refleja la capacidad general del modelo para realizar predicciones correctas y es adecuado para conjuntos de datos con una distribución equilibrada de clases.

$$\text{Exactitud } (\textit{Accuracy}) = \frac{\text{TP} + \text{TN}}{\text{TP}+\text{FP}+\text{TN}+\text{FN}}$$

- La precisión (Precision) indica la proporción de muestras que el modelo ha predicho correctamente como positivas en relación con el total de muestras que predijo como positivas. La precisión es un indicador crucial cuando se quiere minimizar los falsos positivos. Es especialmente útil en situaciones donde el costo de una falsa alarma es alto. Un modelo con alta precisión asegura que, de las muestras predichas como positivas, una gran proporción son realmente positivas.

$$\text{Precisión } (\textit{Precision}) = \frac{\text{TP}}{\text{TP}+\text{FP}}$$

- La tasa de pérdida (*loss*) mide el grado de diferencia entre las predicciones del modelo y los resultados reales. Se utiliza para evaluar el objetivo de optimización durante el entrenamiento del modelo. Generalmente, al minimizar la función de pérdida, se mejora el rendimiento del modelo. La tasa de pérdida puede reflejar el progreso del entrenamiento y la convergencia del modelo.
- La tasa de sensibilidad o recuperación (*recall*) indica la proporción de muestras positivas reales que el modelo predijo correctamente como positivas. Se centra en la capacidad del modelo para identificar muestras positivas, y es adecuada para

escenarios donde la detección de muestras positivas es crítica, como en la detección de enfermedades o fraudes.

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

- El valor F1(F1 Score) es la media armónica de la precisión (*Precision*) y la tasa de *recall* (*Recall*), y combina sus ventajas y desventajas. Es especialmente útil para evaluar modelos en situaciones de desequilibrio de clases, proporcionando un indicador equilibrado entre la precisión y la tasa de *recall*. Es adecuado para escenarios que requieren un equilibrio entre falsos positivos y falsos negativos.

$$\text{Valor F1(F1 Score)} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Además de algunos parámetros obtenidos del conjunto de prueba (*test*), también utilizamos el conjunto de validación. El conjunto de validación desempeña un papel crucial en la evaluación del modelo. A través del rendimiento del modelo en el conjunto de validación, podemos entender su capacidad de generalización y detectar rápidamente si el modelo está experimentando un sobreajuste.

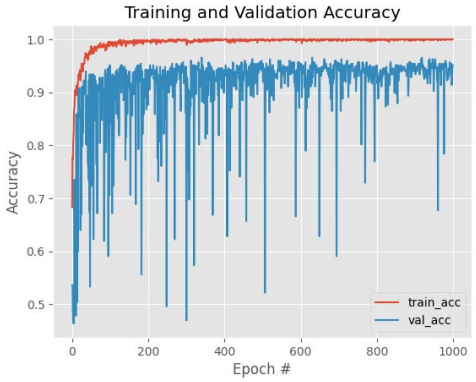
También generamos gráficos que muestran cómo varían la tasa de pérdida y la tasa de *recall* del conjunto de entrenamiento y el conjunto de validación con el número de ciclos. Estos parámetros nos ayudan a analizar el rendimiento del modelo y a detectar visualmente la presencia de sobreajuste.

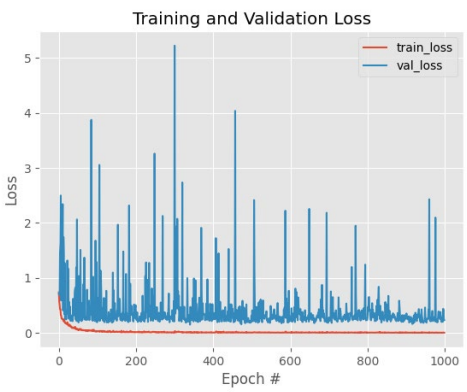
A continuación, enumeraremos algunos de los intentos más importantes realizados en este trabajo y los resultados obtenidos para su evaluación. A través de la evaluación de estos resultados, iremos filtrando y optimizando nuestro modelo de manera gradual. Este proceso es como se indica en el diagrama de flujo anterior (Figura 10).

Modelo CNN-b entrenado desde cero

Inicialmente, utilizamos una función definida por nosotros para entrenar un modelo CNN desde cero. La definición y estructura de este modelo ya se han explicado en la sección anterior (5.2.2).

	precision	recall	f1-score	support
No	0.95	0.94	0.94	217
Yes	0.94	0.95	0.94	217
accuracy			0.94	434
macro avg	0.94	0.94	0.94	434
weighted avg	0.94	0.94	0.94	434





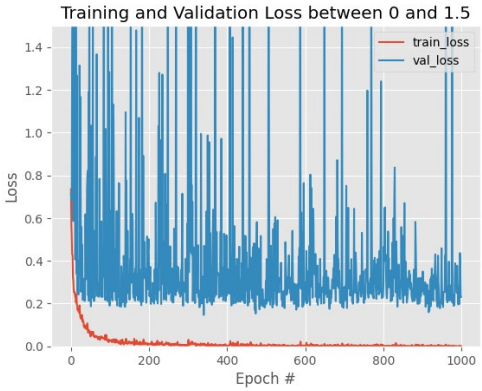


Figura 15 : Informe y figuras de exactitud (Accuracy) y pérdida del Modelo CNN-b entrenado desde cero

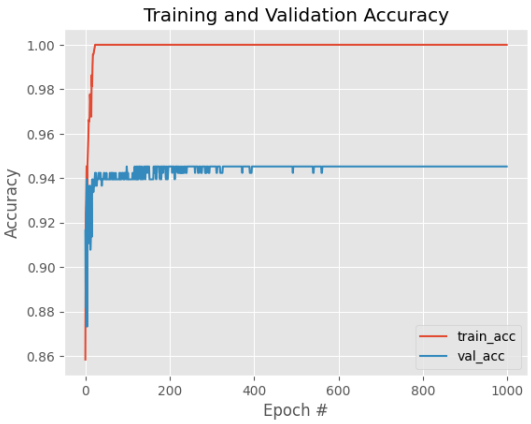
Del informe de nuestro primer modelo, podemos ver que la exactitud, la tasa de *recall* y el valor F1 están alrededor de 0.95. Este resultado preliminar es muy prometedor. Además, en los gráficos de pérdida a lo largo de las iteraciones, no se observa un sobreajuste evidente. Sin embargo, aún podemos explorar otras formas de optimizar el modelo o reducir aún más el riesgo de sobreajuste.

Observamos que en los gráficos de pérdida y exactitud del conjunto de validación hay fluctuaciones significativas. En futuras optimizaciones, nos enfocaremos en suavizar estas fluctuaciones.

Modelo CNN-b con modelo preentrenado VGG16

Además del modelo entrenado desde cero, también intentamos modificar y ajustar modelos preentrenados para generar nuestro modelo.

	precision	recall	f1-score	support
No	0.95	0.95	0.95	217
Yes	0.95	0.95	0.95	217
accuracy			0.95	434
macro avg	0.95	0.95	0.95	434
weighted avg	0.95	0.95	0.95	434



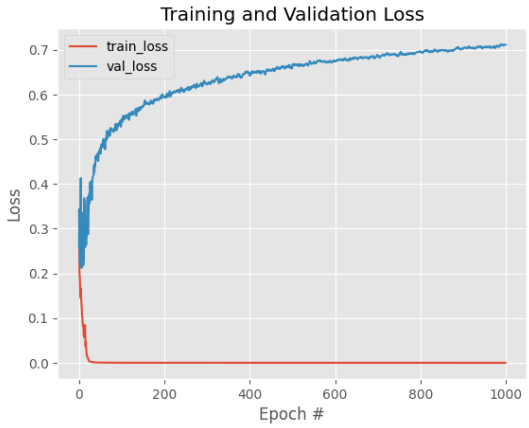


Figura 16 : Informe y figuras de exactitud y perdida de modelo CNN-b con modelo preentrenado VGG16

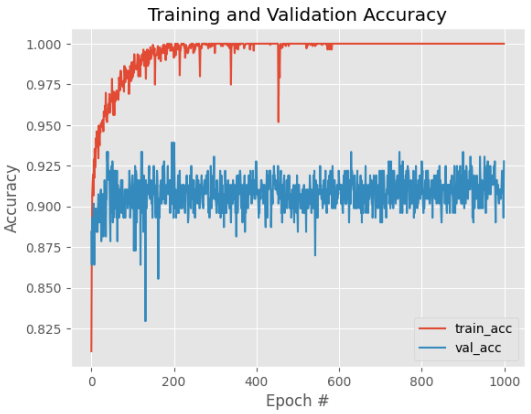
Según el informe de retroalimentación y los gráficos generados, observamos que la pérdida de entrenamiento tiende a cero, lo que indica un ajuste casi perfecto, mientras que la pérdida de validación aumenta, lo que muestra un caso claro y típico de sobreajuste.

El sobreajuste (*Overfitting*) se refiere a un fenómeno en el que un modelo de aprendizaje automático funciona extremadamente bien en los datos de entrenamiento, pero tiene un rendimiento pobre en los datos de prueba o de validación. El sobreajuste significa que el modelo ha capturado el ruido y los patrones específicos del conjunto de datos de entrenamiento en lugar de aprender las reglas generales de los datos [23][26].

Modelo CNN-b con modelo preentrenado VGG16 con aumento de datos

Sobre la base de los modelos preentrenados, añadimos aumentación de datos adicional. Primero utilizamos los parámetros básicos de aumentación de datos mencionados en la sección anterior (5.2.2). Esperamos que esto pueda reducir el sobreajuste hasta cierto punto.

	precision	recall	f1-score	support
No	0.98	0.94	0.96	217
Yes	0.94	0.98	0.96	217
accuracy			0.96	434
macro avg	0.96	0.96	0.96	434
weighted avg	0.96	0.96	0.96	434



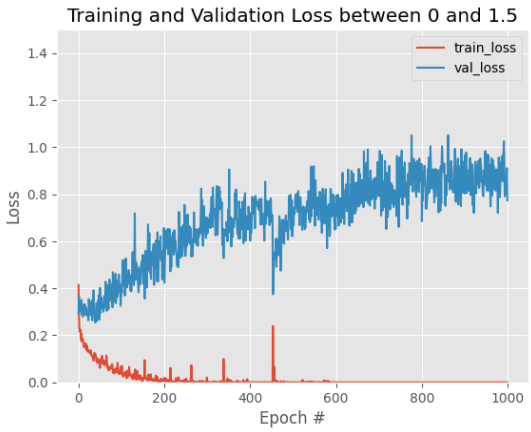


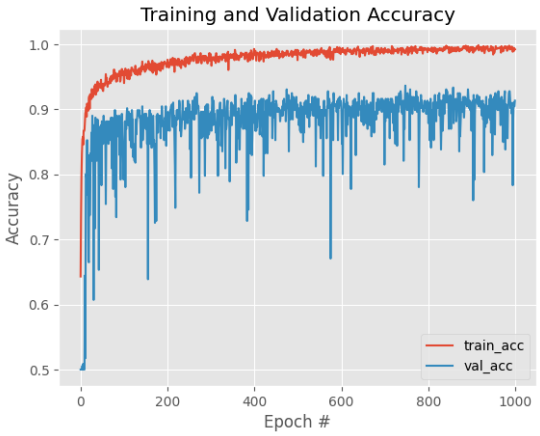
Figura 17 : Informe y figuras de exactitud y perdida de modelo CNN-b con modelo preentrenado VGG16 con aumento de datos

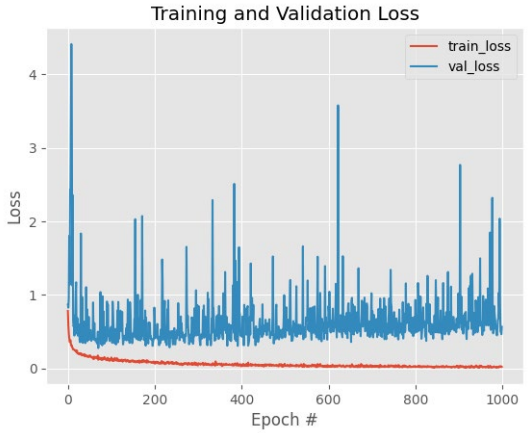
Según los resultados generados, a pesar de haber utilizado la aumentación de datos para reducir el sobreajuste, el conjunto de validación todavía muestra un claro signo de sobreajuste. Sin embargo, hubo una cierta mejora en comparación con no utilizar la aumentación de datos. Creemos que esto se debe a que el modelo preentrenado es demasiado complejo y afecta negativamente nuestro reconocimiento.

Modelo CNN-b entrenado desde cero con aumento de datos

Debido a la evidente aparición de sobreajuste al usar modelos preentrenados, decidimos continuar utilizando el modelo CNN entrenado desde cero, incorporando también la función de aumentación de datos.

	precision	recall	f1-score	support
No	0.98	0.95	0.96	217
Yes	0.95	0.98	0.96	217
accuracy			0.96	434
macro avg	0.96	0.96	0.96	434
weighted avg	0.96	0.96	0.96	434





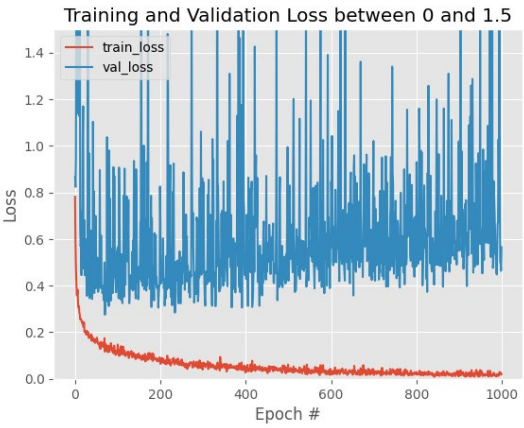


Figura 18 : Informe y figuras de exactitud y perdida de modelo CNN-b entrenado desde cero con aumento de datos

A partir de las imágenes, se puede observar que, incluso con la utilización de la aumentación de datos, la pérdida en el conjunto de validación sigue mostrando grandes fluctuaciones y una tendencia a aumentar. Esto indica que el rendimiento del modelo en el conjunto de validación es inestable y puede estar ocurriendo un sobreajuste. Por lo tanto, es necesario realizar ajustes adicionales.

Modelo CNN-b entrenado desde cero con aumento de datos y ajuste de variables

Después de comparar el modelo CNN binario entrenado desde cero con el modelo que utiliza aumentación de datos adicional, descubrimos que el uso de la aumentación de datos produce mejores resultados. Creemos que los paneles solares tienen una forma

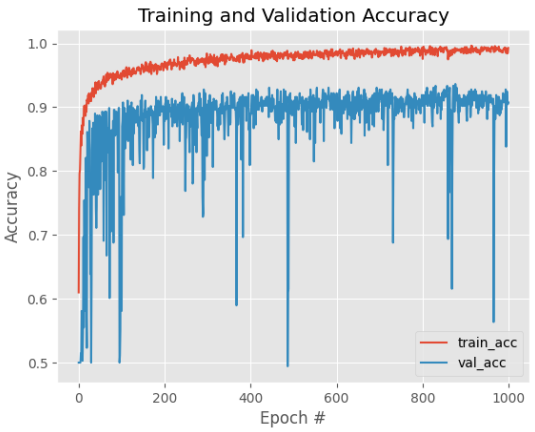
demasiado simple, por lo que un entrenamiento excesivamente complejo podría tener un efecto negativo en los resultados. Por lo tanto, para realizar ajustes adicionales, decidimos modificar los parámetros de la arquitectura del modelo en el segundo grupo de capas, cambiando el número de filtros a 32 o 128, o incluso eliminando una capa o el grupo de capas completo.

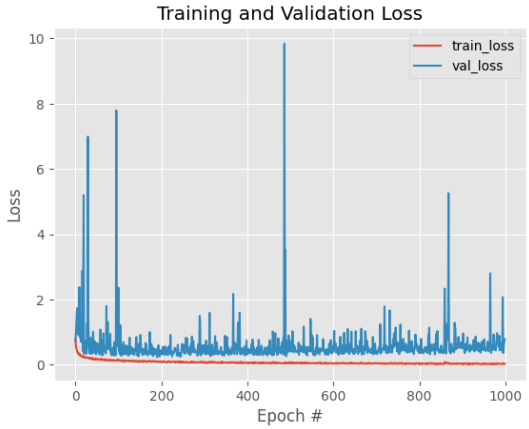
```
# Segundo grupo de capas: CONV => RELU => CONV => RELU => POOL
x2 = Conv2D(64, (3, 3), padding="same", activation="relu")(x1)
if batchNorm:
    x2 = BatchNormalization()(x2)
x2 = Conv2D(64, (3, 3), padding="same", activation="relu")(x2)
if batchNorm:
    x2 = BatchNormalization()(x2)
x2 = MaxPooling2D(pool_size=(2, 2))(x2)
x2 = Dropout(0.25)(x2)
```

Figura 19 : Código original de segundo grupo de capas

Primero, reducimos los parámetros de las capas de convolución en el segundo grupo de capas de 64 a 32.

	precision	recall	f1-score	support
No	0.97	0.99	0.98	217
Yes	0.99	0.97	0.98	217
accuracy			0.98	434
macro avg	0.98	0.98	0.98	434
weighted avg	0.98	0.98	0.98	434





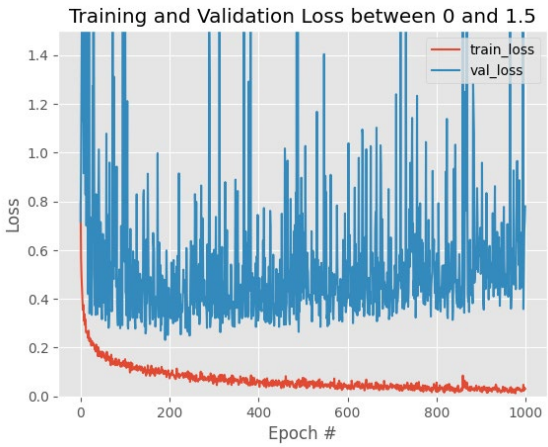


Figura 20 : Informe y figuras de exactitud y perdida de Modelo CNN-b entrenado desde cero con aumento de datos y ajuste variable de segundo grupo de capa al 32

Después, aumentamos los parámetros de las capas de convolución en el segundo

grupo de capas de 64 a 128.

	precision	recall	f1-score	support
No	0.98	0.93	0.96	217
Yes	0.93	0.98	0.96	217
accuracy			0.96	434
macro avg	0.96	0.96	0.96	434
weighted avg	0.96	0.96	0.96	434

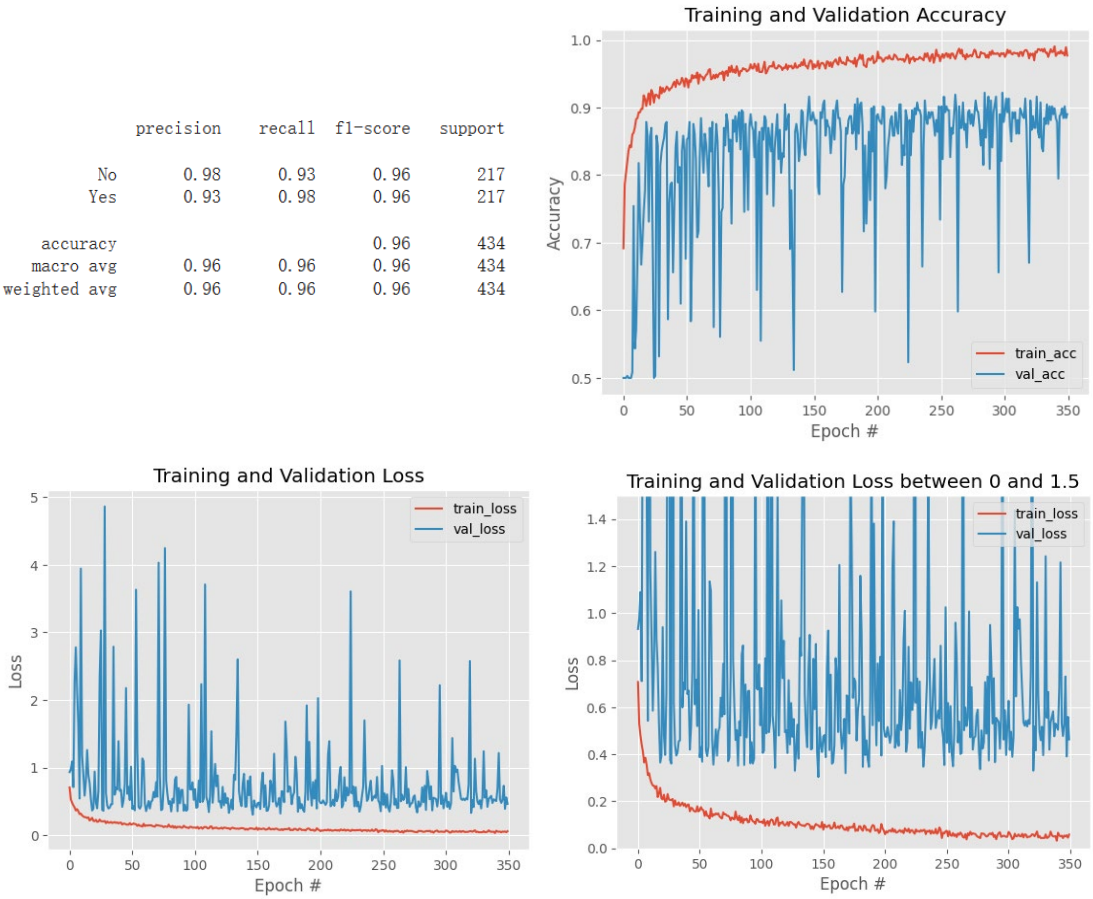


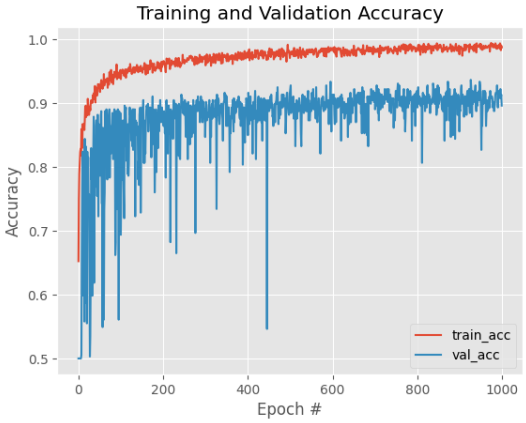
Figura 21 : Informe y figuras de exactitud y perdida de Modelo CNN-b entrenado desde cero con aumento de datos y ajuste variable de segundo grupo de capa al 128

Al modificar los parámetros a 128, puede que el uso de memoria o la gestión de recursos durante el entrenamiento del modelo se vuelvan más inestables. Esto a menudo provocaba bloqueos en la ejecución del código. Por lo tanto, ajustamos el número de épocas a 350. Sin embargo, según los resultados mostrados, la modificación de los parámetros tuvo un impacto muy limitado en el rendimiento del modelo.

Modelo CNN-b entrenado desde cero con aumento de datos y eliminación de una capa

Después de intentar modificar los parámetros, también intentamos eliminar una de las capas de convolución en el segundo grupo de capas. Por supuesto, también eliminamos la capa de normalización por lotes que seguía a esta capa de convolución.

	precision	recall	f1-score	support
No	0.95	0.99	0.97	217
Yes	0.99	0.95	0.97	217
accuracy			0.97	434
macro avg	0.97	0.97	0.97	434
weighted avg	0.97	0.97	0.97	434



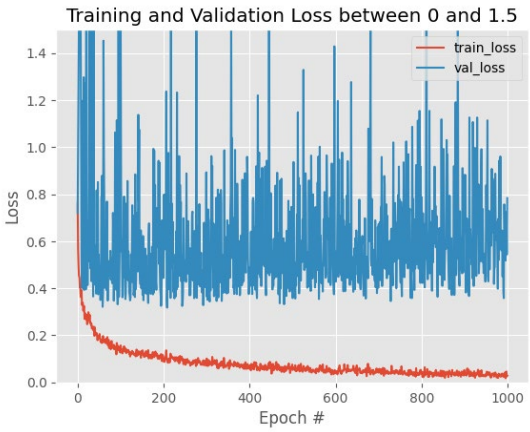


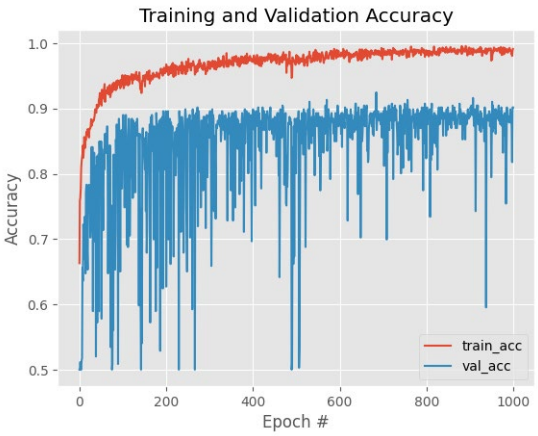
Figura 22 : Informe y figuras de exactitud y perdida de modelo CNN-b entrenado desde cero con aumento de datos y eliminación de una capa

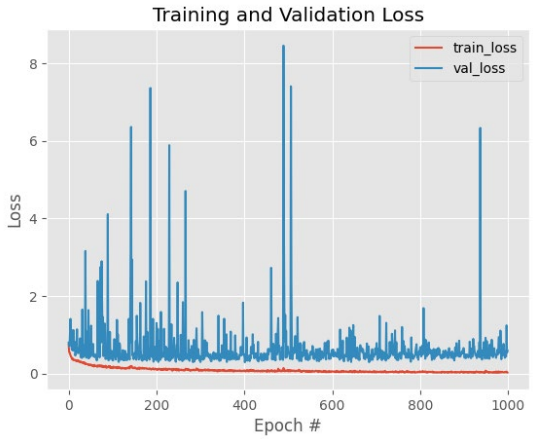
Descubrimos que, al eliminar una capa, aunque las fluctuaciones en la pérdida de validación siguen siendo muy pronunciadas, la tendencia a aumentar se ha controlado. En general, esto ayuda a controlar el sobreajuste en cierta medida y reduce ligeramente la pérdida de validación.

Modelo CNN-b entrenado desde cero con aumento de datos y eliminación de un bloque convolucional

Al mismo tiempo, también intentamos eliminar todo el segundo grupo de capas (un bloque convolucional), incluyendo todas las capas de convolución, las capas de normalización por lotes y las capas de *dropout*.

	precision	recall	f1-score	support
No	0.95	0.98	0.97	217
Yes	0.98	0.95	0.97	217
accuracy			0.97	434
macro avg	0.97	0.97	0.97	434
weighted avg	0.97	0.97	0.97	434





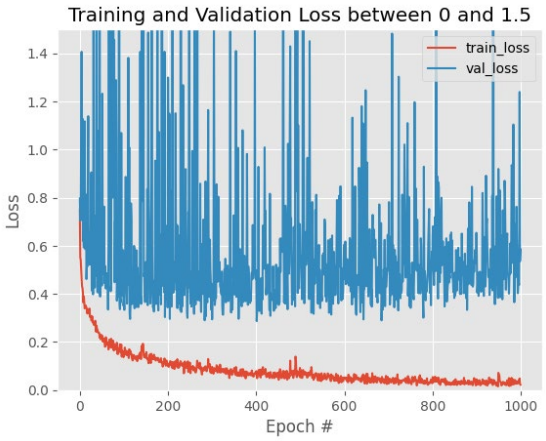


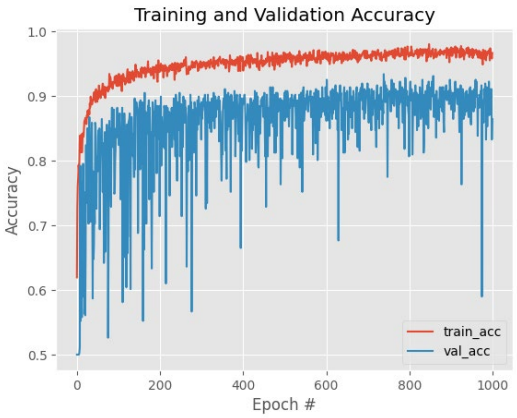
Figura 23 : Informe y figuras de exactitud y perdida de modelo CNN-b entrenado desde cero con aumento de datos y eliminación de un bloque convolucional

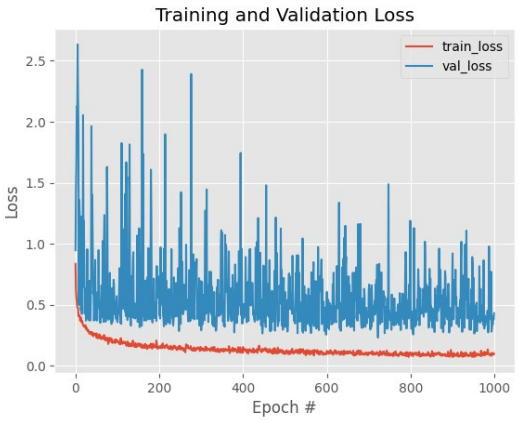
Descubrimos que, al eliminar todo el grupo de capas, la pérdida de validación se redujo aún más y no parece mostrar una tendencia al sobreajuste.

Modelo CNN-b entrenado desde cero con aumento de datos ajustado y eliminación de una capa

Para optimizar mejor nuestro modelo, utilizamos parámetros adicionales para configurar la aumentación de datos en tiempo real. Los parámetros específicos se mencionaron en la sección anterior (5.2.2). Primero, aplicamos esta aumentación de datos ajustada al modelo con una capa eliminada.

	precision	recall	f1-score	support
No	0.94	0.99	0.96	217
Yes	0.99	0.94	0.96	217
accuracy			0.96	434
macro avg	0.96	0.96	0.96	434
weighted avg	0.96	0.96	0.96	434





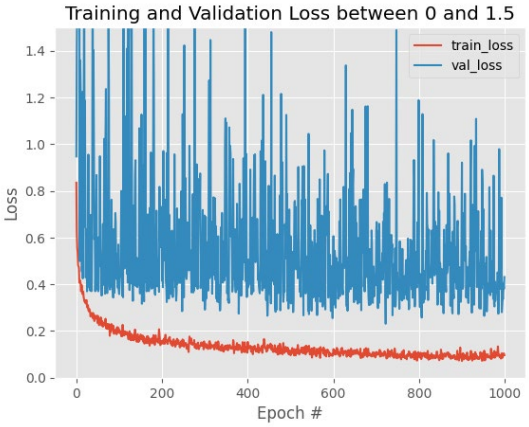
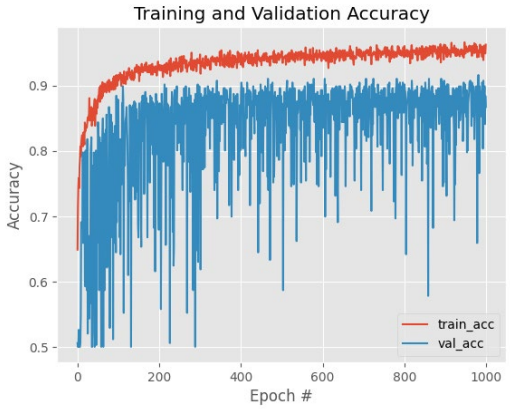


Figura 24 : Informe y figuras de exactitud y perdida de modelo CNN-b entrenado desde cero con aumento de datos ajustado y eliminación de una capa

Modelo CNN-b entrenado desde cero con aumento de datos ajustado y eliminación de un bloque convolucional

Después, también aplicamos la aumentación de datos ajustada al modelo con el grupo de capas completo eliminado.

	precision	recall	f1-score	support
No	0.96	0.96	0.96	217
Yes	0.96	0.96	0.96	217
accuracy			0.96	434
macro avg	0.96	0.96	0.96	434
weighted avg	0.96	0.96	0.96	434



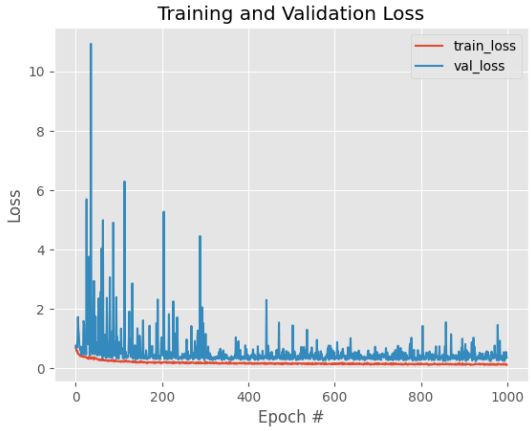


Figura 25 : Informe y figuras de exactitud y perdida de modelo CNN-b entrenado desde cero con aumento de datos ajustado y eliminación de un bloque convolucional

Después de comparar una serie de intentos anteriores, no es difícil ver que el modelo que eliminó todo el grupo de capas y aplicó la aumentación de datos en tiempo real con los nuevos parámetros muestra una mejor pérdida de validación. Además, el modelo obtuvo puntuaciones altas en el informe de clasificación: precisión, recall y F1 score, todos en 0.96, y una precisión general de 0.96, lo que indica que el modelo se desempeña bien en la tarea de clasificación. Aunque las fluctuaciones en la pérdida de validación siguen siendo bastante pronunciadas, lo que nos preocupa, este es el mejor modelo que hemos obtenido hasta ahora a través del análisis. Usaremos este modelo para realizar la tarea de clasificación binaria en las muestras, seleccionando las subimágenes que posiblemente contengan paneles solares.

5.3 Entrenamiento del modelo U-Net

El modelo U-Net es una red neuronal convolucional (CNN) utilizada para tareas de segmentación de imágenes, y es muy popular en campos como la segmentación de imágenes médicas. El modelo U-Net extrae gradualmente características multiescala de la imagen a través de un codificador y recupera gradualmente la información espacial de la imagen a través de un decodificador, combinando conexiones de salto para mejorar la precisión de la segmentación. De esta manera, el modelo U-Net puede aprender eficazmente diversas características de la imagen y generar resultados de segmentación precisos.[28]

El U-Net es una red completamente convolucional (CNN), lo que significa que carece de capas densas o completamente conectadas, lo que facilita su entrenamiento y reduce los recursos computacionales necesarios. Está compuesto por dos conjuntos de capas convolucionales que operan en un esquema de codificador-decodificador. El primer conjunto de capas realiza un muestreo descendente de la imagen, creando un conjunto de imágenes con diferentes resoluciones. El segundo conjunto de capas intenta recrear la imagen a su resolución original mediante la creación de un conjunto de imágenes de muestreo ascendente. En una serie de redes de muestreo descendente, se pueden detectar las características más importantes sin destruir la forma o textura de los objetos. El propósito de la reconstrucción es compensar la pérdida generada durante la etapa de codificación [29].

En nuestro trabajo, definimos y creamos un modelo U-Net que puede aprender las áreas que contienen paneles solares a partir de las máscaras generadas por los datos digitalizados manualmente. Finalmente, este modelo puede generar máscaras que contienen los paneles solares, y en operaciones posteriores, estas máscaras se convierten en archivos vectoriales para su análisis. Durante el entrenamiento del modelo, presentamos dos enfoques: uno usando solo áreas que contienen paneles solares para entrenar el modelo y otro usando áreas que contienen y otras que no contienen paneles solares. La diferencia radica en que las máscaras de las áreas que no contienen paneles solares son completamente negras.

5.3.1. Arquitectura del modelo U-Net

La arquitectura del modelo U-Net que hemos definido se puede consultar en el siguiente diagrama esquemático:

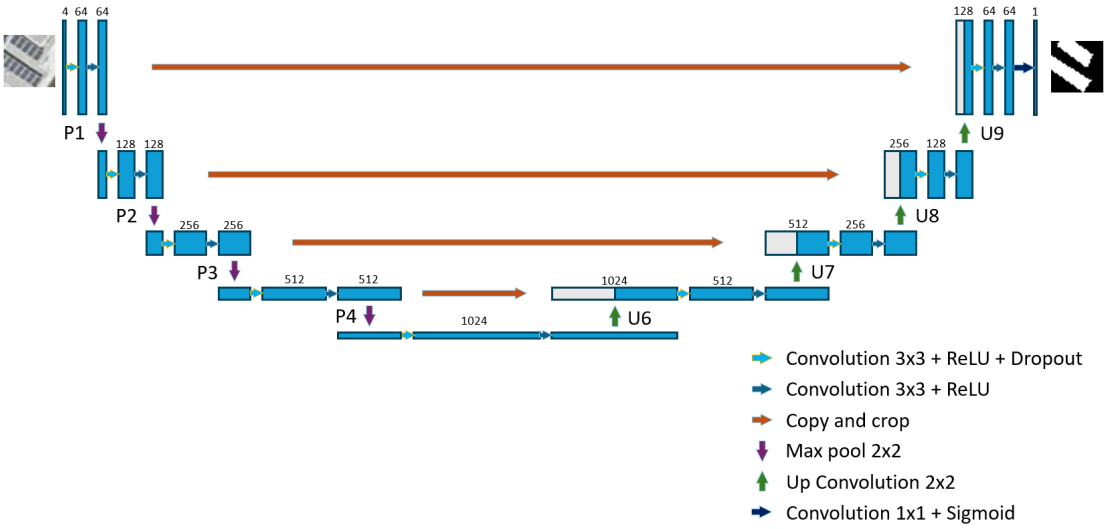


Figura 26 : Esquema de la arquitectura de la red del modelo U-Net

Nuestro modelo U-Net definido utiliza imágenes de entrada de tamaño 32x32 píxeles con 4 canales (RGBA). La ruta de codificación reduce gradualmente el tamaño espacial y aumenta la profundidad de los mapas de características mediante una serie de capas convolucionales y de *max pooling*. Cada bloque convolucional contiene dos capas de convolución 3x3, una función de activación ReLU y una capa Dropout para prevenir el sobreajuste. Las capas de *max pooling* se utilizan para el muestreo descendente. Los niveles de la ruta de codificación son los siguientes: primer bloque convolucional (64 filtros, tamaño de 32x32 a 16x16), segundo bloque convolucional (128 filtros, de 16x16 a 8x8), tercer bloque convolucional (256 filtros, de 8x8 a 4x4), cuarto bloque convolucional (512 filtros, de 4x4 a 2x2) y quinto bloque convolucional (1024 filtros, tamaño de 2x2).

La ruta de decodificación restaura gradualmente el tamaño espacial y reduce la profundidad de los mapas de características mediante capas de deconvolución (Conv2DTranspose). Cada bloque de deconvolución contiene una capa de deconvolución, dos capas de convolución 3x3, una función de activación ReLU y una conexión de salto con el mapa de características correspondiente de la ruta de codificación. Los niveles de la ruta de decodificación son los siguientes: primer bloque de deconvolución (tamaño de 2x2 a 4x4, 512 filtros, conexión de salto con el cuarto bloque convolucional), segundo bloque de deconvolución (de 4x4 a 8x8, 256 filtros, conexión de salto con el tercer bloque convolucional), tercer bloque de deconvolución (de 8x8 a 16x16, 128 filtros, conexión de salto con el segundo bloque convolucional) y

cuarto bloque de deconvolución (de 16x16 a 32x32, 64 filtros, conexión de salto con el primer bloque convolucional). La capa de salida final utiliza una convolución 1x1 para reducir el número de canales a 1 y una función de activación Sigmoid para producir la probabilidad de que cada píxel pertenezca a la región objetivo.

5.3.2. Entrenar modelo U-Net

Para entrenar nuestro modelo U-Net, creamos un script en Python (Anexo 11.2.2) para el entrenamiento y la evaluación del modelo. Al igual que el script utilizado para entrenar el modelo CNN-b, este script puede ejecutarse localmente o en Colab. Si se ejecuta localmente, es importante asegurarse de tener instalados las bibliotecas CUDA y cuDNN de NVIDIA para la aceleración con GPU.

A continuación, configuramos las imágenes y sus máscaras para el entrenamiento y la validación. Usamos una función “load_images_and_masks” para cargar las imágenes y las máscaras correspondientes desde el directorio especificado. Las imágenes se procesan en formato RGBA, mientras que las máscaras se procesan en formato de escala de grises. Si se desea usar imágenes en formato RGB, simplemente se debe cambiar el valor de “convert” a "RGB" y ajustar el número de canales de entrada del modelo a 3.

Después, normalizamos las imágenes y las máscaras para asegurar que los valores de los datos estén en el rango [0,1]. También binarizamos las máscaras para facilitar la tarea de segmentación. Durante el entrenamiento del modelo, comparamos el rendimiento con y sin aumentación de datos. Los parámetros de aumentación de datos son similares a los utilizados en el modelo CNN-b.

Luego, definimos la arquitectura del modelo U-Net según lo descrito en la sección anterior. Al compilar y entrenar el modelo, utilizamos el optimizador Adam y la función de pérdida de entropía cruzada binaria en los conjuntos de datos de entrenamiento y validación especificados. Después de completar el entrenamiento, guardamos el modelo y lo analizamos y evaluamos.

Además de generar los tres gráficos (los primeros dos muestran la curva de pérdida a lo largo de los ciclos de entrenamiento, el primero con el eje Y ajustado automáticamente y el segundo con el eje Y limitado entre 0 y 1, y el tercero muestra la curva de exactitud a lo largo de los ciclos de entrenamiento) como se hizo con el modelo CNN-b, también utilizamos la evaluación IoU (Intersección sobre Unión). Definimos dos funciones para evaluar el IoU usando todo el contenido del conjunto de validación, calculando el IoU entre las máscaras predichas y las máscaras reales, y obteniendo un valor promedio de IoU.

5.3.3. Evaluación y comparación de modelos U-Net

Evaluación IoU

La evaluación de IoU, es decir, la Intersección sobre Unión (Intersection over Union), es una métrica utilizada para evaluar el rendimiento de los modelos de segmentación de imágenes. El principal objetivo de la evaluación de IoU es medir el grado de superposición entre el resultado de la predicción y la segmentación real. Su fórmula de cálculo es la siguiente:

$$\text{IoU} = \frac{\textit{Área de intersección entre la predicción y la segmentación real}}{\textit{Área de unión entre la predicción y la segmentación real}}$$

Al calcular el valor de IoU, primero se determina la superposición (intersección) y el área total cubierta (unión) de los píxeles de la región de segmentación predicha y la región de segmentación real. Luego, se divide el número de píxeles en la intersección por el número de píxeles en la unión para obtener el valor de IoU [23].

Mediante el IoU, se puede evaluar cuantitativamente la precisión de los resultados de segmentación del modelo. El valor de IoU oscila entre 0 y 1; cuanto mayor sea el valor, más cercano estará el resultado predicho al resultado real. Podemos utilizar el valor de IoU para comparar el rendimiento y seleccionar el modelo óptimo.

Modelo U-Net básico

Primero, entrenamos el modelo utilizando la estructura definida de U-Net. A continuación, se muestran los gráficos de la pérdida y la exactitud (accuracy) a medida que aumenta el número de ciclos de entrenamiento.

Figura 27 : Figuras de exactitud y pérdida de modelo U-Net básico

El valor promedio de IoU obtenido mediante la evaluación del programa es 0.7806.

A partir del análisis de las curvas de pérdida de entrenamiento y validación, se puede observar que el modelo U-Net muestra un rendimiento "demasiado bueno" en los

datos de entrenamiento, con una rápida disminución de la pérdida de entrenamiento que tiende a cero. Sin embargo, la pérdida de validación muestra grandes fluctuaciones, es alta en general y es inestable, lo que indica un claro problema de sobreajuste. Además, la evaluación de IoU muestra un valor promedio de solo 0.78, lo que indica que todavía hay mucho espacio para la optimización del modelo.

Modelo U-Net con aumento de datos y capa Dropout

Para reducir la aparición del sobreajuste, introdujimos dos métodos de optimización. Primero, añadimos capas de dropout al definir el modelo U-Net. Dropout es una técnica de regularización que, durante el entrenamiento, ignora aleatoriamente algunas neuronas, impidiendo que el modelo dependa de rutas específicas y, por lo tanto, previniendo el sobreajuste. Además de prevenir el sobreajuste, dropout puede mejorar la robustez del modelo, lo que es particularmente adecuado para modelos simples[23]. En segundo lugar, utilizamos la aumentación de datos en tiempo real, con los mismos parámetros que seleccionamos anteriormente al entrenar el modelo CNN-b.

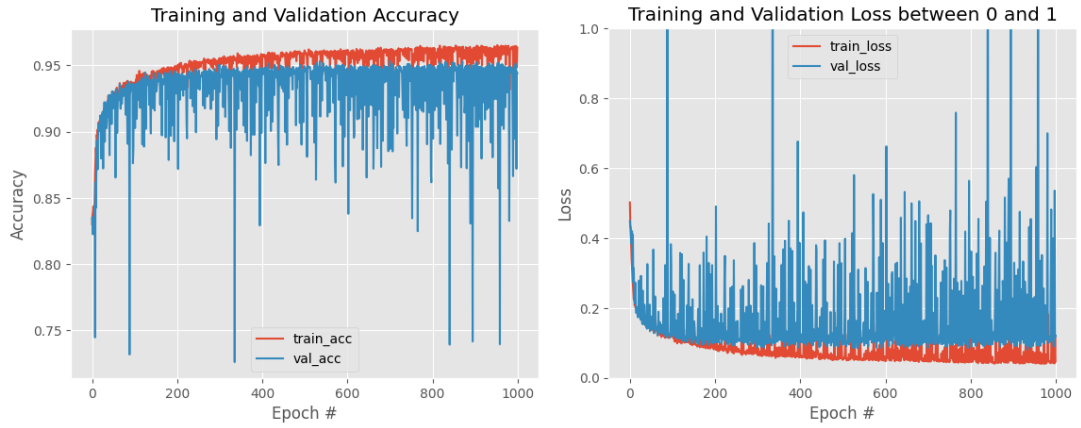


Figura 28 : Figuras de exactitud y pérdida de modelo U-Net básico

El valor promedio de IoU obtenido mediante la evaluación del programa es 0.8435. Después de los ajustes, las curvas de pérdida de entrenamiento y validación muestran una gran mejora en el problema de sobreajuste del modelo. Además, la pérdida de validación se ha reducido significativamente. La evaluación de IoU muestra un valor promedio de 0.85, la exactitud del modelo alcanzó superior de 0.95 y la pérdida es inferior a 0.1. Esto indica que la introducción de capas de Dropout y el uso de la aumentación de datos en tiempo real han mejorado considerablemente el rendimiento del modelo.

Modelo U-Net con mascara mixta de muestras “yes” y “no”

En el entrenamiento y evaluación anteriores del modelo U-Net, solo utilizamos el conjunto de muestras positivas (clase “yes”) y sus máscaras generadas para el

entrenamiento y la evaluación. Sin embargo, nos dimos cuenta de que las muestras negativas (clase “no”) también podrían tener un impacto en la identificación. Por lo tanto, volvimos a entrenar utilizando una combinación de los conjuntos de muestras “yes” y “no”, junto con sus máscaras. Los resultados son los siguientes:

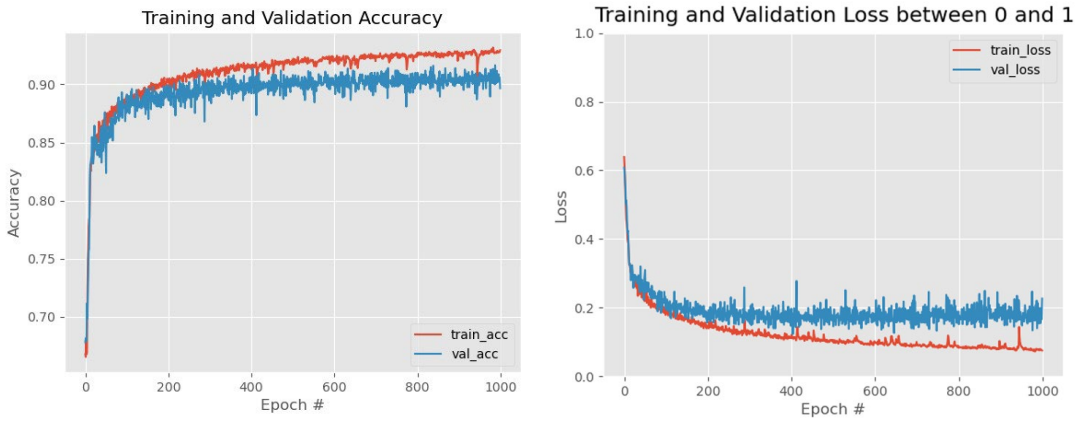


Figura 29 : Figuras de exactitud y perdida de modelo U-Net con mascara mixto de muestras “yes” y “no”

Se puede observar que la fluctuación en el conjunto de validación se ha reducido significativamente. Sin embargo, las curvas de pérdida y exactitud muestran que tanto la pérdida como la exactitud han empeorado en comparación con el ensayo anterior. El valor promedio de la evaluación IoU es ligeramente inferior al anterior.

Dado que no podemos evaluar detalladamente cuál enfoque es más beneficioso para la identificación de los paneles solares antes de aplicar el modelo, necesitamos aplicar el modelo y ajustar la estrategia basada en los resultados obtenidos.

5.4 Aplicar el modelo

Al aplicar el modelo y generar los archivos de polígonos vectoriales, utilizamos dos scripts en Python para completar esta tarea. Primero, utilizamos los dos modelos para identificar las subimágenes dentro del área de estudio y luego generamos archivos shapefile de 32x32 píxeles utilizando las máscaras resultantes. Luego, empleamos otro script para combinar todos los archivos shapefile "fragmentados" en un único multipolygon, utilizando la información geográfica de las subimágenes originales en formato TIFF. Este multipolygon se utilizará para investigaciones y análisis posteriores.

5.4.1. Aplicación del modelo y convertir al SHP

Se ha definido un script en Python (Anexo 11.3.1) para aplicar nuestros modelos a las imágenes dentro del área de estudio y convertir los resultados en archivos shapefile. Después de importar las bibliotecas necesarias, el script debe leer dos modelos y una carpeta que contiene numerosos archivos TIFF de 32x32 píxeles. Podemos cambiar las rutas de los archivos para seleccionar qué modelos utilizar para la identificación.

A continuación, el script convierte automáticamente las imágenes TIFF en formato PNG para aplicarle los modelos y generar máscaras. Al aplicar los modelos, primero utilizamos el modelo CNN-b para una evaluación preliminar. Este proceso puede configurar un umbral para la identificación de placas solares en la imagen, generalmente 0.5. Sin embargo, cuando el modelo CNN muestra menos verdaderos positivos que el modelo U-Net, se puede considerar reducir este umbral. Después de que una imagen pasa la evaluación del modelo CNN-b, se le aplica el modelo U-Net para identificar y generar la máscara. Al guardar la máscara, se realiza una verificación adicional: si la máscara es completamente cero (el modelo U-Net considera que no contiene paneles solares), no se guarda. Al guardar la máscara, también se lee la información geográfica contenida en la imagen TIFF original y se escribe en la máscara. Utilizamos el modelo CNN-b primero porque consume menos recursos computacionales, lo que reduce significativamente el tiempo de cálculo (por ejemplo, utilizar solo el modelo U-Net puede llevar 70 horas, mientras que la combinación de CNN y U-Net puede reducirlo a 20 horas). Luego, utilizamos las bibliotecas rasterio y shapely para convertir los datos de la máscara en shapefiles. La información geográfica escrita anteriormente es crucial en este paso, ya que también debe incluirse en los shapefiles para poder fusionarlos. El programa guarda automáticamente las máscaras generadas (formato TIFF) y los shapefiles para operaciones posteriores.

Además de la aplicación básica de los modelos, también introdujimos algunos módulos para mejorar la evaluación y el rendimiento. Primero, creamos un archivo CSV que registra el nombre del archivo de la imagen, el valor de predicción del modelo CNN-b, si contiene paneles solares (por ejemplo, si es mayor de 0.5 se considera que contiene) y si el modelo U-Net generó una máscara no nula después de la identificación. Este archivo CSV nos ayuda a analizar los resultados preliminares mediante una matriz de confusión y facilita la consulta de operaciones específicas. En cuanto al rendimiento, dado que la identificación de las subimágenes del área de estudio implica procesar más de 5 millones de imágenes. Esto supone una gran carga para la memoria y VRAM. Por lo tanto, limpiamos la memoria y VRAM cada 1000 archivos procesados. También introdujimos una barra de progreso utilizando la biblioteca “tqdm”, que nos informa

dinámicamente del tiempo restante aproximado.

5.4.2. Combinación ficheros SHP

En comparación con el script de aplicación de modelos, el script en Python para combinar todos los archivos Shapefile (Anexo 11.3.2) es bastante simple. Después de importar las bibliotecas necesarias, el script debe leer el directorio que contiene todos los archivos shapefile generados previamente. Al completar la operación de combinación, el script generará un único archivo de polígonos.
La operación de combinación de archivos de polígonos se realiza principalmente a través de las bibliotecas fiona y shapely. El script primero leerá todos los archivos shp. Inicializa una lista vacía de polígonos para almacenar todos los polígonos. Luego, el script recorre todos los archivos shapefile, primero leyendo su sistema de referencia de coordenadas (CRS) y luego leyendo sus objetos geométricos y convirtiéndolos en objetos geométricos de Shapely. Después, almacena estos objetos geométricos en la lista vacía creada anteriormente. Una vez leídos todos los polígonos, se usa la función unary_union para combinar todos los polígonos. Luego, se escriben los datos geométricos y los atributos en un nuevo archivo shapefile. Si el tipo es Polygon, se escribe directamente. Si el tipo es MultiPolygon, se recorre cada subpolígono y se escribe, asignando a cada subpolígono un ID único.

5.5 Evaluación y optimización

5.5.1. Evaluación resultados en QGIS

Para evaluar los resultados después de aplicar nuestro modelo, importamos el archivo shapefile de polígonos obtenido en QGIS. Comparamos las áreas de los paneles solares digitalizados manualmente con las áreas identificadas por el modelo para realizar el análisis. En este proceso, podemos observar y comparar claramente las áreas identificadas con las áreas marcadas.

Figura 30 : Primer resultado que detecta por modelos

El primer resultado que obtuvimos (Figura 30) fue muy decepcionante. Los resultados de identificación del modelo U-Net, tanto al usar solo muestras “yes” como al mezclar muestras “yes” y “no”, no fueron satisfactorios. Observamos que, aunque casi todos los paneles solares fueron correctamente identificados, hubo una gran cantidad de ruido que también fue marcado como paneles solares. Este ruido incluía sombras de vegetación y edificios, calles, estacionamientos, etc. Este resultado indica que tenemos un alto número de verdaderos positivos (true positives), pero también un alto número de falsos positivos (false positives). Este resultado no es aceptable, por lo que decidimos optimizar gradualmente nuestro muestreo para mejorar la precisión del modelo.

Además, observamos que los resultados del modelo U-Net entrenado solo con el conjunto de muestras “yes” fueron ligeramente mejores que los del modelo entrenado con el conjunto mixto de muestras “yes” y “no”. Sin embargo, dado que ninguno de los resultados fue satisfactorio, después de ajustar el modelo continuaremos utilizando ambos métodos de entrenamiento y comparando los resultados.

5.5.2. Entrenamiento con más muestras negativas

Para reducir los errores de identificación del modelo en áreas que no contienen paneles solares, decidimos introducir en el conjunto de muestras “no” algunas

muestras adicionales de las áreas que fueron identificadas incorrectamente. Esperamos que esto aumente la capacidad del modelo CNN y U-Net para reconocer estas áreas correctamente. Para ello, marcamos manualmente las áreas de diferentes tipos de errores de identificación.

Luego, utilizamos un script en Python para copiar todas las imágenes que contienen las áreas marcadas del conjunto de imágenes del área de estudio. El script recorre el directorio especificado, a través de la obtención de la información geográfica de las imágenes en formato TIFF, determina si se superponen con el archivo poligonal. Este script es similar a los que usamos anteriormente (Anexo 11.1.2), pero en esta ocasión optamos por copiar las imágenes a un directorio específico en lugar de moverlas. Este script es más completo, ya que utiliza procesamiento multihilo para invocar la aceleración de GPU y así acelerar la velocidad de procesamiento. Además, se añadió una herramienta de barra de progreso para prestar atención al tiempo de procesamiento aproximado y al tiempo restante. Después de obtener las muestras adicionales de "no", seleccionamos aleatoriamente una parte de estas muestras para agregarlas al conjunto original de muestras "no". Añadimos aproximadamente un 20% de nuevas muestras tanto al conjunto de entrenamiento "no" como al conjunto de prueba "no".

Figura 31 : Segundo resultado con más muestras negativas

Incluso después de agregar muestras negativas adicionales, los resultados (Figura 31)

de identificación del modelo siguen sin ser satisfactorios. No solo persisten numerosos falsos positivos, sino que también se eliminaron algunas áreas de paneles solares correctamente identificadas previamente. Esto indica que el modelo no puede distinguir claramente entre las áreas de ruido y las áreas correctas. Además, observamos que los resultados del modelo U-Net entrenado solo con muestras "yes" fueron peores que los del modelo entrenado con una mezcla de muestras "yes" y "no". Esto se debe a que el modelo U-Net entrenado solo con muestras "yes" no incluyó las muestras negativas adicionales en su entrenamiento, lo que llevó a una reducción en los falsos positivos, gracias a la mayor precisión del modelo CNN-b. En cualquier caso, necesitamos optimizar aún más el modelo.

5.5.3. Añadir banda infrarroja

Dado que encontramos que los resultados del modelo entrenado no eran óptimos al aplicarlo al área de estudio y sus alrededores, realizamos algunos cambios en los datos utilizados. En el primer entrenamiento de los modelos CNN binario y U-Net, usamos imágenes en modo de color RGB. Debido a que el modelo tendía a identificar erróneamente las áreas de vegetación, introdujimos la banda infrarroja, que puede distinguir mejor entre vegetación y paneles solares. La vegetación tiene valores muy altos en la banda infrarroja, mientras que los paneles solares tienen valores muy bajos. Para ello, volvimos a descargar imágenes ortográficas RGBI de cuatro bandas y resolución espacial de 25 cm que cubren el área de estudio desde el sitio web de ICV. Luego, al igual que con las imágenes ortográficas RGB, utilizamos QGIS y Python para combinar las imágenes (4.3.1) y dividirlas en subimágenes de 32x32 píxeles (4.3.3).

5.5.4. Optimización de muestras

Además de añadir la banda infrarroja, realizamos las siguientes optimizaciones:

Eliminación de muestras

Como se muestra en las siguientes imágenes (Figura 32), algunos paneles solares presentan reflejos que los hacen parecer borrosos. Al comparar las ortofotografías de 2023 que usamos con las ortofotografías de 2022 proporcionadas por el ICV, observamos que algunos paneles solares tienen reflejos y aparecen blancos en las imágenes de 2023, mientras que en 2022 se ven normales. Estos paneles solares "problemáticos" pueden causar problemas en el modelo, por lo que eliminaremos estas muestras.

Figura 32 : Comparación de ortofoto ICV RGB entre 2022 (derecha) y 2023 (izquierda)

Modificar muestra

Como se muestra en la imagen a continuación (Figura 33), al marcar áreas extensas de paneles solares, anteriormente utilizamos un solo polígono para marcar toda el área. Esto también podría afectar la capacidad del modelo para reconocer correctamente los paneles solares. Por lo tanto, refinamos nuevamente las marcas de áreas extensas, delimitándolas específicamente alrededor de los paneles solares.

Figura 33 : Comparación de modificación de paneles solares digitalización

Marcar zona para muestra “No” manualmente

Debido a que anteriormente seleccionamos aleatoriamente una cantidad determinada de muestras del directorio para formar el conjunto final de muestras "no", esto podría haber llevado a que el modelo no se entrenara adecuadamente en ciertas áreas específicas, como calles, coches, pasos de peatones, etc. Por lo tanto, decidimos marcar manualmente diversas áreas de interés (Figura 34). Durante el proceso de marcado, intentamos cubrir una variedad de tipos de áreas, como diferentes tipos de viviendas, áreas de sombra en viviendas, patios, pasos de peatones, estacionamientos, diferentes tipos de cultivos y otras áreas que fueron incorrectamente identificadas en evaluaciones anteriores.

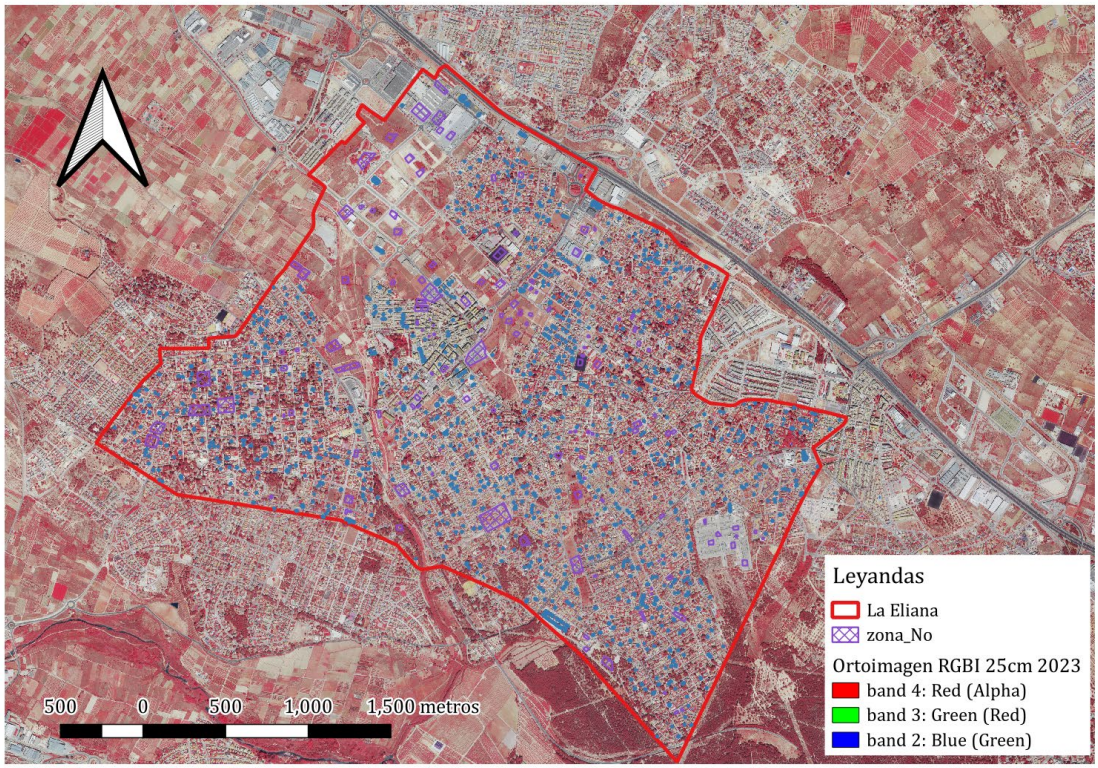


Figura 34 : Zona marcado para muestras "No" en una ortofoto de falsa color IRG

Posteriormente, utilizamos un script en Python (Anexo 11.1.7) para dividir las imágenes que incluyen las áreas marcadas y seleccionar el conjunto final de muestras "no".

Regeneración conjuntos de muestras

Usamos el mismo script (Anexo 11.1.7) para copiar las imágenes candidatas del conjunto de muestras "yes" a un directorio separado. Luego, de manera similar a los pasos anteriores (4.3.4), calculamos su área y utilizamos el script y el archivo CSV generado para filtrar las imágenes donde el área de los paneles solares ocupa más de un cierto porcentaje del área de la subimagen (Anexo 11.1.3). Después, utilizamos un script (Anexo 11.1.4) para mover estas imágenes al conjunto final de muestras "yes". Luego, seleccionamos aleatoriamente una cantidad determinada de imágenes de las candidatas del conjunto de muestras "no" para formar el conjunto final de muestras "no" (Anexo 11.1.5).

En el primer entrenamiento del modelo, los conjuntos de muestras "yes" y "no" tenían cada uno 1086 muestras. Esto se debió a que seleccionamos subimágenes que contenían paneles solares con un porcentaje mayor al 15% de la imagen. Al volver a entrenar el modelo utilizando ortofotografías de 4 bandas (RGBI), consideramos que algunas subimágenes con un menor porcentaje de paneles solares podrían contener más áreas de sombra. Estas imágenes podrían proporcionar mejores resultados al

entrenar el modelo U-Net. Después de revisar individualmente las subimágenes con un bajo porcentaje de paneles solares, seleccionamos las imágenes con un porcentaje mayor al 11.44% como el nuevo conjunto de muestras "yes". En total, este conjunto incluye 1405 subimágenes.

Mediante el uso de las áreas de muestreo "no" que marcamos manualmente, obtuvimos un total de 4416 subimágenes. Esta cantidad es mucho mayor que la del conjunto de muestras de entrenamiento "yes" que determinamos anteriormente. Por lo tanto, seleccionamos aleatoriamente 1405 subimágenes de ellas para formar el nuevo conjunto de entrenamiento "no".

Igual que antes, utilizamos el 80% de las imágenes (1124) como conjunto de entrenamiento y el 20% (281) como conjunto de prueba para conjunto de muestro “yes” y “no”.

5.5.5. Evaluación posterior a la optimización

Después de añadir la banda infrarroja y ajustar el muestreo, reentrenamos y aplicamos nuestros modelos. Los resultados en QGIS se muestran a continuación:

Figura 35 : Tercer resultado con banda infrarroja y muestreo optimizado

Podemos observar claramente que tanto el modelo entrenado solo con muestras "yes" como el modelo U-Net entrenado con una mezcla de muestras "yes" y "no" han

mejorado. Sin embargo, el modelo U-Net entrenado con la mezcla de muestras "yes" y "no" muestra un mejor rendimiento. Por lo tanto, utilizaremos este último modelo para ajustes adicionales.

Aunque hemos observado una mejora significativa en los resultados, todavía hay muchas zonas de calles que se identifican incorrectamente (Figura 36) como paneles solares.

Figura 36 : Área mal identificado (calles)

Por lo tanto, decidimos repetir la operación anterior (5.5.2), añadiendo nuevamente muestras de las carreteras al conjunto de muestras "no" utilizando el mismo método. La cantidad añadida fue el 20% del tamaño del conjunto de entrenamiento y del conjunto de prueba original.

Figura 37 : Cuarto resultado con banda infrarroja y más muestras negativas

Descubrimos que, aunque los errores en las carreteras se eliminaron en gran medida, las áreas que originalmente se identificaron correctamente como paneles solares también se redujeron. Esto indica que los verdaderos positivos (true positives) están

disminuyendo gradualmente, lo que sugiere que estamos acercándonos al límite de la capacidad de reconocimiento del modelo U-Net.

Para abordar esto, realizamos una prueba utilizando únicamente el modelo U-Net y descubrimos que el modelo U-Net puede identificar correctamente paneles solares completos. Los errores de identificación probablemente se deban principalmente a que el modelo CNN-b eliminó muchas imágenes que contenían paneles solares. Por lo tanto, utilizamos el archivo CSV generado para analizar los resultados mediante una matriz de confusión. Generamos un informe mediante un script en Python, que incluye indicadores importantes como precisión, exactitud y recall. A continuación, se muestra la matriz de confusión generada, que muestra los resultados de la identificación utilizando los modelos CNN y U-Net por separado.

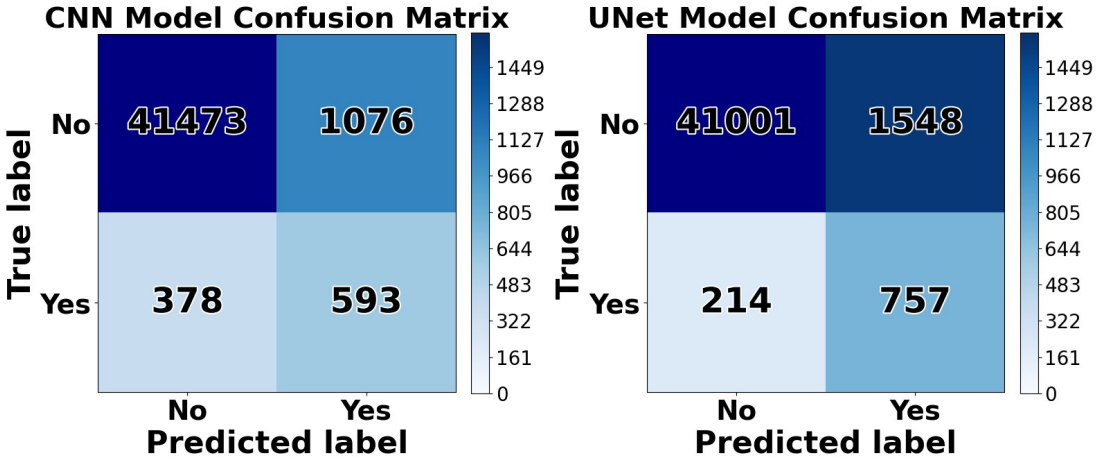


Figura 38 : Matriz de confusión de modelo CNN-b y modelo U-Net

A partir de la matriz de confusión, podemos observar que ambos modelos tienen un valor muy alto de True Negative. Esto se debe a la diferencia significativa entre la aplicación del modelo en el área de estudio y la evaluación del modelo utilizando solo el conjunto de validación. En la aplicación práctica del modelo, la mayoría de las áreas no contienen paneles solares. Por otro lado, al evaluar el modelo con el conjunto de validación, la cantidad de imágenes que contienen y no contienen paneles solares es aproximadamente igual. Aunque ambos modelos tienen un número considerable de False Positives, para el modelo CNN estamos más preocupados por los True Positives.

CNN Classification Report:

	precision	recall	f1-score	support
No	0.99	0.97	0.98	42549
Yes	0.36	0.61	0.45	971
accuracy			0.97	43520
macro avg	0.67	0.79	0.72	43520
weighted avg	0.98	0.97	0.97	43520

UNet Classification Report:

	precision	recall	f1-score	support
No	0.99	0.96	0.98	42549
Yes	0.33	0.78	0.46	971
accuracy			0.96	43520
macro avg	0.66	0.87	0.72	43520
weighted avg	0.98	0.96	0.97	43520

Figura 39 : Informe de modelo CNN

A partir del análisis del informe, podemos observar que no hay una diferencia significativa entre precisión y exactitud de los dos modelos. Ambos modelos muestran un buen desempeño en la identificación de muestras negativas, pero tienen deficiencias en la identificación de muestras positivas, lo que resulta en una baja tasa de recall. Sin embargo, la tasa de recall del modelo U-Net es un 17% más alta que la del modelo CNN-b.

Por lo tanto, decidimos optimizar el proceso ajustando el umbral de predicción del modelo CNN al aplicar el modelo. Al reducir el umbral de predicción, más imágenes, tanto correctas como incorrectas, pasarán por el modelo CNN y serán identificadas por el modelo U-Net. Aunque esto aumentará en cierta medida los falsos positivos (false positives), también incrementará los verdaderos positivos (true positives). Dado que nuestro objetivo principal es asegurar que todos los paneles solares sean identificados correctamente, intentaremos reducir el umbral de predicción del modelo CNN-b tanto como sea posible.

Figura 40 : Quinto resultado con umbral ajustado

6 Resultados

6.1 Detección de nuevos paneles solares

Después de completar todas las optimizaciones del modelo, aplicamos ambos modelos a las ortofotografías de 2022 y 2023 y obtuvimos sus respectivos archivos de polígonos correspondientes a las placas existentes en cada uno de estos dos años. Con estos archivos de polígonos, esperamos determinar cuántos paneles solares nuevos se añadieron en el transcurso del año. Para ello, ejecutamos un script en Python en QGIS (Anexo 11.4.1). El script se ejecuta directamente en la terminal de Python dentro de QGIS. El script lee los archivos de polígonos generados por los dos modelos para los años 2022 y 2023 dentro del área de estudio (L'Eliana). Luego, crea una nueva capa de polígonos para almacenar los polígonos identificados como nuevos paneles solares. Mientras tanto, el script registra continuamente la cantidad de paneles solares que ya existían y los nuevos paneles solares añadidos. Al final, imprime en la consola el número de paneles solares que no han cambiado y el número de nuevos paneles solares.

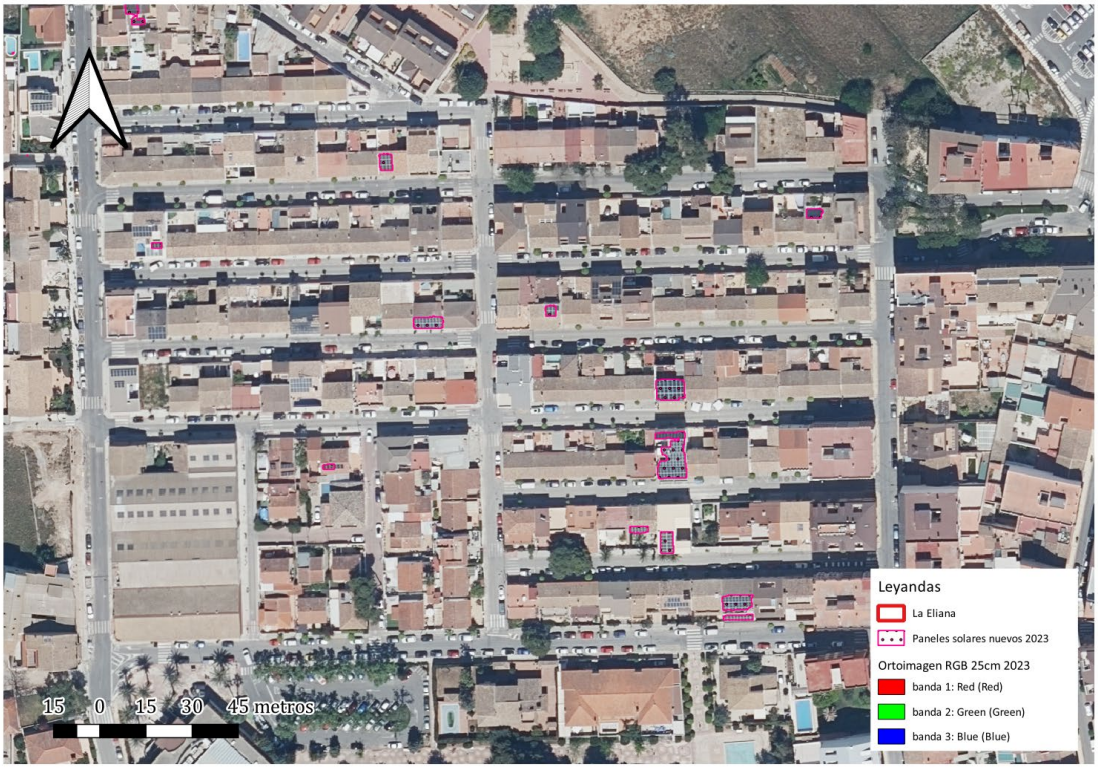


Figura 41 : Paneles solares actualizado con ortofoto del año 2023

Figura 42 : Paneles solares actualizado con ortofoto del año 2022

Las dos imágenes anteriores muestran los nuevos paneles solares marcados en un área pequeña, utilizando las imágenes ortográficas de ICV de 2022 y 2023, respectivamente. Hemos creado un atlas que cubre toda el área de estudio, y en Cartografía (10) se muestra una de las páginas de este atlas. El objetivo de esto es asegurar que cada mapa tenga una buena legibilidad. En los mapas se presentan todos los paneles solares detectados por el modelo en 2023 y se han marcado las nuevas instalaciones añadidas en 2023.

Según los resultados del código, en 2023 se identificaron 686 paneles solares que no cambiaron y se agregaron 1709 paneles solares nuevos. En las dos imágenes anteriores, podemos ver que las áreas de color rosa representan las zonas marcadas como nuevos paneles solares. Es posible que haya un margen de error en el número de nuevos paneles solares debido a una tasa ligeramente alta de falsos positivos en el modelo. Esto puede llevar a que se marquen algunas áreas que no contienen paneles solares como si las tuvieran.

Para abordar esto, realizamos una verificación manual de una pequeña sección del área de estudio. Esta operación nos permite determinar la confianza promedio en los resultados finales del área de estudio.

Después de realizar la verificación manual de una pequeña sección del área, descubrimos que casi todos los paneles solares de 2023 fueron correctamente identificados. De las 467 áreas digitalizadas manualmente, solo 23 paneles solares no fueron detectados, lo que indica una precisión de identificación del 95.07%. Este resultado es muy satisfactorio.

Sin embargo, de los 406 polígonos marcados como nuevos paneles solares en el área de estudio, solo 261 fueron identificados correctamente, mientras que 145 fueron identificaciones erróneas del modelo. Esto da una precisión de 64.28% en la identificación de nuevos paneles solares. Durante la verificación, observamos que la mayoría de las identificaciones erróneas correspondían a sombras de edificios y piscinas. Estas áreas son difíciles de distinguir de los paneles solares con una resolución espacial de 25 cm.

Estos resultados de la verificación manual indican que nuestro modelo final tiene un índice de falsos negativos muy bajo, pero un índice de falsos positivos relativamente alto. Esto significa que nuestro modelo puede identificar la gran mayoría de los paneles solares, pero es posible que clasifique erróneamente algunas áreas como paneles solares.

Considerando la precisión de la identificación de los paneles solares, podemos estimar que entre 2022 y 2023, se instalaron aproximadamente 1098 nuevos paneles solares en L'Eliana, lo que representa una tasa de crecimiento del 160.1%.

6.2 Posible mejora en futuro

A través de múltiples modificaciones y la verificación final del modelo, hemos observado que, con una resolución espacial de 25 cm, incluso utilizando imágenes ortográficas RGBI de 4 bandas, muchas áreas de sombra son fácilmente identificadas incorrectamente. Para abordar estos errores, tenemos algunas posibles soluciones de optimización.

Primero, podemos usar múltiples modelos, cada uno para identificar diferentes características. Dado que casi todos los paneles solares están ubicados en los techos de las casas, podríamos identificar primero las áreas donde se encuentran las casas y luego identificar los paneles solares en sus superficies.

En segundo lugar, observamos que las áreas identificadas incorrectamente tienen formas que pueden diferir de los paneles solares. Estas áreas suelen ser más pequeñas y más redondeadas. Por lo tanto, podemos optimizar los resultados mediante algunos procesos de post-procesado. Por ejemplo, creando clasificadores basados en árboles de decisión o reglas para filtrar los resultados.

7 Presupuesto

Basándonos en el salario promedio de los ingenieros de desarrollo y los ingenieros GIS en la Comunidad Valenciana, que es aproximadamente 2.096,83 euros mensuales, y sumando el coste de la seguridad social de los empleados (que representa alrededor del 40% del salario total), podemos estimar el coste laboral anual por empleado para la empresa. Este cálculo considera 14 meses de salario, lo que resulta en un coste anual de 41.088,07 euros por empleado. Además, se planea otorgar un bono de rendimiento o una paga extra de aproximadamente el 10% del salario anual.

Para calcular el tiempo de trabajo anual, descontamos 14 días de festivos oficiales y 14 días de vacaciones pagadas, resultando en un promedio de 1856 horas de trabajo al año por persona. Con esto, podemos determinar el coste promedio por hora de trabajo para cada empleado, que es aproximadamente 24,35 euros por hora. Con base en este coste, podemos estimar el presupuesto total del proyecto.

Presupuesto General de Contrata se calcula basado en P.E.M. (presupuesto ejecución material), G.G. (Gastos Generales) y B.I. (beneficio industrial) según siguiente formula:

$$P.G.C. = P.E.M. + G.G. + B.I.$$

Actividades	Duración	Coste
Estudio en profundidad CNN y U-Net	150 horas	3.652,5 €
Análisis de datos ICV 25cm RGB/RGBI	8 horas	194,8 €
Creación de programa Python en preproceso de datos	110 horas	2.678,5 €
Creación de programa Python en entrenamiento de modelos	150 horas	3.652,5 €
Creación de programa Python para aplicar modelo CNN y N-Net y obtener Shapefile	120 horas	2.922 €
Depurando el código Python.	72 horas	1.753,2 €
Análisis de resultados	40 horas	974 €
Búsqueda de información para la memoria	30 horas	730,5 €
Redacción, diseño y en maquetación para elaboración de la memoria	90 horas	2.191,5 €
Total	770 horas	18.749,5 €

Tablas 1 : Coste directo del proyecto

La duración total del proyecto es 770 horas con coste personal (recursos humanos) total 18.749,5€. Por lo que el tiempo para completar el trabajo es de aproximadamente 4,5 meses. Calculamos los costos indirectos en base a un período de 5 meses.

Gastos anuales de tipo general	Costes Anuales	Costes para el proyecto (5 meses)
Agua y Luz	1600€	666,67 €
Alquiler de Oficina	9000€	3.750 €
Amortización de equipos informáticos	800€	333,33 €
Internet y Telefonía	1500€	625 €
Mobiliario de Oficina	300€	125 €
Papelería	600€	250 €
Seguro Multirriesgo Oficina	300€	125 €
Riesgos por impagos	2500€	1.041,67 €
Limpieza	800€	333.33 €
Total	17400€	7.250 €

Tablas 2 : Coste indirecto del proyecto

La suma del coste directo e indirecto son 25.999,5€ (valor de P.G.M).

Normalmente los gastos generales y beneficio industrial se deducen 15% del P.G.M. que son 3.899,93€. Por lo tanto, el presupuesto general de contrata es 29.899,43€.

8 Conclusión

En este trabajo, utilizamos las ortofotografías proporcionadas por el ICV para entrenar y aplicar un modelo CNN binario y un modelo U-Net. A través de ajustes y evaluaciones repetidas, finalmente obtuvimos los modelos más adecuados. Combinando ambos modelos, identificamos pequeños paneles solares en el área de L'Eliana, en la provincia de Valencia, y evaluamos su crecimiento.

Nuestros dos modelos mostraron una precisión muy alta en la identificación de muestras negativas. Sin embargo, aún existe margen de mejora en la identificación de muestras positivas. Los modelos lograron identificar casi todos los paneles solares, pero también marcaron incorrectamente algunas áreas de sombra como paneles solares.

Según el análisis realizado con nuestros modelos para los años 2022 y 2023 en L'Eliana, se instalaron 1098 nuevos paneles solares en 2023, lo que representa un aumento del 160.1% respecto a 2022.

El uso del modelo U-Net en investigaciones relacionadas con la ciencia de la teledetección también es muy viable. U-Net puede generar eficazmente máscaras de las áreas necesarias para un análisis posterior, lo que ofrece una ventaja significativa en comparación con los modelos CNN tradicionales que solo pueden predecir por imagen.

En conclusión, creemos que el uso del aprendizaje profundo para identificar objetivos pequeños es muy factible, pero aún presenta muchos desafíos. En particular, es difícil para el modelo distinguir entre áreas de sombra y áreas correctas. También hemos propuesto algunas posibles soluciones de optimización para futuras investigaciones.

9 Bibliografía

[1] Janusz, P., & Kaliski, M. (2018). Prospects for the use of LNG terminals to meet the demand for natural gas in the EU. Polityka Energetyczna, 21(3), 69–80. https://doi.org/10.33223/epj/103677

[2] Gielen, D., Boshell, F., Saygin, D., Bazilian, M. D., Wagner, N., & Gorini, R. (2019). The role of renewable energy in the global energy transformation. Energy Strategy Reviews, 24, 38–50. https://doi.org/10.1016/j.esr.2019.01.006

[3] Bórawski, P., Holden, L., & Bełdycka-Bórawska, A. (2023). Perspectives of photovoltaic energy market development in the european union. Energy, 270, 126804. https://doi.org/10.1016/j.energy.2023.126804

[4] Kruitwagen, L., Story, K. T., Friedrich, J., Byers, L., Skillman, S., & Hepburn, C. (2021). A global inventory of photovoltaic solar energy generating units. Nature,

598(7882), 604–610. https://doi.org/10.1038/s41586-021-03957-7

[5] Zhang, X., Xu, M., Wang, S., Huang, Y., & Xie, Z. (2022). Mapping photovoltaic power plants in China using Landsat, random forest, and Google Earth Engine. Earth System Science Data, 14(8), 3743–3755. https://doi.org/10.5194/essd-14-3743-2022

[6] Wang, X., Xiao, X., Zhang, X., Ye, H., Dong, J., He, Q., Wang, X., Liu, J., Li, B., & Wu, J. (2023). Characterization and mapping of photovoltaic solar power plants by Landsat imagery and random forest: A case study in Gansu Province, China. Journal of Cleaner Production, 417, 138015. https://doi.org/10.1016/j.jclepro.2023.138015

[7] De Información Geográfica Instituto Geográfico Nacional, O. C. N. (n.d.). Plan Nacional de Ortofotografía Aérea. Plan Nacional De Ortofotografía Aérea. https://pnoa.ign.es/web/portal/pnoa-imagen/presentacion

[8] Ortofotos - ICV - Generalitat Valenciana. (n.d.). ICV. https://icv.gva.es/va/ortofotos

[9] Google Colab. (n.d.). https://research.google.com/colaboratory/faq.html

[10] Google Colab. (n.d.-b). https://colab.research.google.com/notebooks/intro.ipynb

[11] Google Colab. (n.d.-c). https://colab.research.google.com/notebooks/io.ipynb

[12] Project Jupyter Documentation — Jupyter Documentation 4.1.1 alpha documentation. (n.d.). https://docs.jupyter.org/en/latest/

[13] Colab paid services pricing. (n.d.). https://colab.research.google.com/signup?utm_source=resource_tab&utm_medium=link&utm_campaign=payg_learn_more

[14] Debug | PyCharm. (n.d.). PyCharm Help. https://www.jetbrains.com/help/pycharm/debugging-code.html

[15] NVIDIA CUDA GPUs - Compute Capability. (n.d.). NVIDIA Developer. https://developer.nvidia.com/cuda-gpus

[16] NVIDIA cuDNN — NVIDIA cuDNN v9.2.0 documentation. (n.d.). https://docs.nvidia.com/deeplearning/cudnn/latest/

[17] Distributed training with Keras. (n.d.). TensorFlow. https://www.tensorflow.org/tutorials/distribute/keras

[18] Chollet, B. F. (n.d.). The Keras Blog - Francois Chollet. https://blog.keras.io/author/francois-chollet.html

[19] Fiona: access to simple geospatial feature data — Fiona documentation. (n.d.). https://fiona.readthedocs.io/en/latest/

[20] Qgis. (n.d.). GitHub - qgis/QGIS: QGIS is a free, open source, cross platform (lin/win/mac) geographical information system (GIS). GitHub. https://github.com/qgis/QGIS

[21] Ortofotos - ICV - Generalitat Valenciana. (n.d.). ICV. https://icv.gva.es/es/ortofotos

[22] Deep learning. (n.d.-b). NVIDIA Developer. https://developer.nvidia.com/deep-learning

[23] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT Press.

[24] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15(1), 1929–1958. https://jmlr.csail.mit.edu/papers/volume15/srivastava14a/srivastava14a.pdf

[25] Ioffe, S., & Szegedy, C. (2015, February 11). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv.org. https://arxiv.org/abs/1502.03167

[26] Neal, R. M. (2007). Pattern Recognition and Machine Learning, by Christopher M. Bishop. Technometrics, 49. http://asq.org/technometrics/2007/08/pattern-recognition-and-machine-learning.pdf

[27] Huang, J. Z. (2014). An Introduction to Statistical Learning: With Applications in R by Gareth James, Trevor Hastie, Robert Tibshirani, Daniela Witten. Journal of Agricultural, Biological, and Environmental Statistics, 19(4), 556–557. https://doi.org/10.1007/s13253-014-0179-9

[28] Ronneberger, O., Fischer, P., & Brox, T. (2015, May 18). U-NET: Convolutional Networks for Biomedical Image Segmentation. arXiv.org. https://arxiv.org/abs/1505.04597

[29] Devi, B. R. (2022, January 6). Tips for training a 3D-Unet model for segmentation tasks. Medium. https://bhavya-rema.medium.com/tips-for-training-a-3d-unet-model-for-segmentation-tasks-5232851d0116

[30] Dui, Z., Huang, Y., Jin, J., & Gu, Q. (2023). Automatic detection of photovoltaic facilities from Sentinel-2 observations by the enhanced U-Net method. Journal of Applied Remote Sensing, 17(01). https://doi.org/10.1117/1.jrs.17.014516

10 Cartografía

Utilizamos las imágenes ortográficas de 25 cm de resolución espacial proporcionadas por ICV para 2023 como mapa base. Cartografiamos los nuevos paneles solares identificados por el modelo entre 2022 y 2023. Para mejorar la legibilidad del mapa, creamos un atlas que cubre toda el área de estudio. El atlas está en tamaño A2 y cada mapa tiene una escala de 1:1000. A continuación, se muestra una de las páginas de nuestro atlas.

Reemplazar por mapa después de generar el PDF

11 Anexo

11.1 Scripts Python para preproceso de datos

11.1.1. Segmentar imagen dentro zona

```
import rasterio
from rasterio.mask import mask
import geopandas as gpd
from shapely.geometry import box
from pathlib import Path
image_path = r"E:/<…>/orto 2023 laeriana ICV RGBI 25cm merge.tif"
shp_path = r"E:/<…>/L’ Eliana 25830.shp"
output_dir = r"F:\ <…>\seg_image"
def split_satellite_image(image_path, shp_path, output_dir, tile_size_pixels=(32, 32)):
    try:
        # Leer el archivo shp del límite de la ciudad
        city_boundary = gpd.read_file(shp_path)
        with rasterio.open(image_path) as src:
            print(f"Sistema de coordenadas de la imagen de satélite: {src.crs}")
            # Convertir el límite de la ciudad al mismo sistema de referencia de coordenadas (CRS) que la imagen de satélite
            city_boundary = city_boundary.to_crs(src.crs)
            print(f"Sistema de coordenadas del archivo shp: {city_boundary.crs}")
            # Asegurarse de que el directorio de salida exista
            Path(output_dir).mkdir(parents=True, exist_ok=True)
            for x in range(0, src.width, tile_size_pixels[0]):
                for y in range(0, src.height, tile_size_pixels[1]):
                    # Calcular los límites geográficos del mosaico actual
                    left, top = src.transform * (x, y)
                    right, bottom = src.transform * (x + tile_size_pixels[0], y + tile_size_pixels[1])
                    # Crear el cuadro de límites del mosaico
                    tile_box = box(left, bottom, right, top)
                    # Verificar si el mosaico está al menos parcialmente dentro del límite de la ciudad
                    if city_boundary.intersects(tile_box).any():
                        # Recortar el mosaico usando el método mask
                        out_image, out_transform = mask(src, [tile_box], crop=True)
                        # Actualizar los metadatos de la imagen de salida
                        out_meta = src.meta.copy()
                        out_meta.update({
                            "driver": "GTiff",
                            "height": tile_size_pixels[1],
                            "width": tile_size_pixels[0],
                            "transform": out_transform,
                            "count": src.count,    # Asegurarse de escribir todas las bandas
                            "compress": "lzw"    # Especificar el uso de compresión LZW
```

```
                    })
                    # Guardar el mosaico
                    output_filepath = Path(output_dir) / f"tile_{x}_{y}.tif"
                    with rasterio.open(output_filepath, "w", **out_meta) as dest:
                        dest.write(out_image)
                        print(f"Guardado: {output_filepath}")
    except Exception as e:
        print(f"Error: {e}")
# Llamar a la función
split_satellite_image(image_path, shp_path, output_dir)
```

11.1.2. Seleccionar muestras con SHP

```
import os
import shutil
import geopandas as gpd
import rasterio
from rasterio.features import geometry_mask
from shapely.geometry import box

samples_shp_path = r'E:/ <...>/placa_solares.shp'
image_folder = r'E:/ <...>/train data'
yes_folder = r'E:/ <...>/train data si'
no_folder = r'E:/ <...>/train data no'
# Asegurarse de que las carpetas de destino existan
os.makedirs(yes_folder, exist_ok=True)
os.makedirs(no_folder, exist_ok=True)
# Cargar el archivo shp
samples_gdf = gpd.read_file(samples_shp_path)
def image_contains_samples(image_path, samples_gdf):
    with rasterio.open(image_path) as img:
        # Obtener los límites geográficos de la imagen
        bounds = img.bounds
        img_box = box(*bounds)
    # Comprobar si los límites geográficos de la imagen se intersectan con cualquier polígono de muestras
    return samples_gdf.intersects(img_box).any()
# Recorrer la carpeta de imágenes
for image_name in os.listdir(image_folder):
    if image_name.endswith('.tif'):    # Asegurarse de que se procesan imágenes TIFF
        image_path = os.path.join(image_folder, image_name)
        # Determinar si la imagen contiene la zona de muestras
        if image_contains_samples(image_path, samples_gdf):
            # Mover a la carpeta "yes"
            shutil.move(image_path, os.path.join(yes_folder, image_name))
        else:
            # Mover a la carpeta "no"
            shutil.move(image_path, os.path.join(no_folder, image_name))
print("program done")
```

11.1.3. Calcular áreas

```
import rasterio
import geopandas as gpd
from shapely.geometry import box
import pandas as pd
import os
# Variables de ruta
samples_shp_path = r'E:/ <...>/municipio de trabajo/placa_solares3.shp'
image_folder = r'E:/ <...>/train data'
output_csv = r'E:/ <...>/output_percentage.csv'
# Cargar el archivo shp
samples_gdf = gpd.read_file(samples_shp_path)
# Preparar la lista para guardar los resultados
results = []
# Área total de la imagen (en metros cuadrados), cada píxel representa 0.25m * 0.25m, en total 32*32 píxeles
total_area = (32 * 0.25) * (32 * 0.25)
# Crear un GeoDataFrame vacío para almacenar los límites de todas las imágenes
img_bounds_list = []
# Leer cada archivo de imagen y calcular sus límites geográficos
i = 0
for image_name in os.listdir(image_folder):
    if image_name.endswith('.tif'):
        image_path = os.path.join(image_folder, image_name)
        with rasterio.open(image_path) as img:
            # Obtener los límites geográficos de la imagen y crear un polígono
            img_box = box(*img.bounds)
            img_bounds_list.append({'geometry': img_box, 'image_name': image_name})
        print(f"Processed bounds for {i} image: {image_name}")
        i += 1
# Generar GeoDataFrame
img_bounds_gdf = gpd.GeoDataFrame(img_bounds_list, crs=samples_gdf.crs)
# Crear un índice espacial para mejorar la eficiencia
img_bounds_gdf.sindex
# Calcular el porcentaje del área de cada polígono en cada imagen
for index, poly in samples_gdf.iterrows():
    # Encontrar los límites de imagen que se intersecan con el polígono actual
    possible_matches_index = list(img_bounds_gdf.sindex.intersection(poly.geometry.bounds))
    possible_matches = img_bounds_gdf.iloc[possible_matches_index]
    # Para cada coincidencia posible, calcular el porcentaje de área de intersección real
    for _, img_bound in possible_matches.iterrows():
        # Calcular el área de la intersección real
        intersection = poly.geometry.intersection(img_bound['geometry'])
        # Si la intersección existe, calcular el porcentaje de área
        if not intersection.is_empty:
            intersection_area = intersection.area
            percentage = (intersection_area / total_area) * 100
            print(f"Image: {img_bound['image_name']} - Polygon ID: {index} - Percentage: {percentage:.2f}%")
```

```
                results.append({"Image Name": img_bound['image_name'], "Polygon ID": index, "Percentage": percentage})
# Guardar los resultados en un archivo CSV
df = pd.DataFrame(results)
df.to_csv(output_csv, index=False)
print(f"Results saved to {output_csv}")
```

11.1.4. Mover muestras con CSV

```
import pandas as pd
import shutil
import os
# Definir carpetas de origen y destino
source_folder = r'E:/ <...>/train all yes'    # Ruta de la carpeta de origen
destination_folder = r'E:/ <...>/train yes'    # Ruta de la carpeta de destino
# Leer archivo CSV
csv_file = r'E:/ <...>/output_percentage_gt15.csv'    # Ruta del archivo CSV
data = pd.read_csv(csv_file)
# Asegurarse de que la carpeta de destino exista
if not os.path.exists(destination_folder):
    os.makedirs(destination_folder)
# Recorrer las filas del CSV
for index, row in data.iterrows():
    file_name = row['Image Name']
    file_path = os.path.join(source_folder, file_name)
    # Comprobar si el archivo existe en la carpeta de origen
    if os.path.exists(file_path):
        # Mover archivo
        shutil.move(file_path, destination_folder)
        print(f'El archivo {file_name} se ha movido con éxito.')
    else:
        print(f'El archivo {file_name} no existe en la carpeta de origen.')
print('Movimiento de archivos completado.')
```

11.1.5. Mover fichero aleatorio

```
import os
import random
import shutil

source_directory = r' E:/ <...>/train all no'    # Ruta de la carpeta de origen
target_directory = r' E:/ <...>/train no'    # Ruta de la carpeta de destino
number_of_images_to_move = 1086    # Cantidad de imágenes a mover
def move_random_images(source_dir, target_dir, num_images):
    # Verificar si las carpetas de origen y destino existen
    if not os.path.exists(source_dir):
```

```
        raise ValueError(f"La carpeta de origen '{source_dir}' no existe.")
    if not os.path.exists(target_dir):
        os.makedirs(target_dir)
    # Obtener todas las imágenes .tif
    all_images = [f for f in os.listdir(source_dir) if f.endswith('.tif')]
    image_count = len(all_images)
    # Imprimir el número total de imágenes
    print(f"Cantidad total de imágenes .tif en '{source_dir}': {image_count}")
    # Si la cantidad de imágenes a mover es mayor que la cantidad de imágenes disponibles, lanzar una excepción
    if num_images > image_count:
        raise ValueError("El número de imágenes a mover es mayor que el número de imágenes disponibles.")
    # Seleccionar aleatoriamente la cantidad especificada de imágenes
    selected_images = random.sample(all_images, num_images)
    # Mover las imágenes
    for image in selected_images:
        source_path = os.path.join(source_dir, image)
        target_path = os.path.join(target_dir, image)
        shutil.move(source_path, target_path)
        print(f"Se movió '{image}' a '{target_dir}'")
move_random_images(source_directory, target_directory, number_of_images_to_move)
```

11.1.6. Generar mascaras para U-Net

```
import os
import geopandas as gpd
import rasterio
from shapely.geometry import box
from rasterio.features import rasterize
from pathlib import Path
from PIL import Image
import numpy as np

# Parámetros de ruta de archivos
mask_dir = r"F:\ <...>\test"
tile_dir = r"F:\ <...>\unet_mask"
shp_path = r"E:\ <...>\placa_solares.shp"
# Asegurarse de que el directorio de salida exista
Path(mask_dir).mkdir(parents=True, exist_ok=True)
# Leer datos vectoriales
gdf = gpd.read_file(shp_path)
def generate_mask_for_tile(tile_path, gdf, mask_dir):
    with rasterio.open(tile_path) as src:
        # Convertir datos vectoriales al sistema de coordenadas de la imagen
        gdf_transformed = gdf.to_crs(src.crs)
        # Crear el polígono de los límites geográficos del bloque de imagen actual
        src_bounds_polygon = box(*src.bounds)
        # Encontrar polígonos que intersectan con el bloque de imagen actual
        gdf_intersected = gdf_transformed[gdf_transformed.intersects(src_bounds_polygon)]
```

```
            # Crear una máscara en blanco
            mask = np.zeros((src.height, src.width), dtype=np.uint8)
            if gdf_intersected.empty:
                print(f"No hay geometría válida dentro de los límites de {tile_path}")
            else:
                # Recortar polígonos y rasterizar
                for geom in gdf_intersected.geometry:
                    clipped_geom = geom.intersection(src_bounds_polygon)
                    mask |= rasterize([(clipped_geom, 255)], out_shape=src.shape, fill=0, transform=src.transform, dtype=np.uint8)
            # Guardar la máscara
            mask_image = Image.fromarray(mask)
            mask_path = os.path.join(mask_dir, f"mask_{os.path.basename(tile_path)}")
            mask_image.save(mask_path.replace('.tif', '.png'), format='PNG')
            print(f"Máscara guardada: {mask_path.replace('.tif', '.png')} con valores únicos: {set(np.ravel(mask))}")
# Procesar todos los bloques de imagen
for tile_filename in os.listdir(tile_dir):
    tile_path = os.path.join(tile_dir, tile_filename)
    generate_mask_for_tile(tile_path, gdf, mask_dir)
print("Procesamiento completado")
```

11.1.7. Copiar fichero con SHP

```
import os
import shutil
import geopandas as gpd
import rasterio
from shapely.geometry import box
from concurrent.futures import ThreadPoolExecutor, as_completed
from tqdm import tqdm

# Configurar rutas
shp_file = r'E:\ <...>\zona_No.shp'
source_folder = r'F:\ <...>\seg_image'
dest_folder = r'F:\ <...>\entrena_no'

def process_file(file_path, polygons, dest_folder):
    try:
        # Leer archivo tif
        with rasterio.open(file_path) as src:
            # Obtener los límites del archivo tif
            bounds = src.bounds
            bbox = gpd.GeoDataFrame({'geometry': [box(bounds.left, bounds.bottom, bounds.right, bounds.top)]},
                                    crs=src.crs)
            # Comprobar si el archivo tif se intersecta con algún polígono
            intersects = bbox.intersects(polygons.unary_union)
            if intersects[0]:
                # Copiar archivo tif a la carpeta de destino
```

```
                shutil.copy(file_path, dest_folder)
                return f"Copiado {os.path.basename(file_path)} a {dest_folder}"
            else:
                return f"{os.path.basename(file_path)} no se intersecta con el shapefile."
    except Exception as e:
        return f"Error al procesar {os.path.basename(file_path)}: {e}"
def copy_intersecting_tiles(shp_file, source_folder, dest_folder, batch_size=500):
    # Crear carpeta de destino (si no existe)
    if not os.path.exists(dest_folder):
        os.makedirs(dest_folder)
    # Leer archivo shapefile
    polygons = gpd.read_file(shp_file)
    # Obtener todos los archivos tif en la carpeta de origen
    tif_files = [os.path.join(source_folder, f) for f in os.listdir(source_folder) if f.endswith('.tif')]
    # Inicializar barra de progreso
    with tqdm(total=len(tif_files), desc="Procesando archivos") as pbar:
        # Procesar archivos por lotes
        for i in range(0, len(tif_files), batch_size):
            batch_files = tif_files[i:i + batch_size]
            # Usar multi-threading para procesar los archivos
            with ThreadPoolExecutor() as executor:
                futures = {executor.submit(process_file, file_path, polygons, dest_folder): file_path for file_path in
                           batch_files}
                for future in as_completed(futures):
                    print(future.result())
                    pbar.update(1)
copy_intersecting_tiles(shp_file, source_folder, dest_folder)
```

11.1.8. Convertir TIF al PNG

```
from PIL import Image
import os

def convert_tif_to_png(source_dir, target_dir):
    if not os.path.exists(target_dir):
        os.makedirs(target_dir)
    for filename in os.listdir(source_dir):
        if filename.endswith(".tif"):
            filepath = os.path.join(source_dir, filename)
            # Load the TIFF image
            image = Image.open(filepath)
            # Convert the image to PNG and save it
            png_path = os.path.join(target_dir, filename.replace(".tif", ".png"))
            image.save(png_path, "PNG")
source_train_yes_dir = r'F:\<...>\train\yes'
target_train_yes_dir = r'F:\<...>\train_png\yes'
source_test_yes_dir = r'F:\<...>\test\yes'
target_test_yes_dir = r'F:\<...>\test_png\yes'
```

```
source_train_no_dir = r'F:\<...>\train\no'
target_train_no_dir = r'F:\<...>\train_png\no'
source_test_no_dir = r'F:\<...>\test\no'
target_test_no_dir = r'F:\<...>\test_png\no'

# Convert images in train and test directories
convert_tif_to_png(source_dir=source_train_yes_dir, target_dir=target_train_yes_dir)
convert_tif_to_png(source_dir=source_test_yes_dir, target_dir=target_test_yes_dir)
convert_tif_to_png(source_dir=source_train_no_dir, target_dir=target_train_no_dir)
convert_tif_to_png(source_dir=source_test_no_dir, target_dir=target_test_no_dir)
```

11.2 Scripts Python para entrenamiento de modelos

11.2.1. Entrenamiento de modelos CNN binarios

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Conv2D, Input, Activation, Flatten, Dense, Dropout, BatchNormalization, MaxPooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import classification_report, accuracy_score, f1_score, matthews_corrcoef
import os
import matplotlib

matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
os.environ["PYTHONIOENCODING"] = "utf-8"
gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
    try:
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)
        logical_gpus = tf.config.experimental.list_logical_devices('GPU')
        print(len(gpus), "Physical GPUs,", len(logical_gpus), "Logical GPUs")
    except RuntimeError as e:
        print(e)

# Configurar rutas
data_dir = "F:\ <...>\data_RGBI"
train_data_dir = os.path.join(data_dir, 'train_ultra_png')
test_data_dir = os.path.join(data_dir, 'test_ultra_png')
model_save_path = "F:/ <...>/CNN_b_RGBI.h5"
```

```
# Definir la arquitectura del modelo
def deep_CNN(width, height, depth, classes=1, batchNorm=True):
    inputs = Input(shape=(height, width, depth))
    # Primer conjunto de capas: CONV => RELU => CONV => RELU => POOL
    x1 = Conv2D(32, (3, 3), padding="same", activation="relu")(inputs)
    if batchNorm:
        x1 = BatchNormalization()(x1)
    x1 = Conv2D(32, (3, 3), padding="same", activation="relu")(x1)
    if batchNorm:
        x1 = BatchNormalization()(x1)
    x1 = MaxPooling2D(pool_size=(2, 2))(x1)
    x1 = Dropout(0.25)(x1)
    # Capas completamente conectadas: FC => RELU
    xfc = Flatten()(x1)
    xfc = Dense(512, activation="relu")(xfc)
    if batchNorm:
        xfc = BatchNormalization()(xfc)
    xfc = Dropout(0.5)(xfc)
    # Clasificador softmax
    predictions = Dense(classes, activation="sigmoid")(xfc)
    # Construir el modelo
    model = Model(inputs=inputs, outputs=predictions)
    return model
# Función de normalización de imágenes
def normalize(image, label):
    return tf.cast(image / 255.0, tf.float32), label
# Generador de datos de imágenes
datagen = ImageDataGenerator(
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.1,
    horizontal_flip=True,
    vertical_flip=True,
    fill_mode='nearest',
    rescale=1. / 255,
    validation_split=0.2
)
# Preparar conjuntos de entrenamiento y validación
train_datagen = datagen.flow_from_directory(
    train_data_dir,
    target_size=(32, 32),
    color_mode='rgba',
    class_mode='binary',
    batch_size=32,
    shuffle=True,
    seed=1,
    subset='training'
```

```
)
val_datagen = datagen.flow_from_directory(
    train_data_dir,
    target_size=(32, 32),
    color_mode='rgba',
    class_mode='binary',
    batch_size=32,
    shuffle=True,
    seed=1,
    subset='validation'
)
test_dataset = tf.keras.preprocessing.image_dataset_from_directory(
    test_data_dir,
    labels="inferred",
    color_mode="rgba",
    label_mode="binary",
    image_size=(32, 32),
    batch_size=32
)
test_dataset = test_dataset.map(normalize)
# 1349+1124 / 1124
# 1349+1124 / 1349
class_weight = {0: 1.85, 1: 2.2}
# Compilar el modelo
print("[INFO]: Compilando el modelo...")
model = deep_CNN(width=32, height=32, depth=4, batchNorm=True)
model.compile(loss="binary_crossentropy", optimizer=SGD(learning_rate=0.01), metrics=["accuracy"])
# Entrenar el modelo
print("[INFO]: Entrenando la red...")
H = model.fit(train_datagen, validation_data=val_datagen, epochs=1000, verbose=1, class_weight=class_weight)
# Guardar el modelo
model.save(model_save_path)
# Evaluar el modelo
print("[INFO]: Evaluando el modelo...")
labels = np.array([])
predictions = np.array([])
for x, y in test_dataset:
    preds = model.predict(x, verbose=0)
    predictions = np.concatenate([predictions, preds.ravel()])
    labels = np.concatenate([labels, y.numpy().ravel()])
# Configurar umbral de predicción
predictions = (predictions > 0.5).astype(int)
print(classification_report(labels, predictions, target_names=['No', 'Yes']))
# Graficar curvas de pérdida y precisión de entrenamiento y validación
plt.style.use("ggplot")
# Graficar curva de pérdida
plt.figure()
plt.plot(np.arange(0, len(H.history["loss"])), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, len(H.history["val_loss"])), H.history["val_loss"], label="val_loss")
```

```
plt.title("Training and Validation Loss")
plt.xlabel("Epoch #")
plt.ylabel("Loss")
plt.legend()
plt.show()
# Graficar curva de pérdida, en el rango de 0-1
plt.figure()
plt.plot(np.arange(0, len(H.history["loss"])), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, len(H.history["val_loss"])), H.history["val_loss"], label="val_loss")
plt.title("Training and Validation Loss between 0 and 1")
plt.xlabel("Epoch #")
plt.ylabel("Loss")
plt.legend()
plt.ylim(0, 1)
plt.show()
# Graficar curva de precisión
plt.figure()
plt.plot(np.arange(0, len(H.history["accuracy"])), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, len(H.history["val_accuracy"])), H.history["val_accuracy"], label="val_acc")
plt.title("Training and Validation Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
accuracy = accuracy_score(labels, predictions)
f1 = f1_score(labels, predictions)
mcc = matthews_corrcoef(labels, predictions)
print(f"Overall Accuracy (OA): {accuracy}")
print(f"F1 Score: {f1}")
print(f"Matthews Correlation Coefficient (MCC): {mcc}")
```

11.2.2. Entrenamiento de modelos U-Net

```
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Conv2D, Conv2DTranspose, Input, MaxPooling2D, concatenate, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
from PIL import Image
import matplotlib

matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
os.environ["PYTHONIOENCODING"] = "utf-8"
gpus = tf.config.experimental.list_physical_devices('GPU')
```

```
if gpus:
    try:
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)
        logical_gpus = tf.config.experimental.list_logical_devices('GPU')
        print(len(gpus), "Physical GPUs,", len(logical_gpus), "Logical GPUs")
    except RuntimeError as e:
        print(e)
# Configurar rutas
train_image_dir = r"F:/ <...>/train_png"
train_mask_dir = r"F:/ <...>/mask/train"
test_image_dir = r"F:/ <...>/test_png"
test_mask_dir = r"F: /<...>/mask/test"
model_save_path = r"F:/ <...>/UNet_RGBI.h5"
# Cargar imágenes y máscaras
def load_images_and_masks(image_dir, mask_dir):
    images = []
    masks = []
    label_dir = os.path.join(image_dir, 'yes')
    if not os.path.exists(label_dir):
        raise ValueError(f"Directory {label_dir} does not exist.")
    for filename in os.listdir(label_dir):
        if filename.endswith('.png'):
            image_path = os.path.join(label_dir, filename)
            image = Image.open(image_path).convert('RGBA')
            images.append(np.array(image))
            mask_filename = 'mask_' + filename
            mask_path = os.path.join(mask_dir, mask_filename)
            if os.path.exists(mask_path):
                mask = Image.open(mask_path).convert('L')
                masks.append(np.array(mask))
                print(f"Loading mask for image {filename}")
            else:
                print(f"Mask not found for {mask_path}")
    return np.array(images), np.array(masks)
# Leer datos de entrenamiento y validación
train_images, train_masks = load_images_and_masks(train_image_dir, train_mask_dir)
test_images, test_masks = load_images_and_masks(test_image_dir, test_mask_dir)
# Asegurarse de que los datos se cargaron correctamente
if len(train_images) == 0 or len(train_masks) == 0:
    raise ValueError("No training data found. Please check the data paths and file names.")
# Expandir la dimensión de las máscaras
train_masks = np.expand_dims(train_masks, axis=-1)
test_masks = np.expand_dims(test_masks, axis=-1)
# Normalizar imágenes y máscaras
train_images = train_images / 255.0
test_images = test_images / 255.0
train_masks = train_masks / 255.0
test_masks = test_masks / 255.0
```

```
# Asegurarse de que los valores de las máscaras están en el rango [0, 1]
train_masks[train_masks > 0.5] = 1
train_masks[train_masks <= 0.5] = 0
test_masks[test_masks > 0.5] = 1
test_masks[test_masks <= 0.5] = 0
# Aumento de datos
data_gen_args = dict(rotation_range=40,
                     width_shift_range=0.2,
                     height_shift_range=0.2,
                     shear_range=0.2,
                     zoom_range=0.1,
                     horizontal_flip=True,
                     vertical_flip=True,
                     fill_mode='nearest')
image_datagen = ImageDataGenerator(**data_gen_args)
mask_datagen = ImageDataGenerator(**data_gen_args)
train_images, val_images, train_masks, val_masks = train_test_split(train_images, train_masks, test_size=0.2,
                                                                    random_state=42)
# Crear objeto tf.data.Dataset
def create_dataset(image_generator, mask_generator):
    image_dataset = tf.data.Dataset.from_generator(lambda: image_generator,
                                                   output_signature=tf.TensorSpec(shape=(None, 32, 32, 4),
                                                                                  dtype=tf.float32))
    mask_dataset = tf.data.Dataset.from_generator(lambda: mask_generator,
                                                  output_signature=tf.TensorSpec(shape=(None, 32, 32, 1),
                                                                                 dtype=tf.float32))
    return tf.data.Dataset.zip((image_dataset, mask_dataset))
train_image_generator = image_datagen.flow(train_images, batch_size=32, seed=42)
train_mask_generator = mask_datagen.flow(train_masks, batch_size=32, seed=42)
val_image_generator = image_datagen.flow(val_images, batch_size=32, seed=42)
val_mask_generator = mask_datagen.flow(val_masks, batch_size=32, seed=42)
train_dataset = create_dataset(train_image_generator, train_mask_generator)
val_dataset = create_dataset(val_image_generator, val_mask_generator)
# Definición del modelo UNet
def unet_model(input_size=(32, 32, 4)):
    inputs = Input(input_size)
    c1 = Conv2D(64, (3, 3), activation='relu', padding='same')(inputs)
    c1 = Dropout(0.1)(c1)
    c1 = Conv2D(64, (3, 3), activation='relu', padding='same')(c1)
    p1 = MaxPooling2D((2, 2))(c1)
    c2 = Conv2D(128, (3, 3), activation='relu', padding='same')(p1)
    c2 = Dropout(0.1)(c2)
    c2 = Conv2D(128, (3, 3), activation='relu', padding='same')(c2)
    p2 = MaxPooling2D((2, 2))(c2)

    c3 = Conv2D(256, (3, 3), activation='relu', padding='same')(p2)
    c3 = Dropout(0.2)(c3)
    c3 = Conv2D(256, (3, 3), activation='relu', padding='same')(c3)
    p3 = MaxPooling2D((2, 2))(c3)
```

```
    c4 = Conv2D(512, (3, 3), activation='relu', padding='same')(p3)
    c4 = Dropout(0.2)(c4)
    c4 = Conv2D(512, (3, 3), activation='relu', padding='same')(c4)
    p4 = MaxPooling2D((2, 2))(c4)
    c5 = Conv2D(1024, (3, 3), activation='relu', padding='same')(p4)
    c5 = Dropout(0.3)(c5)
    c5 = Conv2D(1024, (3, 3), activation='relu', padding='same')(c5)
    u6 = Conv2DTranspose(512, (2, 2), strides=(2, 2), padding='same')(c5)
    u6 = concatenate([u6, c4])
    c6 = Conv2D(512, (3, 3), activation='relu', padding='same')(u6)
    c6 = Dropout(0.2)(c6)
    c6 = Conv2D(512, (3, 3), activation='relu', padding='same')(c6)
    u7 = Conv2DTranspose(256, (2, 2), strides=(2, 2), padding='same')(c6)
    u7 = concatenate([u7, c3])
    c7 = Conv2D(256, (3, 3), activation='relu', padding='same')(u7)
    c7 = Dropout(0.2)(c7)
    c7 = Conv2D(256, (3, 3), activation='relu', padding='same')(c7)
    u8 = Conv2DTranspose(128, (2, 2), strides=(2, 2), padding='same')(c7)
    u8 = concatenate([u8, c2])
    c8 = Conv2D(128, (3, 3), activation='relu', padding='same')(u8)
    c8 = Dropout(0.1)(c8)
    c8 = Conv2D(128, (3, 3), activation='relu', padding='same')(c8)
    u9 = Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same')(c8)
    u9 = concatenate([u9, c1])
    c9 = Conv2D(64, (3, 3), activation='relu', padding='same')(u9)
    c9 = Dropout(0.1)(c9)
    c9 = Conv2D(64, (3, 3), activation='relu', padding='same')(c9)
    outputs = Conv2D(1, (1, 1), activation='sigmoid')(c9)
    model = Model(inputs=[inputs], outputs=[outputs])
    return model
# Compilar y entrenar el modelo
model = unet_model(input_size=(32, 32, 4))
model.compile(optimizer=Adam(learning_rate=1e-4), loss='binary_crossentropy', metrics=['accuracy'])
# Entrenar con los datos de entrenamiento
H = model.fit(
    train_dataset,
    validation_data=val_dataset,
    epochs=1000,
    steps_per_epoch=len(train_images) // 32,
    validation_steps=len(val_images) // 32,
    verbose=1
)
# Guardar el modelo
model.save(model_save_path)
# Evaluar el modelo
loss, accuracy = model.evaluate(test_images, test_masks)
# Calcular IoU
def compute_iou(prediction, ground_truth):
    intersection = np.logical_and(prediction, ground_truth).sum()
```

```
    union = np.logical_or(prediction, ground_truth).sum()
    if union == 0:
        return 1.0 if intersection == 0 else 0.0
    return intersection / union
def evaluate_iou(model, test_images, test_masks):
    iou_scores = []
    for img, true_mask in zip(test_images, test_masks):
        pred_mask = model.predict(img[np.newaxis, ...])  # Agregar dimensión de lote
        pred_mask = (pred_mask > 0.5).astype(np.uint8)  # Binarizar
        iou = compute_iou(pred_mask.squeeze(), true_mask.squeeze())
        iou_scores.append(iou)
    return np.mean(iou_scores)
mean_iou = evaluate_iou(model, test_images, test_masks)
print(f"Test Loss: {loss}")
print(f"Test Accuracy: {accuracy}")
print(f"Mean IoU: {mean_iou}")
# Dibujar las curvas de pérdida y precisión de entrenamiento y validación
plt.style.use("ggplot")
# Dibujar curva de pérdida
plt.figure()
plt.plot(np.arange(0, len(H.history["loss"])), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, len(H.history["val_loss"])), H.history["val_loss"], label="val_loss")
plt.title("Training and Validation Loss")
plt.xlabel("Epoch #")
plt.ylabel("Loss")
plt.legend()
plt.show()
# Dibujar curva de pérdida en el rango 0-0.5
plt.figure()
plt.plot(np.arange(0, len(H.history["loss"])), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, len(H.history["val_loss"])), H.history["val_loss"], label="val_loss")
plt.title("Training and Validation Loss between 0 and 0.5")
plt.xlabel("Epoch #")
plt.ylabel("Loss")
plt.legend()
plt.ylim(0, 0.5)
plt.show()
# Dibujar curva de precisión
plt.figure()
plt.plot(np.arange(0, len(H.history["accuracy"])), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, len(H.history["val_accuracy"])), H.history["val_accuracy"], label="val_acc")
plt.title("Training and Validation Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```

11.3 Aplicación de modelo

11.3.1. Aplicar modelo y generar Shapefile

```
import os
import csv
import numpy as np
from osgeo import gdal
import rasterio
from rasterio.features import shapes
import tensorflow as tf
from keras.models import load_model
from PIL import Image
import fiona
from shapely.geometry import shape, mapping
from pyproj import CRS
import gc
from tqdm import tqdm
# Cargar el modelo
cnn_model_path = r"F:\ <...>\CNN_ultra_RGBI.h5"
unet_model_path = r"F:\ <...>\UNet_ultra_mix_RGBI.h5"
cnn_model = load_model(cnn_model_path)
unet_model = load_model(unet_model_path)
# Configurar rutas
input_dir = "F:\TFM\detect_RGBI\image_s_2022"
output_png_dir = "F:\TFM\detect_RGBI\image_s_2022_png"
mask_dir = "F:\TFM\detect_RGBI\mask_2022"
shapefile_dir = "F:\TFM\detect_RGBI\shapefiles_2022"
csv_output_path = "F:\TFM\detect_RGBI\predictions_2022_RGBI.csv"
# Asegurar que los directorios de salida existan
os.makedirs(output_png_dir, exist_ok=True)
os.makedirs(mask_dir, exist_ok=True)
os.makedirs(shapefile_dir, exist_ok=True)
# Abrir el archivo CSV y preparar para escribir
csv_file = open(csv_output_path, mode='w', newline='')
csv_writer = csv.writer(csv_file)
csv_writer.writerow(['filename', 'cnn_prediction', 'solar_panel_detected', 'mask_generated'])  # Escribir la cabecera
def convert_to_png(filepath, output_dir):
    with rasterio.open(filepath) as src:
        image = src.read([1, 2, 3, 4])  # Leer cuatro bandas RGBA
        image = np.moveaxis(image, 0, -1)  # Convertir a (H, W, C)
        image[:, :, :3] = (image[:, :, :3] / image[:, :, :3].max() * 255).astype(np.uint8)  # Normalizar los tres canales RGB y convertir a 8
bits
        image[:, :, 3] = (image[:, :, 3] / image[:, :, 3].max() * 255).astype(np.uint8)  # Normalizar el canal Alpha y convertir a 8 bits
        filename = os.path.basename(filepath).replace('.tif', '.png')
        png_path = os.path.join(output_dir, filename)
        Image.fromarray(image, 'RGBA').save(png_path)  # Guardar como PNG en formato RGBA
```

```
        return png_path
def process_image(filepath):
    image = Image.open(filepath).convert('RGBA')
    image = np.array(image) / 255.0    # Normalizar
    return image
def save_georeferenced_mask(mask, original_tif_path, mask_filepath):
    mask = np.squeeze(mask)    # Eliminar dimensiones adicionales
    with rasterio.open(original_tif_path) as src:
        meta = src.meta.copy()
        meta.update({
            'driver': 'GTiff',
            'dtype': 'uint8',
            'count': 1,
            'compress': 'lzw'
        })
        with rasterio.open(mask_filepath, 'w', **meta) as dst:
            dst.write(mask, 1)
    return mask_filepath
def clear_memory():
    gc.collect()
    tf.keras.backend.clear_session()
# Obtener el total de archivos
file_list = [f for f in os.listdir(input_dir) if f.endswith(".tif")]
total_files = len(file_list)
# Iterar sobre todas las subimágenes
counter = 0
clear_every = 1000    # Limpiar memoria y VRAM cada 1000 imágenes procesadas
clear_memory()
# Añadir barra de progreso
with tqdm(total=total_files, desc="Processing files") as pbar:
    for filename in file_list:
        filepath = os.path.join(input_dir, filename)
        # Convertir a formato PNG
        png_filepath = convert_to_png(filepath, output_png_dir)
        # Leer la imagen PNG
        image = process_image(png_filepath)
        # Reiniciar variables
        solar_panel_detected = "no"
        mask_generated = None
        # Predecir si existe un panel solar
        image_batch = np.expand_dims(image, axis=0)
        prediction = cnn_model.predict(image_batch)
        prediction_value = prediction[0][0]
        if prediction_value > 0.1:
            solar_panel_detected = "yes"
            # Existe un panel solar, usar el modelo UNet para generar la máscara
            mask = unet_model.predict(image_batch)[0]
            mask = (mask > 0.5).astype(np.uint8) * 255    # Convertir a imagen binaria
            mask_generated = "no"
```

```
            if np.any(mask):    # Verificar si hay píxeles positivos en la máscara
                mask_generated = "yes"
                mask_filename = f"mask_{os.path.basename(filepath)}"
                mask_filepath = os.path.join(mask_dir, mask_filename)
                geo_mask_filepath = save_georeferenced_mask(mask, filepath, mask_filepath.replace('.png', '.tif'))
                # Obtener información geográfica y convertir la máscara a shapefile
                with rasterio.open(filepath) as src:
                    transform = src.transform
                    crs = src.crs.to_dict()
                with rasterio.open(geo_mask_filepath) as mask_src:
                    mask_data = mask_src.read(1)
                    results = shapes(mask_data, mask=(mask_data > 0), transform=transform)
                    shapefile_path = os.path.join(shapefile_dir,
                                                  f"shapefile_{os.path.basename(png_filepath).replace('.png', '.shp')}")
                    schema = {
                        'geometry': 'Polygon',
                        'properties': {'raster_val': 'int'},
                    }
                    with fiona.open(shapefile_path, 'w', 'ESRI Shapefile', schema=schema, crs=crs) as shp:
                        for result in results:
                            geom, value = result
                            if value == 255:
                                shp.write({
                                    'properties': {'raster_val': int(value)},
                                    'geometry': geom
                                })
            # Imprimir el nombre del archivo y el valor de predicción para observar el progreso
            print(
                f"File: {filename}, CNN_prediction: {prediction_value},"
                f"PS_detected: {solar_panel_detected},Mask_genera: {mask_generated}")
            # Escribir resultados en el archivo CSV
            csv_writer.writerow([filename, prediction_value, solar_panel_detected, mask_generated])
            # Incrementar contador y limpiar memoria y VRAM
            counter += 1
            if counter % clear_every == 0:
                clear_memory()
            # Actualizar la barra de progreso
            pbar.update(1)
    # Cerrar archivo
    csv_file.close()
    print("Processing complete.")
```

11.3.2. Combinar shapefile

```
import os
import fiona
from shapely.geometry import shape, mapping, MultiPolygon, Polygon
from shapely.ops import unary_union
from fiona.crs import from_epsg

# Configurar rutas
shapefile_dir = "F:/ <...>/shapefiles_2022/"
final_shapefile_path = "F:/ <...>/final_mix_RGBI_2022.shp"
# Leer todos los shapefiles
shapefiles = [os.path.join(shapefile_dir, f) for f in os.listdir(shapefile_dir) if f.endswith('.shp')]
# Combinar todos los shapefiles en un MultiPolygon
all_polygons = []
for shp_path in shapefiles:
    with fiona.open(shp_path, 'r') as shp:
        crs = shp.crs  # Obtener el sistema de referencia de coordenadas
        for item in shp:
            geom = shape(item['geometry'])
            all_polygons.append(geom)
# Usar unary_union para combinar todos los Polygon y MultiPolygon
merged_polygons = unary_union(all_polygons)
# Crear archivo de salida
schema = {
    'geometry': 'Polygon',
    'properties': {'id': 'int'},
}
# Escribir el shapefile final
with fiona.open(final_shapefile_path, 'w', driver='ESRI Shapefile', schema=schema, crs=crs) as output:
    if isinstance(merged_polygons, Polygon):
        output.write({'geometry': mapping(merged_polygons), 'properties': {'id': 1}})
    elif isinstance(merged_polygons, MultiPolygon):
        for i, polygon in enumerate(merged_polygons.geoms):
            output.write({'geometry': mapping(polygon), 'properties': {'id': i + 1}})
print("Merging complete. Final shapefile created at:", final_shapefile_path)
```

11.4 Análisis de resultados

11.4.1. Calcular nuevos paneles solares vía PyQGIS

```
from qgis.core import (
    QgsVectorLayer,
    QgsFeature,
    QgsGeometry,
    QgsProject,
    QgsSpatialIndex,
    QgsField,
    QgsFields,
    QgsWkbTypes,
    QgsVectorFileWriter
)
from qgis.PyQt.QtCore import QVariant
# Reemplazar con la ruta de tu shapefile
path_2022 = 'E:/ <...>/PS_2022_laEliana.shp'
path_2023 = 'E:/ <...>/PS_2023_laEliana.shp'
new_layer_path = 'E:/<...>/new_PS_2023.shp'
# Cargar shapefile
layer_2022 = QgsVectorLayer(path_2022, '2022', 'ogr')
layer_2023 = QgsVectorLayer(path_2023, '2023', 'ogr')
# Verificar si las capas son válidas
if not layer_2022.isValid() or not layer_2023.isValid():
    print("¡Error al cargar las capas!")
else:
    # Añadir al proyecto de QGIS
    QgsProject.instance().addMapLayer(layer_2022)
    QgsProject.instance().addMapLayer(layer_2023)
    # Crear índice espacial para acelerar la búsqueda
    index_2022 = QgsSpatialIndex(layer_2022.getFeatures())
    # Crear una nueva capa para almacenar los nuevos polígonos añadidos
    fields = QgsFields()
    fields.append(QgsField("id", QVariant.Int))
    new_layer = QgsVectorLayer('Polygon?crs=' + layer_2023.crs().authid(), 'New_Solar_Panels_2023', 'memory')
    new_layer.dataProvider().addAttributes(fields)
    new_layer.updateFields()
    new_layer_dp = new_layer.dataProvider()
    # Preparar variables de conteo
    unchanged_panels = 0
    new_panels = 0
    new_panel_features = []
    # Recorrer cada polígono del 2023
    for feature_2023 in layer_2023.getFeatures():
        geom_2023 = feature_2023.geometry()
        intersects_2022 = index_2022.intersects(geom_2023.boundingBox())
```

```
        overlap_found = False
        # Verificar si hay polígonos superpuestos
        for fid_2022 in intersects_2022:
            feature_2022 = layer_2022.getFeature(fid_2022)
            geom_2022 = feature_2022.geometry()
            # Calcular el área de intersección
            if geom_2023.intersects(geom_2022):
                intersection = geom_2023.intersection(geom_2022)
                if intersection.area() / geom_2023.area() >= 0.4:
                    overlap_found = True
                    break
        if overlap_found:
            unchanged_panels += 1
        else:
            new_panels += 1
            new_feat = QgsFeature(new_layer.fields())
            new_feat.setGeometry(geom_2023)
            new_feat.setAttribute("id", new_panels)
            new_panel_features.append(new_feat)
# Añadir los nuevos polígonos a la nueva capa
new_layer_dp.addFeatures(new_panel_features)
# Añadir la nueva capa al proyecto
QgsProject.instance().addMapLayer(new_layer)
# Guardar la nueva capa como un shapefile
QgsVectorFileWriter.writeAsVectorFormat(new_layer, new_layer_path, "UTF-8", new_layer.crs(), "ESRI Shapefile")
# Imprimir resultados
print(f"Paneles solares sin cambios en 2023: {unchanged_panels}")
print(f"Nuevos paneles solares en 2023: {new_panels}")
```

12 ANEXO II

ANEXO II - RELACIÓN DEL TRABAJO CON LOS OBJETIVOS DE DESARROLLO SOSTENIBLE DE LA AGENDA 2030

Anexo al Trabajo de Fin de Grado y Trabajo de Fin de Máster: Relación del trabajo con los Objetivos de Desarrollo Sostenible de la agenda 2030

Objetivos de Desarrollo Sostenibles		Alto	Medio	Bajo	No Procede
ODS 1.	**Fin de la pobreza.**				X
ODS 2.	**Hambre cero.**				X
ODS 3.	**Salud y bienestar.**				X
ODS 4.	**Educación de calidad.**				X
ODS 5.	**Igualdad de género.**				X
ODS 6.	**Agua limpia y saneamiento.**				X
ODS 7.	**Energía asequible y no contaminante.**	X			
ODS 8.	**Trabajo decente y crecimiento económico.**				X
ODS 9.	**Industria, innovación e infraestructuras.**		X		
ODS 10.	**Reducción de las desigualdades.**				X
ODS 11.	**Ciudades y comunidades sostenibles.**	X			
ODS 12.	**Producción y consumo responsables.**			X	
ODS 13.	**Acción por el clima.**		X		
ODS 14.	**Vida submarina.**				X
ODS 15.	**Vida de ecosistemas terrestres.**				X
ODS 16.	**Paz, justicia e instituciones sólidas.**				X
ODS 17.	**Alianzas para lograr objetivos.**				X