Detector de nubes en imágenes satélite con segmentación semántica

Susana Sánchez Ropero

1 de julio de 2024

Resumen– En este trabajo se examina el funcionamiento de un modelo basado en la arquitectura U-Net para la detección de nubes en imágenes satelitales RGB. El objetivo es identificar los cinco tipos diferentes de nubes presentes en el conjunto de datos seleccionado. Se implementará el modelo y se comparará su desempeño utilizando distintos parámetros y configuraciones. Estas comparaciones permitirán determinar cuál es la mejor configuración de parámetros para optimizar la detección de nubes. Además, se presentarán los resultados obtenidos y se discutirá la eficacia del modelo en la clasificación y segmentación de nubes en las imágenes satelitales. Por último, se propondrán posibles mejoras para futuros trabajos en esta área, que consideramos importantes destacar. También se abordará la implementación de técnicas de preprocesamiento de datos y se analizará su impacto en el rendimiento del modelo.

Palabras clave– Detector nubes, segmentación semántica, deep learning, imágenes satélite, redes neuronales, Python.

Abstract– In this work, the functioning of a model based on the U-Net architecture for cloud detection in RGB satellite images is examined. The objective is to identify the five different types of clouds present in the selected dataset. The model will be implemented and its performance will be compared using different parameters and configurations. These comparisons will allow us to determine the best parameter configuration to optimize cloud detection. Additionally, the obtained results will be presented and the model's effectiveness in cloud classification and segmentation in satellite images will be discussed. Finally, possible improvements for future work in this area, which we believe are important to highlight, will be proposed. The implementation of data preprocessing techniques and their impact on the model's performance will also be addressed. We aim to contribute significantly to the field of cloud detection.

Keywords– Cloud detector, semantic segmentation, deep learning, satellite images, neural networks, Python.

1 Introducción

En los últimos años, el lanzamiento de misiones espaciales ha incrementado el interés por desarrollar muchas aplicaciones de observación de la Tierra como la identificación de los usos del terreno, la detección de problemas causados por la contaminación o la evaluación de daños causados por desastres naturales.

Uno de los problemas principales que enfrentan estas aplicaciones, es la presencia de nubes en las imágenes, las cuales impiden la visualización de la superficie de la Tierra y dificultan la obtención de información relevante. Las nubes pueden limitar la cantidad de datos útiles que pueden extraerse de las imágenes. Por lo tanto, pueden entorpecer expediciones, operaciones o misiones agrícolas, militares, geológicas, etc. Estas pueden sufrir percances importantes si los equipos no van lo suficientemente preparados por no poseer toda la información del terreno y el entorno.

De aquí, surge la importancia de desarrollar herramientas para detectar y eliminar las nubes de las imágenes satelitales. Una ventaja del detector de nubes que se puede aplicar al envío de datos a la Tierra desde los satélites que las captan puede ser establecer un filtro. Es decir, si la imagen supera un umbral de píxeles tapados por

• E-mail de contacto: 1494978@uab.cat
• Mención realizada: Computación
• Trabajo tutorizado por: Robert Benavente Vidal (Departamento de Ciencias de la Computación)
• Curso 2023/24

nubes, descartar esta y no hacer el envío. Ya que este proceso de envío de imágenes desde los nanosatélites a las instalaciones donde se almacenan es muy costoso. Además, esos nanosatélites tienen hardware limitado, es decir, no pueden almacenar infinitas imágenes. Por esta razón, si filtramos con un umbral establecido previamente, haremos que este proceso sea óptimo y no se almacenen datos que en un futuro no puedan usarse o no sean de gran ayuda.

A lo largo del trabajo, se mencionarán los diferentes tipos de imágenes extraídas del satélite Sentinel-2 del conjunto de datos escogido CloudSEN12[4]. En estas, se identifican estos cinco tipos de imágenes:

1. Sin nubes
2. Poco nublado
3. Casi claro
4. Medio nublado
5. Nublado

Según esta clasificación podremos establecer un filtro de descarte de imágenes como hemos mencionado anteriormente.

2 Estado del arte

En la actualidad, el uso de imágenes captadas por satélites es un recurso común para observar la Tierra. Estas imágenes se utilizan para fines de monitorización de cambios en el medio ambiente, meteorología, navegación, estudio del terreno, planificación urbana, entre otros. Sin embargo, en todos estos ámbitos hay un problema común a la hora de obtener las imágenes: que estén parcial o totalmente tapadas por nubes.

Para solucionar este problema, se han implementado una serie de modelos basados en aprendizaje profundo, como las redes neuronales convolucionales, que clasifican las imágenes según tengan o no nubes.

El método más usado para resolver este problema es a través de imágenes multiespectrales que capturan la longitud de la onda de luz [1]. Pero usando esto último, las imágenes que contienen espectros semejantes, como por ejemplo, la nieve, el hielo, estructuras artificiales reflectantes, y arena en desiertos, provocaban una incorrecta clasificación de dichas imagenes.

Eligiendo bien el modelo, ya sea aplicando técnicas que utilizan umbrales y aprendizaje profundo, máquinas de vectores de soporte (SVM), redes neuronales convolucionales (CNN) o métodos basados en ensambles [2], y validando y ajustando estos a medida que se van viendo los resultados de las clasificaciones se conseguirá una mejor clasificación.

Por lo tanto, una vez se obtiene el modelo de clasificación que mejor resultado da según las necesidades establecidas, es hora de poner solución e intentar eliminar la nube de la imagen satélite. Esto es todo un reto actualmente. Existen herramientas como GEOBIA y Sen2Cor [3] para realizar las correcciones atmosféricas y así poder eliminar las nubes. Pero solo pueden eliminar las grandes tormentas y conjuntos grandes de nubes. Otra herramienta con esta finalidad es el Cloud Masking [3]. Esto genera una máscara de nubes para Landsat y se eliminarán desde QGIS [20]. Desgraciadamente, no se rellenarán todos sus píxeles.

La forma que mejores resultados obtiene es la fusión de varias imágenes [3]. Se obtienen imágenes justo de la misma escena, tiempo antes o después en la que no aparezcan nubes o solo en algunas partes. De manera que, se sustituirán los píxeles ocupados por nube por los de otra imagen que no tenga. En la plataforma Google Earth Engine [3] se podrán construir las imágenes satélites sin nubes.

3 Objetivos

Tras analizar el estado del arte para resolver el problema de detectar las nubes en las imágenes, junto con las posibles soluciones para eliminar lo máximo posible el rastro de nubes de ellas, procedo a establecer los objetivos a alcanzar en este trabajo:

1. Estudiar las diferentes opciones de conjuntos de datos útiles para resolver este problema. Y obtener uno adecuado formado por imágenes satélite donde aparezcan nubes para entrenar el modelo y poder clasificarlas.
2. Realizar un modelo de detección de nubes con imágenes satélite en lenguaje Python. Una vez acabado el punto anterior, con los conocimientos adquiridos se procederá a crear un modelo basado en segmentación semántica (U-Net) capaz de identificar y delimitar las partes de las imágenes con nubes de manera eficiente y precisa.
3. Comparar el modelo anterior con diferentes parámetros para comprobar cuál es el más adecuado y genera mejores resultados de clasificación. Usaré las siguientes medidas de evaluación del modelo: la Curva ROC y AUC, intersection-over-union, accuracy y la matriz de confusión para hacer la comparación entre las diferentes versiones del modelo. También mostraremos y compararemos las máscaras obtenidas.

Una vez acabados los tres puntos anteriores del trabajo y se dispone de tiempo suficiente, se tratarán los siguientes puntos:

4. Diseñar e implementar una técnica de eliminación de nubes en las imágenes.
5. Comparar los resultados con otras técnicas de eliminación de nubes para ver cuál da mejores resultados.

4 Metodología

Para solucionar el problema de detección de nubes en imagenes satélite, vamos a crear un modelo U-Net.

4.1. Conjunto de datos CloudSEN12

Como se ha mencionado anteriormente, el dataset escogido es el proporcionado por CloudSEN12 [4]. Estos datos son recogidos del satélite Sentinel-2 de la Agencia Espacial Europea. Lo que captura son imágenes en diferentes bandas espectrales, lo que nos permite poder hacer detectores de nubes con ellas de manera precisa y eficaz.

CloudSEN12 está dividido en subcarpetas, en las cuales encontramos imágenes TIFF como las de las siguientes ilustraciones. Estas tres imágenes corresponden a la carpeta ROI_0001:

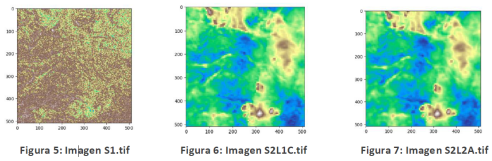


Fig. 1: Imágenes TIFF

Estas imágenes son extraidas del satélite Sentinel-2 de todos los continentes excepto la Antártida. En la Figura 2 se puede observar la organización en subcarpetas del conjunto de datos CloudSEN12. Este se organiza en subcarpetas llamadas ROI_XXX. Dentro de las cuales se encuentran las siguientes subcarpetas y archivos:

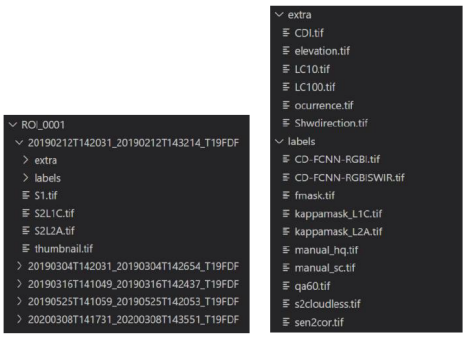


Fig. 2: Organización subcarpetras de CloudSEN12

Como podemos ver en la imagen de la derecha de la figura 3, tenemos, dentro de la carpeta labels, el archivo llamado fmask.tif. Este archivo es la máscara correspondienter a la imagen, donde cada píxel está clasificado según el tipo de nube que aparece en él. Esta máscara tiene la siguiente forma:

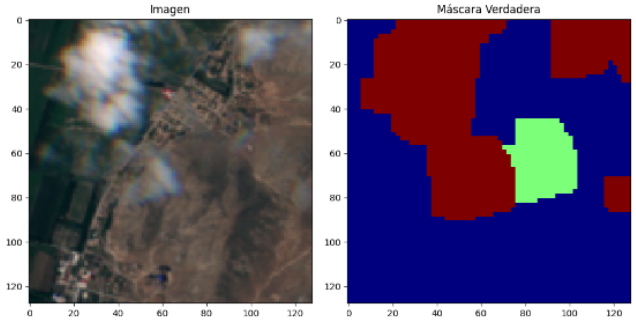


Fig. 3: Imagen vs Máscara

El conjunto de datos está dividido por imágenes y sus máscaras correspondientes. Estas máscaras estan compuestas por píxeles clasificados del 0 al 4 según la clase de nube que se encuentre en este. Como hemos mencionado en apartados anteriores, estos son los tipos de nubes que clasifica:

1. Sin nubes (0)
2. Poco nublado (1)
3. Casi claro (2)
4. Medio nublado (3)
5. Nublado (4)

Estas cinco clases serán representadas por los colores indicados en la figura 4 en las máscaras:

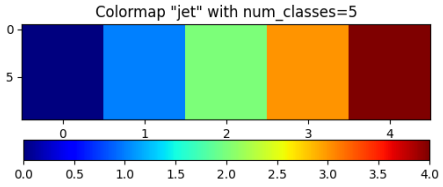


Fig. 4: Leyenda colores correspondientes a cada clase de las máscaras

4.2. Preparación del conjunto de datos

Las imágenes del conjunto de datos se componen de varias bandas. En este trabajo vamos a usar las bandas 2, 3 y 4 correspodientes a los colores azul, verde y rojo respectivamente [5]. Al superponerlas, formamos una imagen RGB, como podemos ver en la figura 5:

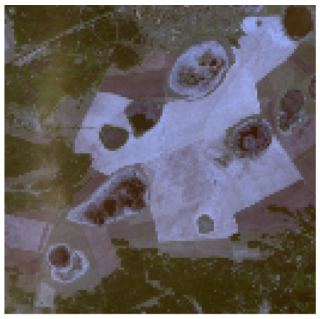


Fig. 5: Imagen RGB

Se va a usar el modelo U-Net de la biblioteca Keras [6]. Este modelo está basado en arquitecturas de red neuronal convolucional (CNN). Es útil en la detección de nubes en imágenes RGB ya que trabaja con segmentación semántica [6], lo que clasifica cada píxel de la imagen en una clase.

Como las imágenes originales son de 509x509 píxeles, debemos recortar un recuadro de la imagen de 128x128 para poder pasarle el conjunto de datos al modelo. De esta forma, las imágenes tendrán el tamaño de (128, 128, 3) píxeles y sus máscaras respectivas de (128, 128). Normalizaremos las imágenes para obtener valores entre 0 y 1.

4.3. Definición del modelo U-Net

Ya tenemos el conjunto de datos preprocesado y almacenado en un dataset. Procedemos a definir el modelo.

El modelo se compone de las siguientes capas:

1 (entrada) + 27 (encoder) + 16 (decoder) + 1 (salida) = 45 capas

Estos pueden dividirse en 2 bloques principales, encoder y decoder, y 3 bloques adicionales en los cuales se definen el tipo de input y output deseado y su normalización.

1. Entrada: La entrada del modelo acepta imágenes RGB con 3 canales del tamaño que se le pase. En este caso (128, 128, 3).
2. Primer bloque:
   a) Conv2D: Convoluciona la entrada en una única dimensión espacial para producir un tensor de salidas [7].
   b) BatchNormalization: Realiza la normalización para acelerar el entrenamiento y mejorar la estabilidad del modelo [8].
   c) Activation relu: Aplica la función de activación ReLU para introducir no linealidad [9].
3. Bloque Encoder:
   a) SeparableConv2D: Aplica una convolución separable en profundidad, una técnica eficiente para reducir la cantidad de parámetros y el costo computacional en una red neuronal convolucional [10].
   b) BatchNormalization: Realiza la normalización por lotes.
   c) MaxPooling2D: Reduce la resolución espacial mientras mantiene las características más importantes para controlar el sobreajuste [11].
   d) Add: Implementa conexiones residuales y atajos, sumando los tensores procesados con sus correspondientes tensores residuales para facilitar el flujo de información y gradientes [12].
4. Bloque Decoder:
   a) Activation relu: Aplica la función de activación ReLU.
   b) Conv2DTranspose: Aplica una convolución transpuesta para aumentar la resolución espacial [13].
   c) BatchNormalization: Realiza la normalización por lotes.
   d) UpSampling2D: Duplica la resolución espacial [14].
   e) Add: Implementa conexiones residuales y atajos, sumando los tensores procesados con sus correspondientes tensores residuales para facilitar el flujo de información y gradientes.
5. Output:
   a) Capa de Salida:
      1) Conv2D: Aplica una convolución final con un número de filtros igual al número de clases de salida (en este caso 5).
      2) Activation softmax: Aplica la función de activación softmax para producir las probabilidades de pertenencia a cada clase por píxel [15].
      3) La matriz de salida es de (128, 128, 5).

Una vez definido el modelo U-Net, procedemos a dividir el dataset en un conjunto de entrenamiento (940 elementos) y otro de validación (160 elementos). De esta forma, podremos compilarlo.

Para realizar la compilación, usaremos inicialmente:

1. Optimizador Adam [16].
2. Función de pérdida "sparse_categorical_crossentropy", ya que tenemos más de dos clases con etiquetas one-hot [17].
3. Las métricas Accuracy y Intersection Over Union [18].

Una vez compilado, entrenaremos el modelo utilizando ModelCheckPoint para asegurarnos de escoger el mejor modelo posible, basado en la mejor métrica de validación [19].

5 Resultados

Hecho el entrenamiento, obtenemos los resultados de la figura 6 y 7 utilizando los parámetros siguientes:

- batch_size: 32
- Número de épocas: 50
- Training samples: 940
- Validation samples: 160
- Shape de imágenes: (940, 128, 128, 3)
- Shape de máscaras: (940, 128, 128)

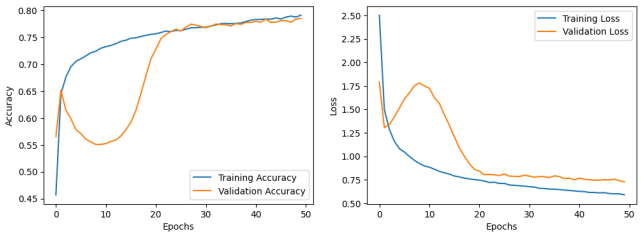


Fig. 6: Accuracy y Loss

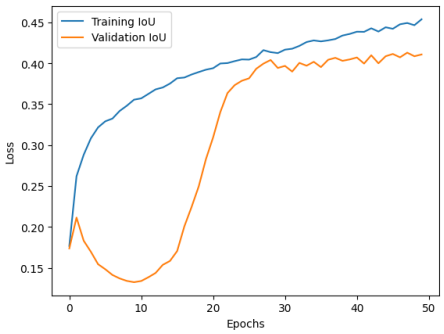
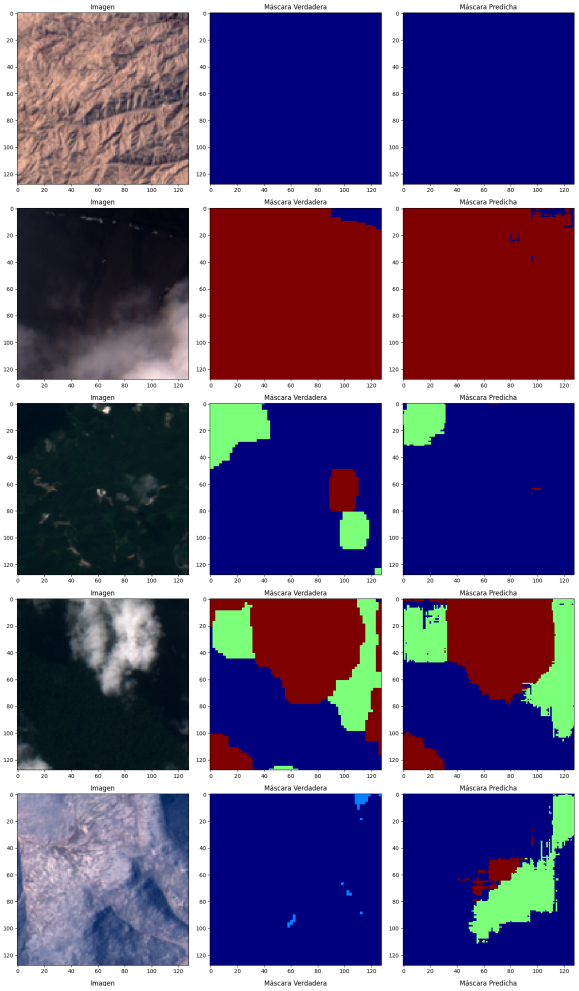


Fig. 7: Intersection Over Union

Como podemos observar, el entrenamiento ha sido exitoso, ya que la pérdida (loss) disminuye a medida que el modelo se entrena. Asimismo, la precisión (accuracy) y la métrica de intersección sobre la unión (intersection over union) aumentan progresivamente. Esto quiere decir que el modelo está aprendiendo correctamente. Además, vemos que no se está produciendo overfiting y vemos que todavía podría aprender si aumentáramos el número de épocas en la ejecución del entrenamiento.

Si mostramos la máscara verdadera y la comparamos con la máscara predicha por el modelo, podremos confirmar lo analizado anteriormente en las gráficas. En la figura 8, observamos la eficacia de la predicción.



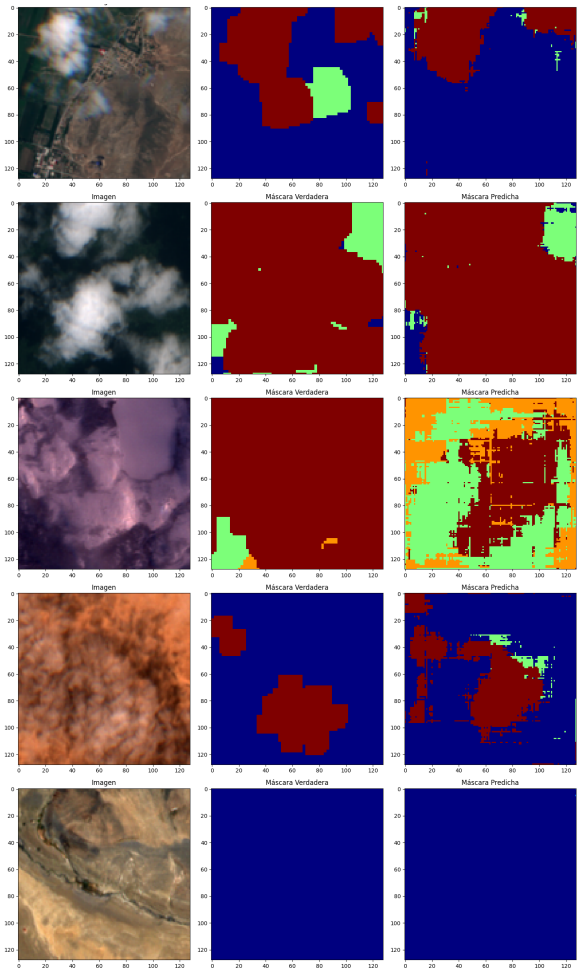


Fig. 8: Máscara verdadera Vs Máscara predicha

Si analizamos la figura anterior, vemos que la clase azul correspondiente a la clase 0 - Sin nubes, tiene un porcentaje muy elevado de predicción correcta. Las máscaras originales y las predichas coinciden prácticamente al 100 %. La misma casuística encontramos con la clase roja correspondiente a la clase 4 - nublado.

Si ahora analizamos la clase verde correspondiente a la clase 2 - casi nublado, vemos que presenta un porcentaje más elevado de confusión en comparación con las dos anteriores. Si observamos la imagen RGB, a simple vista no se percibe ningún tipo de nube. Esto está confundiendo al modelo, ya que clasifica como clase 2 los huecos entre nubes. Es muy difícil para el modelo identificar esta clase correctamente. Es esperable que esto ocurra, ya que ni siquiera a simple vista se puede diferenciar.

Vemos que la presencia de las clases naranja y azul claro es mucho menos frecuente. Estas corresponden a medio nublado y poco nublado respectivamente. Y si analizamos las imágenes RGB donde hay presencia de ellas, vemos que a simple vista, ocurre la misma casuística que con la clase verde.

Podemos concluir después de este estudio, que las clase 1, 2 y 3 se conjunden con el fondo de la imágen RGB y el modelo no logra diferenciarlas correctamente. La clase casi nublado (3) si es identificada de forma más correcta en comparación a la poco nublado (1) y medio nublado (3).

Esto puede deberse a que las clases 1 y 3 tienen mucha menos cantidad de datos de las clases en el dataset. Las clases Sin nubes (0) y nublado (4) tienen una clasificación correcta, ya que es mucho más fácil diferenciarlas.

Ya vistas y analizadas las máscaras predichas, vamos a estudiar la matriz de confusión de la figura 9, donde veremos realmente como de bien clasifica el modelo cada clase:

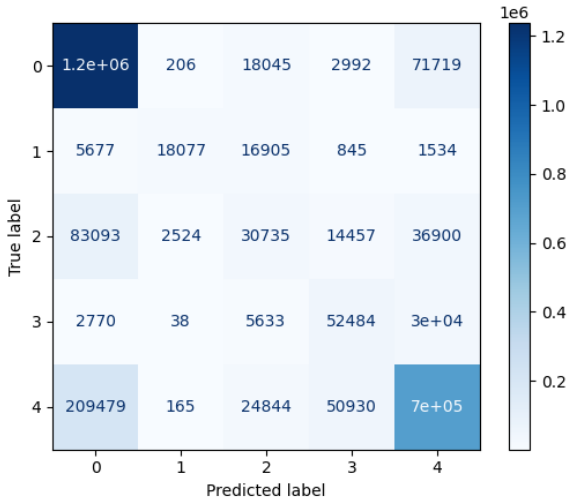


Fig. 9: Matriz de confusión

La concusión obtenida anteriormente se confirma observando los resultados de la matriz de confusión:

- Las clases 1, 2 y 3 (poco nublado, casi nublado y medio nublado) se confunden con el fondo de la imagen RGB, lo que dificulta su correcta identificación por parte del modelo.
- La clase 3 se identifica con mayor precisión en comparación con las clases 1 y 2, probablemente debido a la mayor cantidad de datos disponibles para esta clase en el dataset.
- Las clases 0 (sin nubes) y 4 (nublado) tienen una clasificación correcta porque son más fáciles de diferenciar.

Por último, observemos las curvas ROC de la figura 10:

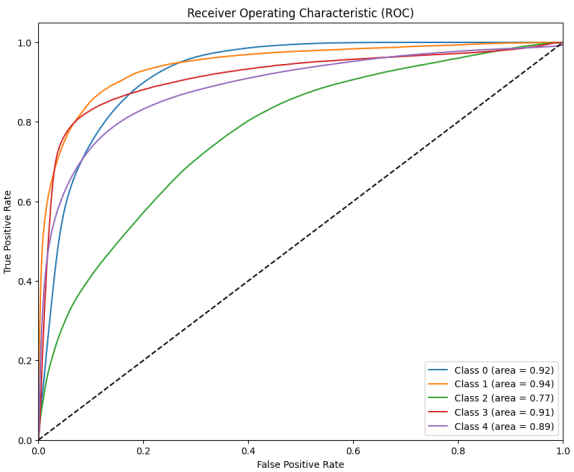


Fig. 10: Gráfica curva ROC

Obervamos que el modelo de clasificación tiene un buen rendimiento en general, especialmente para las clases 0, 1, 3, y 4, con AUCs bastante altos. La clase 2 es la que presenta más confusión, pero aún así, está por encima del 0.5 que sería lo aleatorio.

Se han probado técnicas de balanceo de datos pero, o bien ofrecen resultados muy parecidos a los mencionados. O, en otros casos, se dispara el valor de pédida del entrenamiento.

Las técnicas que se han usado para tratar de resolver esta casuística son las siguentes:

1. Data aumentation: esta técnica está actualmente aplicada en el entrenamiento. Los resultados obtenidos han estado generados aplicándola.

2. Sobre-muestreo de las clases minoritarias: esta técnica daba resultados muy bajos con una pérdida extremadamente alta.

3. Añadir más datos al dataset: se han añadido más imágenes al conjunto de datos. Esta solución aumenta ligeramente la buena clasificación en el entrenamiento, pero no es significativo ya que, en el conjnto de datos CloudSEN12 tampoco se encuentra variedad de las clases menos prioritarias.

Consideramos después de analizar los resultados, que el conjunto de datos en vez de dividirse en 5 clases, podría reajustarse a 3, de esta forma se obtendrán mejores resultados y tendremos un modelo mucho más robusto. Consideramos que la clase 1 y la clase 3 no son lo suficientemente consistentes como para considerarse clases independientes a las clases 0, 2 y 4.

Dicho esto, podriamos reajustar las clases del modelo de la siguiente forma:

- Clase 0 - Sin nubes
- Clase 1 - Poco nublado / Casi nublado / Medio nublado
- Clase 2 - Nublado

5.1. Entrenamiento con 3 clases

Se han unificado las clases 1, 2 y 3 en una sola dejando así tres clases.

Vamos a evaluar los resultados:

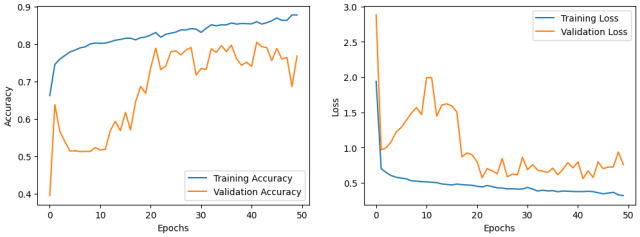


Fig. 11: Accuracy y Loss

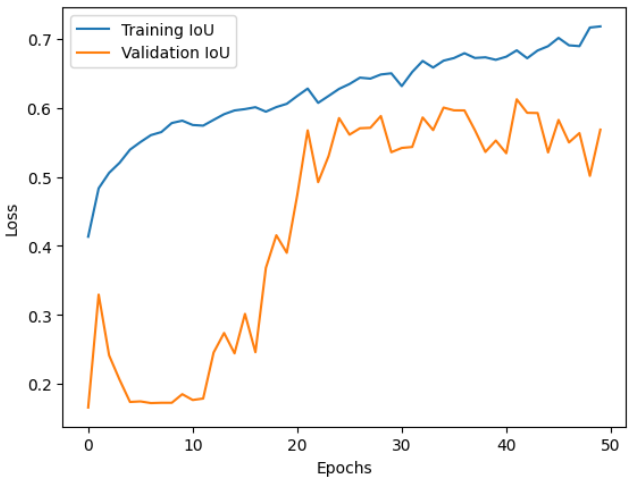


Fig. 12: Intersection Over Union

Si comparamos estos resultados con los del entrenamiento realizado con 5 clases, podemos ver una mejoría.

Observemos la figura 13 para comparar la máscara predicha con la verdadera.

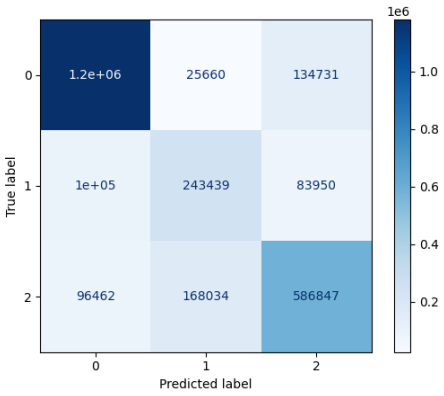


Fig. 13: Máscara verdadera vs Máscara predicha

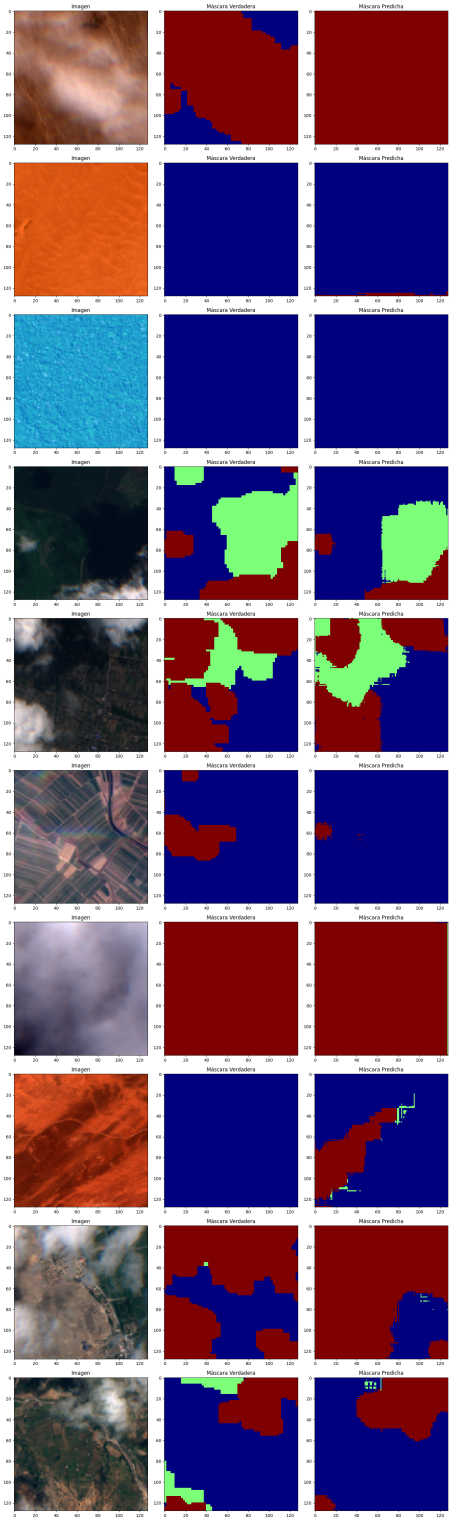


Fig. 14: Matriz de confusión

Podemos observar que la clase verde ahora es mucho más precisa, aunque todavía existe confusión.

Para ver si realmente ha habido mejoría con este cambio en el conjunto de datos, observemos la matriz de confusión de la figura 14.

Existe una mejoría notoria en la clase medio nublado (1), aunque se confirma que permanece la confusión. Esto es debido a que se confunde con el fondo.

En la figura 15 podemos ver que las clases tienen un área bastante elevada. Implica que el entrenamiento ha sido exitoso.

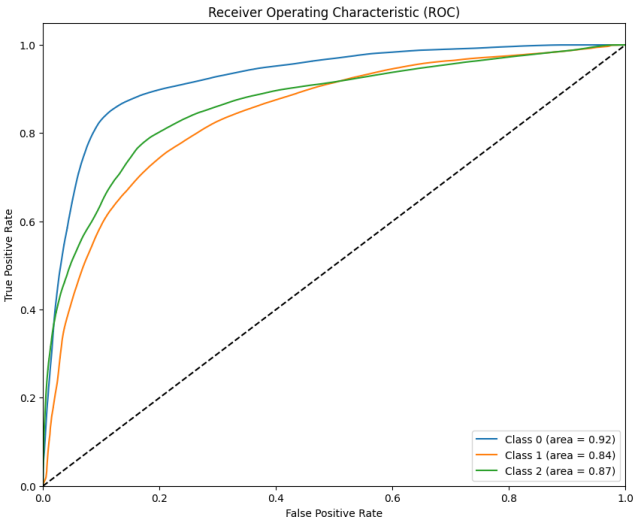


Fig. 15: Gráfica curva ROC

6 Conclusiones

Al unir las clases 1, 2 y 3 en una sola clase, el modelo parece haber mejorado en general. La reducción en el número de clases puede haber ayudado al modelo a aprender mejor las características distintivas entre las clases. Sin embargo, sigue habiendo algo de confusión entre las clases, pero esto es esperable dado que estas clases originalmente eran difíciles de diferenciar.

En resumen, los resultados muestran una mejora general en la precisión del modelo y en su capacidad para diferenciar entre áreas con nubes y áreas sin nubes. Las métricas sugieren que el modelo ahora tiene un mejor rendimiento en comparación con la versión anterior.

6.1. Trabajos futuros

Una vez finalizado este trabajo, vemos que un modelo de detección de nubes en imágenes satélite puede tener varias utilidades muy útiles en varios ámbitos.

Uno de ellos, y el más conveniente para seguir trabajando una vez concluido esta implementación. Es aplicarlo para hacer una eliminación de las nubes.

Este segundo paso, puede realizarse de varias formas:

1. Utilizar imágenes anteriores y posteriores de la misma área para interpolar y rellenar las áreas nubladas.
2. Utilizar datos de múltiples espectros (como infrarrojo y visible) para separar la información de la nube de la información del suelo.

Entre otras. De esta forma, ayudaremos a la monitorización ambiental, la agricultura, cartografía, expediciones, misiones espaciales, etc.

Referencias

[1] Jeppesen, J., Jacobsen, R. H., Inceoglu, F., & Toftegaard, T. S. (2019). A cloud detection algorithm for satellite imagery based on deep learning. *Remote Sensing Of Environment*, 229, 247-259. [Online]. Último acceso: 10 Marzo 2024. Disponible: https://doi.org/10.1016/j.rse.2019.03.039

[2] Francis, A., Sidiropoulos, P., & Müller, J. (2019). CloudFCN: Accurate and Robust Cloud Detection for Satellite Imagery with Deep Learning. *Remote Sensing*, 11(19), 2312. [Online]. Último acceso: 10 Marzo 2024. Disponible: https://www.mdpi.com/2072-4292/11/19/2312

[3] Gisadminbeers. (2019, 4 julio). Cómo generar imágenes satélite sin nubes - Gis&Beers. Gis&Beers. [Online]. Último acceso: 12 Marzo 2024. Disponible: https://www.gisandbeers.com/generar-imagenes-satelite-sin-nubes/#:~:text=Puedes%20emplear%20cualquier%20recurso%20sat%C3%A9lite%20y%20excluir%20su,mejorar%20la%20visualizaci%C3%B3n%20y%20corregir%20atmosf%C3%A9ricamente%20las%20im%C3%A1genes

[4] CloudSEN12: A Benchmark Dataset for Cloud Semantic Understanding. (s. f.). [Online]. Último acceso: 5 Abril 2024. Disponible: https://cloudsen12.github.io/

[5] Alonso, D. (2019). Combinación de bandas en imágenes de satélite Landsat y Sentinel. MappingGIS. [Online]. Último acceso: 23 Mayo 2024. Disponible:https://mappinggis.com/2019/05/combinaciones-de-bandas-en-imagenes-de-satelite-landsat-y-sentinel/

[6] Segmentación de imágenes U-Net en Keras. 21 de febrero de 2022. [Online]. Último acceso: 15 Abril 2024. Disponible:https://pyimagesearch.com/2022/02/21/u-net-image-segmentation-in-keras/

[7] Capa Conv2D. Documentación de Keras. Último acceso: 15 Junio 2024. Disponible:https://keras.io/api/layers/convolution_layers/convolution2d/

[8] BatchNormalization layer. Documentación de Keras. Último acceso: 15 Junio 2024. Disponible:https://keras.io/api/layers/normalization_layers/batch_normalization/

[9] Funciones de activación de capas. [Online]. Último acceso: 29 Junio 2024. Disponible:https://keras.io/api/layers/activations/

[10] SeparableConv2D layer. [Online]. Último acceso: 29 Junio 2024. Disponible:https://keras.io/api/layers/convolution_layers/separable_convolution2d/

[11] MaxPooling2D layer. Documentación de Keras. Último acceso: 15 Junio 2024. Disponible:https://keras.io/api/layers/pooling_layers/max_pooling2d/

[12] Add Layer. Documentación de Keras. Último acceso: 15 Junio 2024. Disponible:https://keras.io/api/layers/merging_layers/add/

[13] Conv2DTranspose layer. Documentación de Keras. Último acceso: 15 Junio 2024. Disponible:https://keras.io/api/layers/convolution_layers/convolution2d_transpose/

[14] UpSampling2D layer. Documentación de Keras. Último acceso: 15 Junio 2024. Disponible:https://keras.io/api/layers/reshaping_layers/up_sampling2d/

[15] Softmax layer. Documentación de Keras. Último acceso: 15 Junio 2024. Disponible:https://keras.io/api/layers/activation_layers/softmax/

[16] Optimizador Adam, 27 Mar 2024. [Online]. Último acceso: 10 Abril 2024. Disponible: https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam

[17] Pérdidas probabilísticas. Documentación de Keras. Último acceso: 15 Junio 2024. Disponible: https://keras.io/api/losses/probabilistic_losses/#:~:text=sparse_categorical_crossentropy%20function&text=Computes%20the%20sparse%20categorical%20crossentropy,to%20be%20a%20logits%20tensor.

[18] Intersection over Union (IoU). Encord Computer Vision Glossary. Último acceso: 15 Junio 2024. Disponible: https://encord.com/glossary/iou-definition/#:~:text=Intersection%20over%20Union%20(IOU)%20is,annotated%20region%20from%20a%20dataset.

[19] ModelCheckpoint. Keras 3 API documentation / Callbacks API / ModelCheckpoint. Último acceso: 15 Junio 2024. Disponible: https://keras.io/api/callbacks/model_checkpoint/

[20] QGIS. Un Sistema de Información Geográfica libre y de Código Abierto. Último acceso: 29 Junio 2024. Disponible: https://www.qgis.org/es/site/