2022-06

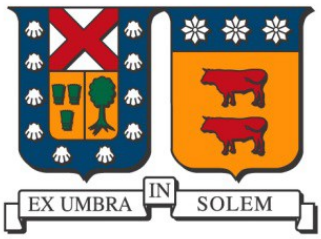
Automatización del proceso de obtención de cicatrices de incendios usando Sistemas de Información Geográfica (SIG) e Inteligencia Artificial (IA)

Mancilla Wulff, Ian Franco

Universidad Técnica Federico Santa María

Departamento de Ingeniería Química y Ambiental

Valparaíso, Chile



"Automatización del proceso de obtención de cicatrices de incendios usando Sistemas de Información Geográfica (SIG) e Inteligencia Artificial (IA)"

Ian Franco Mancilla Wulff

Tesis para optar al título de Ingeniero Civil Ambiental

Prof. Guía	:	Santiago García (Departamento Ingeniería Química y Ambiental)
Prof. correferentes	:	Prof. Andrés Weintraub (Universidad de Chile), Cristóbal Pais (UC Berkeley) y Jaime Carrasco (Universidad de Chile).

Junio del 2022

Agradecimientos

A mi madre, padre y Oma. A mis amigas y amigos que me apoyaron en el trayecto, tanto de la vida como de la universidad, gracias por tanto. A la universidad en sí, comenzando por Federico Santa María, y siguiendo con toda la comunidad sansana con la que compartí estos 7 años, contando a sus académicos, estudiantes, exestudiantes, ayudantes y funcionarios. Al equipo FONDEF y al ISCI por su confianza. A Chile por permitirme estudiar libremente. Y finalmente, a mí.

Dedicatoria

Dedico esta memoria a los ambientalistas, estudiantes de mérito, a quienes buscan la justicia para este planeta, y a mi madre, padre y Oma.

Resumen

Los incendios son un problema de gran interés en la actualidad, tanto en Chile como globalmente. En las últimas décadas se han visto tendencias crecientes en su ocurrencia y áreas afectadas, conllevando impactos ambientales, económicos, y sociales a su vez. Paralelamente, surge la necesidad de estudiar y cuantificar las superficies afectadas, para lo cual se requiere mapear cada cicatriz de incendio. Esto se realiza normalmente con la ayuda de imágenes satelitales, y hasta hace unos años, de algoritmos que requieren de iteraciones manuales o de una acuciosa supervisión. Las desventajas de este enfoque son su gran costo en capital humano, y los errores debido a sus limitaciones para captar la totalidad de cada cicatriz. Como alternativa a esto, se han ido desarrollando herramientas de Inteligencia Artificial (IA) en busca de automatizar y optimizar el proceso. En los últimos años, los enfoques más relevantes han sido los que usan *Deep Learning* (DL) y redes neuronales convolucionales (CNN), obteniendo mejores resultados usando procesos automáticos para data de los sectores específicos estudiados. De esta manera, este trabajo surge desde el proyecto FONDEF 2021 enmarcado en el estudio de incendios, donde se tiene como objetivo específico proponer una metodología para automatizar y optimizar el proceso de obtención de cicatrices de incendios de Chile, usando dos regiones prototipos que son de las más afectadas: Valparaíso y BioBío. La metodología de este trabajo utiliza un enfoque multitemporal con imágenes satelitales LANDSAT 5, 7 y 8 OLI con los incendios de estas regiones, desde las cuales se diseñaron dos datasets según el tamaño de las cicatrices y las imágenes pre y post-incendio: 1) All Sizes (AS), y 2) 128x128 (128), para entrenar, validar y testear separadamente un modelo U-Net, que es la CNN usada en este trabajo. Para mejorar el rendimiento del modelo, se realizó una optimización de hiperparámetros. Desde esto se obtuvieron los siguientes resultados para un testeo independiente con 100 imágenes representativas de la zona de estudio: un *Dice Coefficient* (DC)=0.932, error de omisión (OE)=0,049 y error de comisión (CE)=0,076 para el modelo AS; y un DC=0,843, OE=0,131 y CE=0,143 para el modelo 128. Esto indica que un dataset más balanceado logra un mejor rendimiento.

Palabras claves—áreas quemadas; deep learning; Landsat; redes neuronales convolucionales; segmentación.

Abstract

Fires are a problem receiving a lot of attention nowadays, as much in Chile as worldwide. Among the last decades, higher number of events and larger affected areas have been reported, producing environmental, economic, and social impacts alongside. Correspondingly, studying and quantifying the areas affected is necessary, mapping every fire scar. Normally, satellite data is used for this task, and in the last decade, iterative algorithms required of high supervision have been used to carry this on. The disadvantages of this approach are the human resources involved and the errors due to its limitations to capture the entire area of each fire scar. As an alternative, Artificial Intelligence (AI) tools have been developed in order to automate and optimize the process. During the last few years, Deep learning (DL) and Convolutional Neural Networks (CNN) have succeeded over all the previous approaches using automatic workflows for data of specific locations. Thus, this work originates from the FONDEF 2021 project on fire studies, where the specific objective is to propose a methodology to automate and optimize the burned area mapping process for Chile, using two prototype regions of most concern: Valparaíso y BioBío. A multitemporal approach using LANDSAT 5,7 and 8 OLI with the data of these regions is carried out, creating two datasets according to the fire scars and the pre and post fire image sizes: 1) All Sizes (AS), and 2) 128x128 (128), to train, validate and test separately a CNN variant used in this work, the U Net model. To improve the model performance, Hyperparameter Optimization (HPO) is implemented. The following results were achieved for an independent testing using 100 representative images of the study area: a Dice Coefficient (DC)=0.932, Omission error (OE)=0,049 and Commission Error (CE)=0,076 for the model AS, and DC=0,843, OE=0,131 and CE=0,143 for the model 128. This indicates that a more balanced dataset results on a better performance.

Keywords—burned areas; convolutional neural network; deep learning; Landsat; segmentation.

Acrónimos

128: Referido al dataset o modelo compuesto por imágenes de dimensiones 128x128

Adam Optimizer: *Adaptive Moment Estimation Optimizer*, Optimizador de estimación de momento adaptativo

ANN: *Artificial neural networks*

AS: All sizes, referido al dataset o modelo compuesto por data de los tamaños iniciales

BN: *Bayesian networks*

BCELoss: *Binary Crossentropy loss*

BRT: *Boosted regression trees*

CART: *Classification Regression Tree*

CE: Error de comisión, en inglés *Commission error*

CPU: Central Processing Unit

CNN: Red neuronal convolucional, en inglés *Convolutional Neural Network*

DC: *Dice Coefficient*

DL: *Deep Learning*

DT: *Decision trees*

FN: *False Negative*, falso negativo

FP: *False Positive*, falso positivo

GA: *Genetic algorithms*

GEE: Google Earth Engine

GP: *Genetic Programming*

GPU: Graphic Processing Unit

IA: Inteligencia Artificial

IQR: Rango intercuartil

KNN: *k nearest neighbor*

L5: Satélite Landsat 5

L7: Satélite Landsat 7

L8: Satélite Landsat 8

LULC: Land use land cover

ML: *Machine Learning*

MLC: *Maximum Likelihood Classification*

NB: *Naive Bayes*

NDVI: *Normalized Difference Vegetation Index*, Índice de Vegetación de Diferencia Normalizada

NIR: *Near Infrared*, luz infrarroja corta

OBC: Método de clasificación basado en objetos, OBC por sus siglas en inglés

OE: *Omission error*, error de omisión

PA: *Producer's Accuracy*

RdNBR: *Relative Delta Normalized Burned Ratio*, diferencia relativa de la razón de quema normalizada

RF: *Random Forest*

S1: Satélite Sentinel-1

S2: Satélite Sentinel-2

SGD: *Stochastic Gradient Descent*, gradiente estocástico descendente.

SIG; Sistema de Información Geográfica

SVM: *Support vector machines*

TN: *True Negative*, verdadero negativo

TP: *True Positive*, verdadero positivo

UA: *User's Accuraccy*

USGS: United States Geological Survey

Índice

Índice de figuras

Índice de tablas

Introducción

Los incendios son un problema muy importante del último siglo a nivel global, debido a sus impactos ambientales, económicos y sociales, que pueden ser todavía más críticos en un contexto de cambio climático. Más aún, se ha visto una tendencia creciente en la ocurrencia y las áreas afectadas a causa de incendios en Chile dentro de los últimos años (CONAF, 2022), por lo que es necesario estudiar estos fenómenos acuciosamente para planificar estrategias de prevención, manejo y restauración asociadas a cada siniestro, con el fin de intentar disminuir sus estragos. En particular, el mapeo de superficies quemadas de incendios es una herramienta valiosa para el estudio de estos, permitiendo estimar los impactos de cada evento, además de estudiar las dinámicas geoespaciales de los regímenes de fuegos, e incluso la recuperación post-incendio de cada incidente.

En la actualidad, los métodos de Inteligencia Artificial (IA) permiten el reconocimiento automático de objetos y de clases dentro de imágenes mediante distintas alternativas (Alzubaidi et al., 2021), facilitando los trabajos de metodologías previas a la incursión de la IA, que en muchos casos recurren a iteraciones manuales ajustando hiperparámetros para lograr sus funciones, lo cual es costoso en capital humano y conlleva limitaciones en sus rendimientos.

En el caso de utilizar metodologías sin IA para la obtención de cicatrices de incendios se encuentra la Corporación Nacional Forestal (CONAF), y también el Centro de Ciencia del Clima y la Resiliencia (CR^2), el cual recientemente recopiló un dataset de cicatrices de incendios de Chile (Miranda et al., 2022) disponible en: https://www.pangaea.de/tok/6dcc6e08241c5076ef6bff47bbe73014308d4881, del cual se seleccionaron los datos de las regiones de interés para este trabajo, Valparaíso y BioBío. Este dataset contenía las imágenes pre y post incendio en formato ráster, así como las mismas cicatrices de incendio, es decir, el área quemada asociada a un incendio separada del contexto en archivos apartes, que en este caso eran dos archivos, uno ráster y uno vectorial, entre otros archivos que no se utilizaron para este trabajo. Este dataset se construyó usando el Sistema de Información Geográfica (SIG) Google Earth Engine (GEE) mediante un código (https://code.earthengine.google.com/554027d16823525d890ab2f6c45167d9) desde el cual se podían comparar las diferencias entre las imágenes pre y post incendio, en cuanto a vegetación, humedad, entre otras variables; con el fin de obtener el área de incendio. En particular, se fueron modificando índices espectrales y áreas de búsquedas o *buffers* iterativamente hasta lograr capturar el área quemada asociada a cada incendio, hasta lograr incorporar la mayor superficie quemada posible, procurando asegurar que el área seleccionada estuviera efectivamente quemada para no tener errores de comisión (i.e. incluir área no quemada como tal), pero a causa de esto, generando errores de omisión significativos (i.e. omitir área que está efectivamente quemada). En la Figura 0.1 se resume este proceso de obtención de cicatrices de incendios sin IA, mientras que en el Anexo B se esquematiza con mayor detalle el proceso con la toma de decisiones asociada.

Modelos de *Machine Learning* (ML) y *Deep Learning* (DL) se han usado para el mapeo de áreas quemadas (Chuvieco et al., 2019), destacando por sus resultados a las redes neuronales convolucionales (CNN, por *Convolutional Neural Network* en inglés) (Pinto et al., 2021), y dentro de estas, especialmente al modelo U-Net (Hu et al., 2021; Knopp et al., 2020), logrando superar el rendimiento de los modelos previos existentes. Sin embargo, las investigaciones realizadas han sido enfocadas para determinadas coberturas de suelos y

regiones, donde hasta la fecha no se ha incluido a Chile, por lo que no se asegura un ajuste equivalente para los registros nacionales.
Los modelos de IA requieren de datos de entrenamiento, validación y testeo, los cuales deben contar con al menos un input y un *label*, o data de referencia. En este caso, para identificar el área quemada, se requiere de una imagen con una cicatriz de incendio presente (*input*) y un vector o ráster con la figura ya reconocida (*label*) respectivamente. En particular, el uso de imágenes satelitales multibandas ha sido lo más utilizado para esta tarea, dada su disponibilidad sin costo y con alcance global para varios satélites, desde donde se destacan los satélites de mediana resolución como Landsat (30 m resolución), y de mediana a alta resolución como el Sentinel 2 (10 m resolución). Un enfoque multitemporal (i.e. incluyendo imágenes pre y post-incendio) tiene las ventajas de poder captar las requemas identificando las diferencias entre las imágenes pre y post, así como ignorar quemas previas, además de aportar una fácil visualización de cada registro.

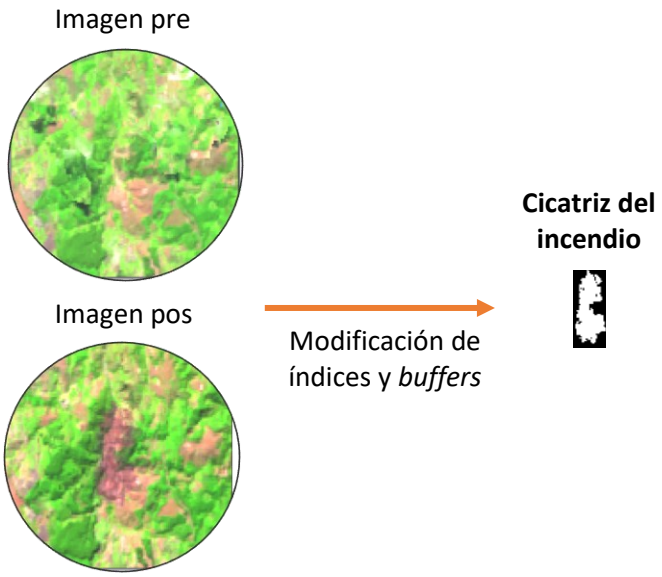


Figura 0.1 Obtención de cicatrices de incendios sin IA

En vista de lo anterior, se tiene como objetivo proponer una metodología para automatizar y optimizar el proceso de obtención de cicatrices de incendios de Chile usando el modelo U-Net con un enfoque multitemporal, teniendo como regiones prototipo a BioBío y Valparaíso.

Objetivos

i. Objetivo General

- Automatizar el proceso de obtención de cicatrices de incendios utilizando herramientas de IA.

ii. Objetivos Específicos

- Colaboración con el proyecto FONDEF 2021 enmarcado en el estudio de incendios y con sus instituciones asociadas.

- Ahorrar horas hombre y mejorar la eficiencia del proceso actual de recopilación de cicatrices de incendios, enfocado en las regiones de BioBío y Valparaíso como regiones prototipo para el modelo, pero apuntando a ser adaptable a otras zonas geográficas.
- Desarrollar el código para el procesamiento de las imágenes, así como para la clasificación de los pixeles quemados y obtención de las cicatrices de incendios.

1 Antecedentes

1.1 Trabajo previo

Inicialmente, se presentan y definen las métricas que se utilizan para la validación de modelos de mapeo de áreas quemadas, con el fin de poder comprender el desempeño de cada uno de estos, que son presentados posteriormente en la revisión bibliográfica, y para así poder comparar las alternativas existentes. Posteriormente, se presenta el resumen de los modelos más importantes, desde los más antiguos hasta los más actuales, dándole énfasis a los modelos desarrollados desde el 2015 a la actualidad que han usado imágenes de mediana a alta resolución, puesto que son las que se usarán en este trabajo. El objetivo de esto es evaluar las investigaciones más recientes que han obtenido mejores resultados, indicando sus métricas cuando han sido reportadas y permiten su comparación con los otros modelos.

Es necesario comentar que para el entrenamiento del modelo a desarrollar se usarán imágenes satelitales pre y post-incendio, provenientes de los satélites Landsat 5, 7 y 8 OLI (reajustadas a 30 m de resolución espacial) desde la base de datos señalada que contiene las imágenes correspondientes con los cicatrices de incendios, además de las mismas cicatrices en formato ráster y vectorial (Miranda et al., 2022). Esta base de datos se obtuvo usando GEE, que permitió acceder y procesar las imágenes satelitales desde la plataforma, hasta la obtención de las cicatrices de incendios correspondientes, comprendiendo entre los archivos a las imágenes pre y post incendio en formato ráster, las cicatrices de incendios en formato ráster y vectorial, entre otros. Para la selección de imágenes del dataset se utilizaron los registros de la Corporación Nacional Forestal (CONAF) (www.conaf.cl/conaf/seccion-stadisticas-historicas.html) desde el 1985 hasta el 2018 para geolocalizar los eventos y luego adquirir las imágenes satelitales asociadas a las fechas de cada incendio reportado. Posteriormente, en lo que concierne a este trabajo, se procedió a seleccionar las imágenes de las regiones de Valparaíso y Biobío para el ajuste del modelo a desarrollar, que son regiones que han sido muy afectadas por los incendios de la última década en Chile, por lo que se consideran de alto interés. De este modo, se tiene una amplia base de datos disponible. Específicamente, se tienen disponibles los archivos para cada registro con un incendio detectado, donde los relevantes para este trabajo constan de los archivos ráster y vectoriales de las huellas de los incendios, más los ráster de las imágenes pre y post-incendio.

1.1.1 Métricas para la validación de modelos

Para poder comparar y analizar los modelos, se deben tener en cuenta las métricas de comparación entre los modelos, que se basan generalmente en cuanto al cálculo de los pixeles que han sido correctamente clasificados por el modelo en una categoría (e.g área quemada, no quemada), comparándose con datos de referencia. Las notaciones típicas se presentan en la Tabla 1.1.

Tabla 1.1 Matriz de error de predicción

	Datos de referencia

Precisión del producto	Quemado	No quemado	Total fila
Quemado	P_{11}	P_{12}	P_{1+}
No quemado	P_{21}	P_{22}	P_{2+}
Total columna	P_{+1}	P_{+2}	1

Nota: Se expresa P_{ij} como la proporción de área correctamente clasificada, ubicada en las celdas de la diagonal, o incorrectamente clasificadas, en las celdas fuera de la diagonal. Esto respecto a los datos de referencia. Adaptado de (Padilla et al., 2015)

Es necesario mencionar, que estas notaciones se pueden encontrar también como verdadero positivo (TP), verdadero negativo (TN), falso positivo (FP) y falso negativo (FN) para P_{11}, P_{22}, P_{12} y P_{21} respectivamente.

A partir de lo anterior, se definen a continuación las métricas más empleadas (Barsi et al., 2018; Padilla et al., 2015; Patel & Kaushal, 2010):

Overall Accuracy (OA): Se define como:

$$OA = \frac{P_{11}+P_{22}}{N} = \frac{TP+TN}{TP+FP+FN+TN} \quad (1)$$

Donde N corresponde al número interpretable de pixeles para la región de interés.

No obstante, se debe mencionar que esta métrica no aporta a comparar o evaluar modelos cuando dentro de las imágenes una categoría tiene mucha mayor preponderancia que la otra (Fielding & Bell, 1997). En un estudio reciente sobre el desempeño de 6 modelos globales de reconocimiento de áreas quemadas (Padilla et al., 2015) se concluyó que cuando en una región se tiene un área quemada pequeña, se encontrarán altos valores de OA a pesar de que los modelos tengan precisión moderada para detectar pixeles no quemados, y baja para detectar pixeles quemados.

Commission error ratio (CE), razón de error de comisión:

$$CE = \frac{P_{12}}{P_{11}+P_{12}} = \frac{FP}{TP+FP} \quad (1.1)$$

Esta métrica mide los fallos del modelo por sobreestimar una clase. Nótese que alternativamente se utiliza la precisión, o llamada *User's Accuracy* (UA), que es igual al valor restante para llegar a 1.

$$User's\ Accuracy = \frac{N°\ de\ pixeles\ clasificados\ correctamente\ para\ una\ clase}{N°\ de\ pixeles\ clasificados\ en\ esa\ clase} \cdot 100 \quad (1.2)$$

Omission error ratio (OE), razón de error de omisión:

$$OE = \frac{P_{21}}{P_{11}+P_{21}} = \frac{FN}{TP+FN} \quad (1.3)$$

Ahora, este error se produce por subestimar u omitir a una clase. Al igual que para el caso anterior, se utiliza en algunas referencias el *Recall* o también llamada *Producer's Accuracy* (PA) como métrica en vez del OE, que corresponde al resto de esta razón para llegar a 1.

$$Producer's\ Accuracy = \frac{N°\ de\ pixeles\ clasificados\ correctamente\ para\ una\ clase}{N°\ de\ pixeles\ de\ referencia\ de\ esa\ clase} \cdot 100 \quad (1.4)$$

Dice Coefficient (DC): Este coeficiente sintetiza el CE y OE, y se define como:

$$DC = \frac{2P_{11}}{(2P_{11}+P_{12}+P_{21})} = \frac{2P_{11}}{P_{1+}+P_{+1}} = \frac{2TP}{2TP+FP+FN} \quad (1.5)$$

Este indicador tiene sensibilidad a la interpretación probabilística (Padilla et al., 2015). Usando un clasificador para identificar pixeles quemados, sean datos de un producto o de referencia, se traduce como la probabilidad condicional de que otro clasificador también identifique el área como quemada. Se ha reconocido como un indicador apropiado cuando hay una pequeña fracción de área quemada dentro de las imágenes (Chuvieco et al., 2018; Padilla et al., 2017; Padilla et al., 2015). Además de este, se encuentra otro coeficiente muy similar, ampliamente usado en problemas de segmentación, que es el *Intersection over Union* (IoU) (o conocido también como *Jaccard Index*) o traducido, intersección sobre la unión:

$$IoU = \frac{TP}{(TP+FP+FN)} \quad (1.6)$$

Por último, se presenta el coeficiente *Cohen's Kappa coefficient* (Cohen, 1960, como se citó en Addison & Oommen, 2018) el cual es un indicador de confianza interlocutor que se ha probado para aplicaciones de sensores remotos (Congalton, 1991). Donde p_o es la precisión general observada y p_e es la probabilidad esperada cuando ambos puntajes se asignan como etiquetas aleatorias:

$$K = \frac{p_o - p_e}{1 - p_e} \quad (1.7)$$

Se define p_o como la precisión total:

$$p_o = \frac{TP+TN}{TP+FP+TN+FN} \quad (1.8)$$

Por último, p_e, es la precisión aleatoria, que se define como:

$$p_e = \frac{(TN+FP)+(TN+FN)+(FN+TP)+(FP+TP)}{(TP+FP+TN+FN)^2} \quad (1.9)$$

Además de lo anterior, se debe tener en cuenta que para la validación de los modelos, un requisito común según el *Committee on Observation Satellites Calibration and Validation protocol* (CalVal protocol), es usar imágenes de referencia de mejor resolución (Boschetti et al., 2009, como se citó en (Pinto et al., 2020)), sean de los satélites Landsat/Sentinel u otros satélites de mediana a alta resolución. Esto es sumamente relevante para poder comparar las métricas de modelos, ya que solo se podrá hacer si se validaron con imágenes de resolución similar.

1.1.2 Revisión bibliográfica

A continuación, se presenta la comparación de los modelos más relevantes encontrados, comenzando por los modelos globales que se han desarrollado con sus rendimientos respectivos, y luego detallando los resultados de modelos *Machine Learning* (ML) y *Deep Learning* (DL) de interés. El enfoque en estos últimos es debido a que han alcanzado mejores resultados que los modelos de algoritmos de varias fases tradicionales, que utilizan índices de forma iterativa, y en muchos casos datasets de satélites de mediana baja resolución para el mapeo de áreas quemadas y su validación, por lo cual han obtenido errores considerables, lo cual se evidencia ampliamente (Boschetti et al., 2015; Liu et al., 2018; Roteta et al., 2019; Roy et al., 2019; Trisakti et al., 2017). Algunos de los problemas con este método son la calibración de los umbrales a usar en terrenos complejos, los posibles reajustes de estos para otros lugares o paisajes, y la falta de análisis de contexto espacial, imposible con esos algoritmos.

1.1.2.1 Modelos globales

El problema de mapear cicatrices de incendios ha sido un área de estudio desde la década de los 90s, desde lo cual se han desarrollado varios modelos globales que detectan estas, usando satélites de resolución gruesa o baja resolución (<300 m por pixel). Sin embargo, se ha visto que la resolución espacial utilizada, es un factor clave para la determinación precisa del área quemada (Hamilton et al., 2019; Roteta et al., 2019), lo cual se comprueba claramente el año 2015 al evaluarse la precisión de 6 modelos globales de baja resolución existentes (Padilla et al., 2015), que se detallan en la Tabla 1.2, validándose con imágenes de resolución media (Landsat TM/ETM+); donde se encontraron errores de comisión mayores a un 40% y sobre un 65% en errores de omisión en todos ellos. Como se mencionaba anteriormente, el OA no sirvió como un buen estimador para la evaluación de la precisión de los modelos, dado que pese a sus valores muy cercanos al 100%, estos productos consideraban más pixeles quemados y al mismo tiempo menos pixeles quemados de los existentes, lo cual se refleja en sus CE y OE respectivamente.

Tabla 1.2 Comparación modelos globales

Producto	Características del sensor	OA (%)	CE (%)	OE (%)	DC (%)
MCD45	MODIS (500 m)	99,7	46	72	37
MCD64	MODIS (500 m) y MODIS de anomalías termales (1 km)	99,6	42	68	42
GEOLAND2	SPOT VGT (1 km)	99,6	74	91	13
MERGED_CCI	SPOT VGT y MERIS (300 m)	99,2	87	79	16
MERIS_CCI	MERIS (300 m)	99,6	64	76	29
VGT_CCI	SPOT VGT (1 km)	99,2	94	93	7

Fuente: Adaptado de (Padilla et al., 2015)

De esta manera, la resolución espacial ha sido una limitante de todos los modelos globales, la cual está entre 250 m (FireCCI50) a 1 km (e.g en el L3JRC) (Chuvieco et al., 2019). Sin embargo, el uso de los satélites de media resolución (10-30 m), ha demostrado importantes mejoras en la precisión obtenida al mapear cicatrices de incendios, como se ven claramente Roteta et al. (2019) usando imágenes del Sentinel-2 (S2) para el mapeo de cicatrices de incendios en África Subsahariana mediante un algoritmo de dos fases, obteniendo cerca de un 80% de aumento en la cobertura de área quemada respecto al producto de la NASA, MCD64A1 c6 para el mismo periodo. Encima, se comenta como causa de esta gran disparidad de rendimiento, que los productos globales de baja resolución detectan raramente incendios de superficies menores a 100 ha.

1.1.2.2 Modelos de Machine Learning tradicionales

Con los avances tecnológicos actuales, se han desarrollado una gran cantidad de modelos de ML en el área de estudio de incendios, y análogamente, en el mapeo de áreas quemadas de incendios y predicción de su severidad, según se reportó en una revisión del estado del arte de los modelos existentes de ML en aplicaciones asociadas a incendios hasta finales del 2019 (Jain et al., 2020). Dentro de los modelos usados en estas dos subtemáticas se encuentran el uso de: *Neuro-fuzzy models* (NFM), *Support vector machines* (SVM), *k – means clustering* (KM), *Genetic algorithms* (GA), *Boosted regression trees* (BRT), *Artificial neural networks* (ANN), *Decision trees* (DT), *Random forest* (RF), *k nearest neighbor* (KNN), *Maximum entropy* (MaxEnt), entre otros. Se detallan las participaciones de cada modelo registradas en la Tabla 3.

Tabla 1.3 Modelos de Machine Learning usados en mapeo de perímetros y severidad de incendios

Modelos	*NFM*	*SVM*	*KM*	*GA*	*BRT*	*ANN*	*DT*	*RF*	*KNN*	*MAXENT*	*Otros*
Mapeo de perímetros y severidad de incendios	1	12	1	2	1	6	1	4	2	1	6

Nota: Estos modelos fueron los reportados por el estado del arte referido, sin embargo, se han registrado más aplicaciones orientadas a esta área de estudio a la fecha, donde las de mayor interés se detallan en esta sección. Adaptado de (Jain et al., 2020).

Dentro de los modelos que han utilizado imágenes de mediana a alta resolución, se encuentran principalmente de SVM, de árboles de decisión, como RF y *Classification and Regression Trees* (CART), el cual se basa en los algoritmos de los DT; y de *Maximum likelihood classification* (MLC).

Primero, en lo que concierne a SVM, Zhao et al. (2015) usaron una metodología con una clasificación previa de candidatos de disturbios usando el algoritmo *vegetation change tracker* (VCT), y luego SVM para el mapeo de varios tipos de estos, entre ellos, incendios forestales, alcanzando para esta categoría un UA de 96% y PA de hasta un 73.2%. Por otra parte, Hamilton et al. (2017) emplearon imágenes de alta resolución provenientes de un *Small*

unmanned aircraft system (sUAS) con 12 megapixeles (MP) de resolución espacial (véase la Figura 1.1, comparando las imágenes sUAS usadas vs el L8), para ejecutar un modelo SVM que mapeara inicialmente el área quemada (i.e. cenizas en la imagen), y clasificara posteriormente según la severidad de los incendios, incorporando la textura del entorno como data de entrada, demostrando que esta mejoró los resultados del modelo en ambas etapas, sin embargo, ambos casos presentaron rendimientos altos. Otra aplicación que usó una metodología similar fue la que probaron posteriormente Hamilton et al. (2019), demostrando la relevancia de la alta resolución espacial de sUASB vs imágenes L8 entrenando separadamente un modelo SVM.

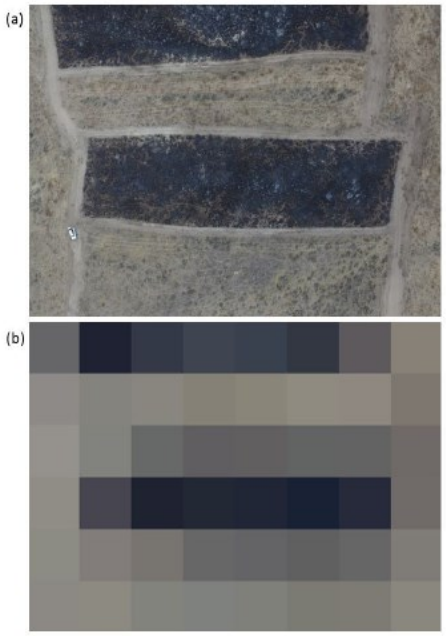


Figura 1.1 Comparación imagen del Landsat-8 vs imagen de alta resolución. (Hamilton et al., 2017).

Nota: a) Imagen obtenida del Suas (12 MP de resolución, equivalente a 6 centímetros de resolución espacial) a 120 metros de altura, y b) Misma escena obtenida desde el L8 (Ajustada a 30 m de resolución).

En cuanto a RF y CART, Meddens et al. (2016) estudiaron ambos algoritmos para un mismo dataset, obteniendo resultados exitosos y cercanos entre ellos, con RF marginalmente superior. Sin embargo, destacaron el desempeño de CART debido a su identificación de umbrales específicos, y a su mucho menor uso de recursos respecto a RF para interpretación de clases. Además de esto, se destaca también la característica conocida de los CART de tener un buen desempeño ante datasets de gran volumen. Otro caso de uso se ve en Thariqa et al. (2016), donde se comparan árboles de decisión analizando su componente espacial.

Adicionalmente, se ha usado también el modelo MLC. En primera instancia en comparación contra un modelo de agrupación ISO (aprendizaje no supervisado), realizado por Oumar (2015), incorporando muestras de datos pequeñas y de un área de estudio específica, donde el MLC se desempeñó de manera exitosa a diferencia del segundo modelo. Posteriormente,

Sertel & Alganci (2016) usaron imágenes de alta resolución SPOT-6 (1.5 m de resolución espacial) para un caso de estudio de incendio, comparando MLC, como método basado en clasificación de pixeles para la categorización de coberturas de suelo y de uso de suelo, denominado *land use and land cover* (LULC); versus un método de clasificación basado en objetos (OBC por sus siglas en inglés). De esta manera, se obtuvieron resultados muy buenos para el OBC, mientras que no tanto para el MLC, lo cual se atribuye en parte a la misma resolución, que permitía notar la heterogeneidad en la imagen.

Se han realizado igualmente comparaciones de modelos, como fue visto por Bar et al. (2020) donde se superan los resultados de SVM. Si bien se usaron imágenes de baja resolución para la validación del modelo, que había sido entrenado con imágenes L8 y S2, se obtuvieron mejores resultados para mapeo de incendios forestales (ciñéndose a categorías de bosques) con RF y CART por sobre SVM. Otro hallazgo fue que presentaron mejores resultados usando las imágenes del L8 en vez del S2, lo cual se pudo atribuir a las bandas espectrales utilizadas.

Al igual que en el caso anterior, Cabral et al. (2018) compararon algoritmos de ML con imágenes L7 y L8, reportando que el de *Genetic Programming* (GP) superaba en dos de las tres pruebas a CART y MLC, figurando con un kappa de 0.93 y un DC de 0.94 en el mejor caso, pero presentando valores de 0.49 y 0.50 en su peor prueba respectivamente (considerando ambas variaciones de GP estudiadas). Sin embargo, sus métricas deficientes señaladas, se atribuyen en parte a un error en la data de referencia, que el modelo aprendió y ejecutó en los *output*, al igual que el algoritmo CART.

Mientras que se han visto distintas conclusiones en modelos que han usado imágenes de baja resolución, como en Pereira et al. (2017), donde se usaron solo imágenes de entrenamiento positivas al algoritmo SVM, es decir, con áreas quemadas presente, y filtrando las imágenes que tuvieran incendios todavía activos, para evitar errores de omisión, obteniendo mejores resultados generales que el producto MCD64A1 en cuanto a OA y CE, pero sin desmarcarse de los bajos rendimientos de los modelos que usan imágenes de baja resolución espacial. Asimismo se ve en Mithal et al. (2018), donde se usan imágenes MODIS (500 m) en un modelo que usa Neural Networks (NN), sin obtener mucho mejores resultados que el producto MCD64A1 en cuanto a PA ni UA, pero logrando mapear una cobertura mucho mayor que el producto a pesar de ello.

En síntesis, de todas las aplicaciones de ML tradicionales revisadas, se encuentran variaciones de rendimiento entre sus modelos constantemente, lo cual no asegura una correcta precisión para distintos tipos de paisajes y de condiciones para otros incendios. Por añadidura, se deben notar 4 otras conclusiones: 1) la resolución espacial es muy importante para mejorar la precisión de cobertura alcanzadas, 2) la incorporación de contextos espaciales ha mejorado el rendimiento de modelos, 3) el enfoque OBC puede ser más beneficioso que la clasificación basada en pixeles en imágenes de muy alta resolución espacial, y 4) es crucial

tener inputs de entrenamiento y testeo muy bien delineados y/o clasificados para no dar pie a errores.

1.1.2.3 Modelos de Deep Learning

Actualmente, ha ido en aumento el uso de modelos DL en problemas de segmentación de objetos y de clases dentro de imágenes, y especialmente el de CNN dentro de estos modelos (Alzubaidi et al., 2021).

A modo de ejemplo, una aplicación de DL en el mapeo de áreas quemadas estudiado por Arruda et al. (2021), con el uso de una red neuronal de varias capas, que es la *Multilayer Perceptron Network* (MLPN), aunque se tuvo un alto OE, de 0.245, pudiéndose mejorar ese rendimiento.

Ahora, las CNN se han utilizado ampliamente en tareas de clasificación de imágenes satelitales, dadas sus ventajas varias, entre ellas: la selección automática de los atributos más relevantes del input sin supervisión humana, ahorrando interacciones innecesarias; la representación equivalente del input, y su capacidad de detectar bordes y formas, más allá de atributos aleatorios detectados en herramientas de ML, o en redes neuronales simples (de una capa) por cada neurona de estos modelos.

Dos estudios que usan CNN en el mapeo de áreas quemadas se ven, en primer lugar, con el modelo convolucional llevado a cabo recientemente por Pinto et al. (2021), el cual se destaca bastante por sus métricas obtenidas y metodología. Se utilizó un enfoque multitemporal con el uso de dos satélites: el VIIRS y el S2, para el mapeo preciso en cuanto a resolución temporal y espacial de los incendios, con una red convolucional que integraba capas *ChannelLinear* para aplicar capas de 1D con el fin de combinar los atributos basados en sus pixeles y así poder aprender sobre características espectrales de forma no lineal. Así, se obtuvieron DC muy exitosos, con un 0.97 general, y superiores a 0.92 en 5 de 6 pruebas de testeo, y esto con alrededor de 160.000 parámetros de aprendizaje, que es un número muy bajo con respecto a otros modelos de DL. Cabe mencionar que este estudio incorporó el uso del BA-Net, que es un modelo de DL que se usa en este caso para trabajar con las imágenes multitemporales del VIIRS, para así poder datar los incendios con mayor precisión.

En segundo lugar, se ven otras aplicaciones con el modelo AlexNET de Belenguer-Plomer et al. (2021), y en lo visto por Zhang et al. (2019), donde en ambos casos usan imágenes radares S1, y en el primero también evaluando imágenes ópticas S2.

Agregando a lo anterior, destaca dentro de las CNN la utilización del modelo U-Net (Ronneberger et al., 2015), el cual se originó para aplicaciones biomédicas, abarcando la segmentación de estructuras neuronales de imágenes de microscopios, buscando sobrellevar el problema de no contar con suficiente data para entrenar los modelos ya existentes, y al mismo tiempo, teniendo como objetivo tener una localización precisa de las clases a reconocer, mientras se sigue capturando el contexto. De tal forma, se superaron los resultados del mejor modelo de convolución previo para esta tarea en el concurso ISBI asociado a esta

aplicación, y esto utilizando solo 30 imágenes de entrenamiento, sometidas a aumentación con deformaciones elásticas. De esta manera, se han originado diversos artículos específicos que han estudiado el mapeo de cicatrices de incendios (de Bem et al., 2020; Hu et al., 2021; Knopp et al., 2020; Pinto et al., 2020; Shamsoshoara et al., 2021).

En alusión a las aplicaciones del U-Net, se destaca la investigación de Knopp et al. (2020), donde se usaron imágenes monotemporales (i.e. solo post-incendio, sin incluir imágenes pre-incendio) obtenidas por el S2 para entrenamiento, validación y testeo; lo cual se hizo a partir de 184 imágenes que fueron recortadas a 2637 unidades para la repartición de los set de entrenamiento, validación y testeo, y luego se triplicaron con otras operaciones de aumentación hasta obtener 6272 muestras para el entrenamiento. Los resultados para 545 secciones de imágenes de testeo fueron de un kappa de 0.94 y de un 5% en CE y OE por igual. Además de esto, se comparó el mismo dataset de testeo usando un modelo de RF, para el cual se obtuvo un kappa de 0.87, un CE de 7% y un OE de un 13%. Otro punto fue el tiempo de inferencia de la *central processing unit* (CPU), que fue de casi la mitad para el modelo U-Net respecto al RF, mientras que el tiempo de la *graphics processing unit* (GPU) fue unas 15 veces más rápido para el U-Net igualmente.

Una segunda referencia donde sobresale el modelo U-Net, se ve en Hu et al, (2021), con un enfoque monotemporal donde se hace una comparación de numerosos métodos para mapeo de áreas quemadas, incluyendo opciones de DL: U-Net, HRNet, Fast-SCNN y DeepLabv3+; de ML, desde donde se hizo una evaluación preliminar entre 13 algoritmos, seleccionando a los mejores, que fueron: *Light Gradient Boosting Machine* (LightGBM), KNN y RF; y de métodos tradicionales basados en el índice NBR (por sus aplicaciones uni-temporales). De tres zonas de testeo, U-Net lideró en general con sus resultados, incluso presentando valores muy buenos en una prueba de traspaso de imágenes de testeo desde el S2, que se usó para su entrenamiento, a L8 (para ver su rendimiento en la transferencia, sin cambiar el entrenamiento). Se concluye a partir de esto la predominancia de los modelos DL por sobre el resto, pese a que los otros métodos tuvieron mejores resultados en mapeo de áreas quemadas dispersas en bosques boreales, se nota la capacidad de su apreciación contextual y espacial que les hace ser la mejor alternativa.

En otras aplicaciones de segmentación de clases dentro de imágenes satelitales, se puede destacar que Wei et al. (2019) desarrollaron un modelo U-Net para la clasificación de cultivos en China usando el sensor S1 SAR para la captación de imágenes radares, y compararon el desempeño del modelo contra RF y SVM, obteniendo los mejores resultados U-NET (kappa 0,82), luego SVM (kappa 0,75) y finalmente RF (kappa 0,74), lo cual corrobora una vez más el rendimiento superior de este modelo por sobre los modelos tradicionales de ML.

1.1.2.4 Resumen modelos

En la Tabla 1.4 se presenta el resumen de los modelos vistos y comentados en esta revisión bibliográfica aplicados en mapeo de incendios, comparando los modelos registrados desde el año 2015 sin considerar modelos globales, en cambio enfocándose en los modelos de ML y

DL. No obstante, se han incorporado también algunos modelos que han utilizado algoritmos sin IA para reforzar la decisión de optar por modelos que usen estas herramientas, dadas sus mejores métricas.

Tabla 1.4 Resumen modelos de mapeo de incendios 2015-2021

CNN		DL	ML tradicionales							Algoritmos
U-NET	Otros		NN	SVM	RF	CART	GP	MLC	Otros*	
5	3	4	1	5	5	4	1	3	2	5

Nota: Algunos de estos modelos señalados se han usado simultáneamente en un mismo estudio. 1) Otros: Comprende al modelo Alex-Net, el modelo convolucional de Pinto et al. (2021), y a una aplicación de implicit Radar Convolutional Burn Index (RCBI). 2) DL: Corresponden a los modelos MLPN, HRNet, Fast-SCNN y DeepLabv3+. 3) Otros*: KNN y LightGBM.

1.1.2.5 Selección del modelo

En conclusión, a partir de esta revisión se escoge al modelo U-Net para este trabajo, considerando su acoplamiento correcto a la metodología proyectada y a los objetivos asociados, usando imágenes satelitales Landsat con los incendios históricos de las regiones de Valparaíso y BioBío de Chile, siendo especialmente útil al considerar que este modelo requiere de menos muestras de entrenamiento que los ML convencionales, pudiendo así escoger y seleccionar con mayor escrutinio las cicatrices de incendios para el aprendizaje óptimo del modelo. Otro motivo para esta decisión, ha sido su amplia validación, como es la recién comentada, y su comparación con otros modelos, donde ha superado a los demás en casi todas las pruebas, a diferencia del modelo convolucional antes comentado (Pinto et al., 2021), o el HR-Net.

1.2 Redes neuronales convolucionales y U-NET

Como se ha mencionado previamente, las CNN son actualmente las redes más usadas dentro de las alternativas de DL en muchas aplicaciones (Alzubaidi et al., 2021), incluyendo la segmentación semántica en imágenes como una aplicación importante, y como se señaló en el estado del arte (sección 1.2.2.3), el mapeo de áreas quemadas no es la excepción.

Las CNN se constituyen por las operaciones convolucionales discretas como su unidad fundamental. En particular, en aplicaciones de CNN con imágenes, estas operaciones matemáticas usan filtros o *kernels* que operan con todos los valores numéricos correspondientes a los pixeles de las imágenes ingresadas al modelo, creando nuevas relaciones algebraicas que permiten que una función de pérdida asocie cada uno de los aspectos de la imagen a un factor positivo o negativo para la predicción de la clase u objeto deseado. Con esto, se pueden relacionar colores, formas o interacciones de distintos atributos dentro de la misma imagen, inicialmente con relaciones abstractas, con numerosos factores o pesos para cada una de estas interacciones, lo que permite finalmente predecir el objeto, clase o atributo con un menor error de predicción. Aquí cumple su rol la función de pérdida mencionada, que minimiza el error entre el output y la data de referencia, recalculando cada uno de los pesos hasta alcanzar su valor mínimo óptimo.

La operación convolucional ocurre entre un kernel y la matriz del input. Los kernels o filtros son matrices formadas por números discretos llamados pesos, y se establece su cantidad para cada convolución o bloque antes de iniciar el entrenamiento. Más claramente, la convolución consta de un producto punto entre el kernel y la matriz del *input* que recorre la imagen tanto en el eje horizontal como vertical.

Esto se ve claramente en la Figura 1.2, usando un kernel 2x2. De esto se obtiene el *output*, conocido como *output feature map*, traducido en español, el mapa de atributos de salida. Adicionalmente, en la Ec. 1.10 se presenta su operación matemática aplicada a datos multidimensionales como imágenes. Se define o como la salida de esas coordenadas respectivas, F (k_w x k_h x k_d) es el kernel con sus dimensiones de ancho (w), alto (h) y número de filtros (d), I es la imagen, de dimensiones (x,y,z), mientras que b representa al *bias*. Por último, se debe señalar que en este caso se asume un relleno con el objetivo de mantener las mismas dimensiones del input tras la operación, por lo que los límites de la sumatoria consideran las celdas para esto.

$$o^{(x,y,z)}(I) = b + \sum_{i_x=1-[\frac{k_w}{2}]}^{[\frac{k_w}{2}]} \sum_{i_y=1-[\frac{k_h}{2}]}^{[\frac{k_h}{2}]} \sum_{i_c=1}^{d} F_z(i_x, i_y, i_c) \cdot I(x + i_x, y + i_y, i_c) \quad (1.10)$$

Se debe notar que la salida tendrá la misma dimensión de profundidad que la cantidad de filtros aplicados. Alternativamente, se puede usar un sesgo o *bias*, que es un vector adicional de pesos que se utiliza sumado como una neurona más, no vinculada a las capas anteriores, pero a la cual se le adaptan los pesos igualmente, lo cual es útil por ejemplo cuando un atributo tiene valor de cero, y se espera reproducir otro valor, además de que otorga mayor complejidad al modelo.

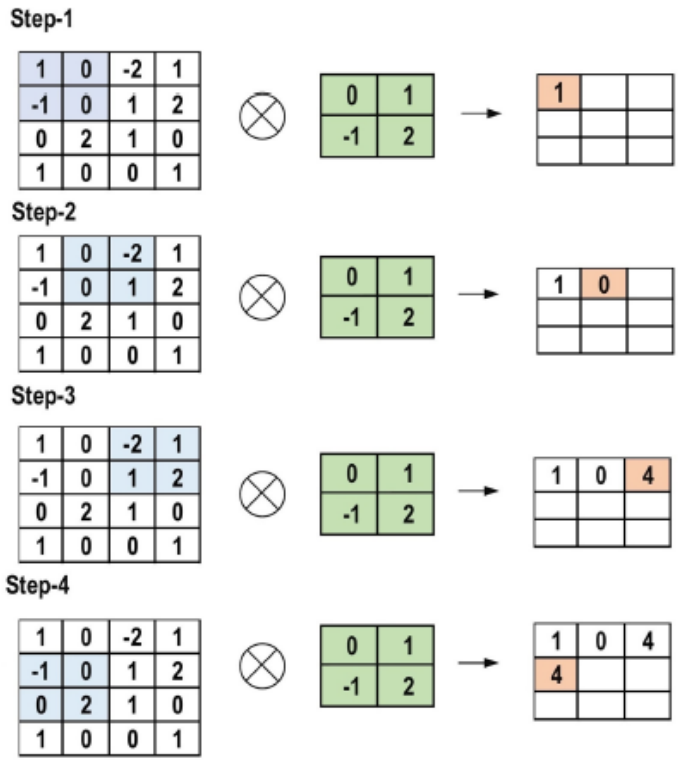


Figura 1.2 Ejemplo estructura típica de una CNN.(Alzubaidi et al., 2021)

Los modelos CNN se constituyen a modo general por 3 aspectos (Goodfellow et al., 2016): Interacciones selectivas entre las neuronas, compartición de parámetros y representaciones equivalentes. Esto se entiende considerando que pocos valores de un kernel contribuyen a un *output feature map*, y de la misma manera, se puede ver la compartición de parámetros considerando que los mismos pesos se utilizan para múltiples secciones del *input,* lo cual posteriormente permite predecir un atributo que inicialmente estaba en cierta zona de la imagen de entrada, en otras zonas o bajo otras traslaciones. De esta manera, es que se disminuyen considerablemente la cantidad de parámetros respecto a otras redes de DL, lo cual simplifica el proceso de entrenamiento y acelera a la red (Alzubaidi et al., 2021).

Como se ha mencionado, el modelo U-Net (Ronneberger et al., 2015) utiliza las operaciones convolucionales para el aprendizaje de las clases introducidas, y más aún, presenta mejoras en cuanto a los grandes requerimientos de datos para el aprendizaje de otros modelos de DL, optimizando el uso de la data además de permitir un aprendizaje del contexto y de la localización de las clases a identificar. Esto se explica en gran medida debido a que la gran cantidad de canales que posee el camino de expansión o de *upsampling*, logra mantener una buena eficiencia en el reconocimiento contextual de cada pixel; y a los canales concatenados desde el camino descendente o *downsampling* al camino de expansión, que se combinan con estos con el fin de permitir que se pueda localizar cada pixel con mayor precisión, restaurando la información de los bordes. Complementando esto, el camino descendente igualmente contribuye a la captación de información contextual en cada operación.

Su arquitectura que se asemeja a la letra U, desde donde se le nombra como tal. La estructura original se ve en la Figura 1.3. Cabe señalar que todas las imágenes que ingresan al modelo deben tener las mismas dimensiones, para lo cual en muchos casos se utiliza relleno de ceros en todos los canales.

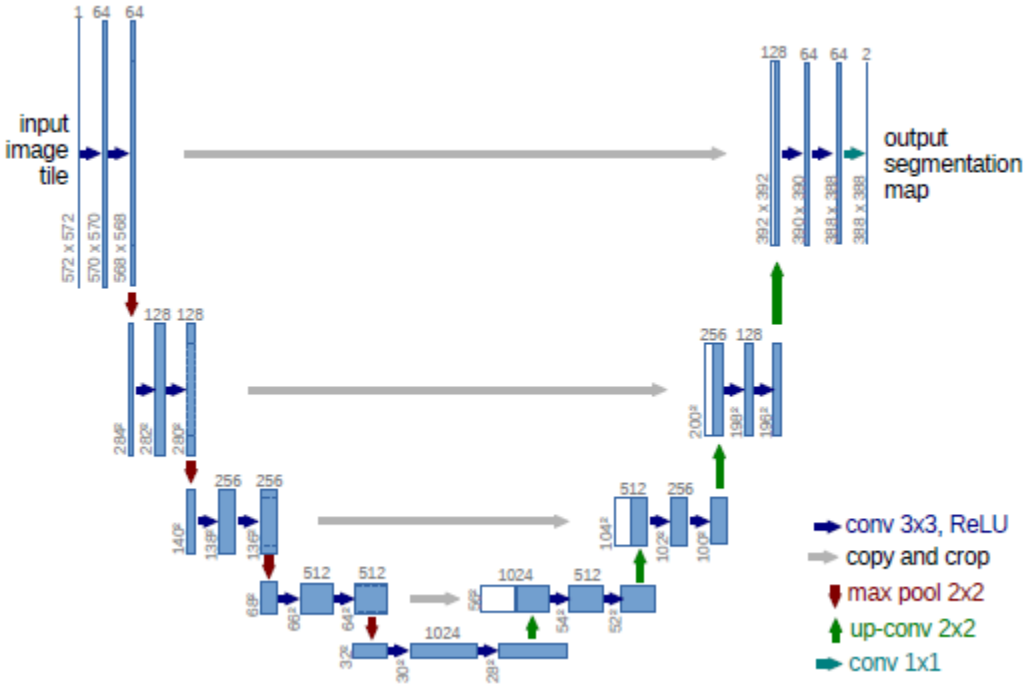


Figura 1.3 Arquitectura Modelo U-Net. (Ronneberger et al., 2015)

Ahora se detallarán cada una de las operaciones que ocurren en esta red además de la convolución:

1.2.1 ReLU

La *Rectified Linear Activation Function* (ReLU), en español la Función de activación linear rectificada, es la capa de activación de la red. Se define su función en la Ec. 1.11. Esta función permite que el modelo pueda aproximar funciones más complejas, representando relaciones no lineales, desde lo cual se recomienda su uso para las CNN.

$$g(x) = máx[0, x] \tag{1.11}$$

1.2.2 Copy and crop

Esta operación consta de copiar y pegar el bloque correspondiente del camino de codificación en el de decodificación, por medio de una concatenación. En Ronneberger et al (2015), se utiliza para recuperar la información de los bordes de la imagen, debido a la reducción del tamaño de las capas producto de las convoluciones sin utilizar *padding*. Por este mismo motivo es que también requiere de un corte, para tener las mismas dimensiones que la data del camino de expansión.

1.2.3 Max pool 2x2

Reduce los *feature maps* obtenidos de las convoluciones en sub-muestras. Específicamente se obtienen los valores máximos de cada uno de los 4 cuadrantes y son ellos los que se transfieren al siguiente bloque convolucional, disminuyendo a la mitad la dimensión de entrada, aplicándose a cada mapa de salida separadamente. Esto permite que el sistema tenga invarianza a pequeñas traslaciones en la entrada, acentuando las activaciones más importantes. En la Figura 1.4 se ve un ejemplo de esta operación.

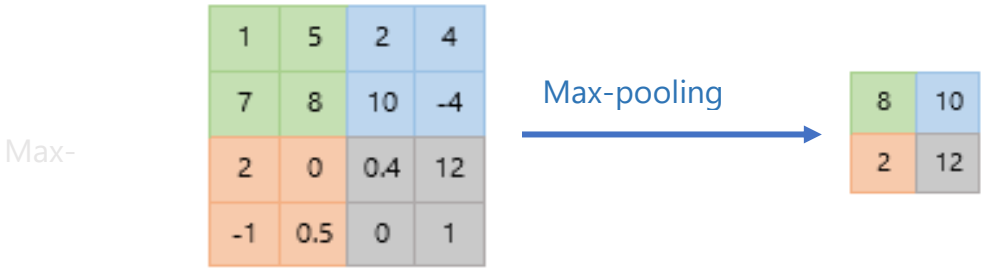


Figura 1.4 Max-pooling 2x2

1.2.4 Up-sampling y up-conv

La up-conv o convolución transpuesta consta de reordenamientos de los kernel respectivos a matrices de mayores dimensiones, rellenadas con ceros, para así poder recuperar los valores originales previo a que se aplicara una convolución.

Alternativamente, en el camino de expansión se pueden usar convoluciones regulares acompañadas de métodos de interpolación para aumentar las dimensiones de las salidas, pudiendo ser de tipo linear, bilinear, entre otros; las cuales no conllevan uso de parámetros a diferencia de las convoluciones transpuestas. En el modelo U Net original, se utiliza la convolución transpuesta combinada con el *copy and crop* para restaurar los bordes de la imagen como se mencionaba previamente, lo cual a su vez logra una mayor complejidad de predicción.

1.2.5 Conv 1x1

Esta operación utiliza un kernel con las dimensiones de alto y ancho unitarias, pero con la misma cantidad de canales de la data de entrada, obteniendo un solo *outut feature map*, que en puede dar paso a un *output* binario dependiendo de la aplicación, como en este caso sería: pixel quemado o no quemado. Cabe destacar que en esta operación se relacionan todos los canales de la capa previa con el *output*.

1.2.6 Batch Norm

Esta función se usa en casos para acelerar el entrenamiento, reduciendo las variaciones que ocurren internamente entre las capas, aplicándose en pequeños lotes, e incluso permitiendo al modelo en algunos casos mejorar su rendimiento notoriamente (Ioffe & Szegedy, 2015).

1.2.7 Función de optimización

Estas funciones cumplen el objetivo de ajustar los pesos para disminuir el error, que es la diferencia entre la data de referencia y el output de la misma data de entrada, los cuales se establecen para cada neurona según el error de la época respectiva y la derivada de la función de activación de esa época. Se describirán dos funciones que son de las más importantes:

- *Stochastic Gradient Descent (SGD) (Bottou, 2010)*, o traducido, el gradiente estocástico descendente, es un optimizador que actualiza los parámetros en cada época de entrenamiento. Debido a estas actualizaciones constantes, se nota el ruido en la obtención del rendimiento óptimo, convirtiendo la convergencia inestable. Para realizar esto, los algoritmos basados en gradientes descendientes, calculan la función objetivo aplicando derivadas de primer orden a los parámetros, para luego actualizar los pesos con *back-propagation*, o retro-propagación, propagando el gradiente de cada neurona a todas sus antecesoras de la capa previa. En las Ec. 1.12 y 1.13 se ven sus ecuaciones para la actualización de sus pesos, donde ${w_{ij}}^{t}$ es el peso actualizado, ${w_{ij}}^{t-1}$ el peso previo, η es el *learning rate*, E el error de predicción y λ el momentum, que sirve para agilizar el proceso de convergencia, sin embargo, un valor muy alto de este puede llevar a saltarse o perder el mínimo global.

$$ {w_{ij}}^{t} = {w_{ij}}^{t-1} - \Delta {w_{ij}}^{t} \tag{1.12} $$

$$ \Delta {w_{ij}}^{t} = \left(\eta * \frac{\partial E}{\partial w_{ij}} \right) + \lambda * \Delta {w_{ij}}^{t-1} \tag{1.13} $$

- Optimizador de *Adaptive Moment Estimation* (Adam) (Kingma & Ba, 2014), que significa la estimación del momento adaptada. Utiliza la optimización de gradientes de funciones objetivo estocásticas, con bajo uso de memoria y de gasto computacional. Utiliza un *learning rate* adaptativo o escalado para cada parámetro del modelo.

1.2.7.1 Función de pérdida

Para problemas de clasificación binaria se suele recurrir a la función de pérdida *Binary Crossentropy loss* (BCELoss). La cual se define en la Ec. 1.14, donde t es el valor del *label* (0 o 1) y p el de la predicción.

$$ BCE(t,p) = -(t * \log(p) + (1-t) * \log(1-p)) \tag{1.14} $$

Existen otras variaciones de esta función, como lo es la BCEWithlogitLoss de Pytorch (Pytorch, 2022), que combina capas sigmoideas con la BCELoss, aplicándole esta función a los valores de p.

1.2.8 Data para el entrenamiento y validación

Los inputs de entrenamiento al modelo que se utilice son fundamentales para obtener un resultado óptimo de este. A continuación, se enuncian algunas de las variables que se pueden estudiar y controlar de la data a seleccionar:

i. Ubicación de las imágenes

ii. Homogeneidad o heterogeneidad

iii. Temporalidad en que se obtienen las imágenes luego de la fecha de incendio y recuperación de la vegetación luego del incendio

iv. Capas de cobertura presentes en las imágenes y carga de combustible en ellas.

v. Severidad de incendio

vi. Rango de variación ambiental (tipo de vegetación y topografía)

vii. Presencia de nubes

viii. Resolución de las imágenes y corrección

ix. Proporción y número de imágenes de entrenamiento asociadas a cada una de las variables previas

Dada su gran complejidad, y debido a que ya se contaba con el dataset de imágenes Landsat disponible y el área de estudio estaba definida, en este trabajo se utilizó principalmente la proporción de área quemada como la variable de orden del dataset, no obstante, se realizaron dos dataset para poder analizar efectos la influencia del contexto en las predicciones y su aplicabilidad (Véase el Capítulo 2 Datos y Métodos).

Respecto a la cantidad de imágenes, en otras investigaciones de ML convencionales se ha indicado que incrementar el número de imágenes de entrenamiento debería mejorar la precisión del modelo hasta que esta se convierte en una asíntota, es decir, con una precisión constante (Millard & Richardson, 2015). Además de esto, se ha señalado que el número de datos de entrenamiento óptimo y de condiciones ambientales, como regiones o paisajes o cantidad de incendios es un factor muy relevante para una clasificación óptima (Collins et al., 2020; Mellor et al., 2015; Millard & Richardson, 2015).

Otras consideraciones para un entrenamiento idóneo son integrar incendios que estén fragmentados, imágenes sin incendios, e incluir incendios con mayor y menor área quemada. Además de esto, si se requiere que se identifique un aspecto en particular, se requerirá de más imágenes de entrenamiento para que se aprenda a identificar un incendio del tipo específico. De esta manera, por ejemplo, para ayudar a que no se identifiquen áreas de cultivo como área quemada, se deben ingresar estas imágenes para que el algoritmo aprenda de ellas y las catalogue como no quemadas. Por otro lado, se deben reconocer los problemas más comunes para identificación de pixeles quemados, tales como los que se ubican en superficies

de agua que se seca, y entrenar al modelo con imágenes que tengan estos cambios, para que este aprenda que estos cambios no significan que exista superficie quemada. Finalmente, los datos utilizados para la validación deben igualmente ser escogidos equitativamente entre las clases, para que así los pesos se actualicen reflejando el balance del dataset. Un ejemplo de esto se puede ve en Langford et al. (2018).

Además de esto, se concluyó en un estudio sobre los parámetros de entrenamiento para modelos de mapeo de severidad de incendios que es importante para tener una mejor clasificación tener un mínimo de 300 puntos de muestra de cada severidad de incendio de al menos 10 incendios independientes (Collins et al., 2020). En este estudio se usó el modelo RF.

Por otra parte, es necesario mencionar los parámetros más relevantes para determinar un buen resultado de clasificación de un modelo CNN. Inicialmente, un factor clave es escoger las imágenes de entrenamiento con mucho escrutinio como se ha mencionado, mientras que otros factores son:

1. Método de normalización
2. Cantidad de capas
3. Parámetros de ingreso

Se puede también destacar que algunos modelos ocupan máscaras que detectan qué áreas pueden ser quemables, según las clasificaciones de uso de suelo identificadas por distintos productos. Por ejemplo, el Land Cover CCI v2.0.7 (LC_CCI) ha sido usado para este propósito en un modelo espacio-temporal para la obtención de cicatrices incendios usando incendios activos (Lizundia-Loiola et al., 2020).

1.2.8.1 Aumentación de datos, batch size y épocas

Se ha comprobado la importancia de la aumentación de datos entrenando modelos de DL cuando existen datos limitados, mejorando la generalización y disminuyendo los sobreajustes (Perez & Wang, 2017). Dentro de las transformaciones más usadas para aumentar la data se encuentran:

a. Variaciones de colores, brillo y contraste.
b. Sobreposición de parches de tamaños aleatorios
c. Erradicaciones aleatorias como cortes
d. Giros y rotaciones
e. Traslaciones
f. Adición de ruido

El uso de imágenes de otras investigaciones que han empleado U-Net para el mapeo de áreas quemadas ha sido variable, pero en todas ellas han utilizado en gran medida la aumentación de datos para aprovechar al máximo los *inpust* con el fin de garantizar el mayor aprendizaje

de los modelos. En la Tabla 1.5 se presenta un resumen de los inputs ingresados en número de imágenes y la cantidad final de fragmentos resultantes de la aumentación.

Tabla 1.5 Input y aumentación en modelos U-Net

Fuente	*Aplicación*	*Input inicial*	*Fragmentos después de cortes y aumentación*	*Epochs*	*Batch Size*
(Shamsoshoara et al., 2021)	Mapeo áreas quemadas	500	5137	30 máx[2]	16
(Hu et al., 2021)	Mapeo áreas quemadas	6 incendios de distintas dimensiones recortados a 256x256	2034 (S2)	350[1]	8
(Knopp et al., 2020)	Mapeo áreas quemadas	184 para el dataset de referencia (usado para training, validación y testing)	6272	20 máx	16
(Wei et al., 2019)	Distintas coberturas agrícolas	4 secciones 1000x1000	7840x224x224 (224x224x6)	10	5

Nota: 1: En esta investigación se vio que el modelo U Net tuvo un peak en 350 épocas, sin embargo, se realizó la comparación de performance entre varios modelos a las 50 épocas. 2: Máx. hace referencia a que se utilizó detención temprana en el caso de no mejorar las métricas.

Además de lo anterior, se deben destacar dos puntos: Primero, los lotes deben tener data representativa de los modelos, por lo que el tamaño de estos está directamente relacionado a la complejidad de la data y las características que se requieran de aprender. Segundo, como se comentaba previamente, se debe analizar que durante el entrenamiento la métrica de interés se estabilice aproximadamente en una asíntota para comprobar que la configuración dada al modelo es la correcta, incluyendo al *batch size*, la cantidad de épocas y la aumentación dada. De la misma manera, durante la validación se debe comprobar una curva estable que no demuestre un *overfitting*.

1.2.8.2 Tamaño del input

Es de relevancia mencionar la investigación hecha por de Bem et al (2020), donde se estudió el efecto del tamaño del input de imágenes L8 en el rendimiento de U-Net y otros modelos, obteniendo que el tamaño óptimo para el rendimiento de este fue de 256x256 pixeles. En la

Figura 1.5 se ve el efecto que tiene escoger ciertas dimensiones en la cantidad de *samples* o muestras, donde para a) 512x512, b) 256x256, c) 128x128, y d) 64x64 pixeles se tienen 168, 747, 3140 y 12860 muestras respectivamente, esto con una sobreposición de un 12,5% entre ventanas de muestreo para reducir las pérdidas de predicción en los bordes.

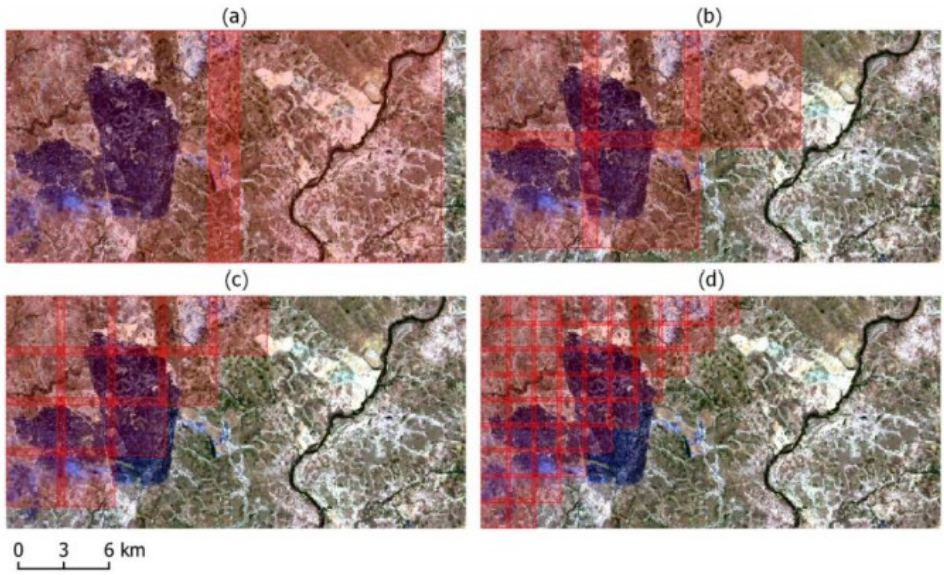


Figura 1.5 Tamaño de input y cantidad de muestras. (de Bem et al., 2020)

1.2.8.3 Normalización

Finalmente, la normalización es muy relevante, tanto para que las distintas bandas no tengan distintos rangos de datos que puedan afectar a los pesos, como para mantener los gradientes asociados a la función de optimización operativos. Dos de las más conocidas y utilizadas, además del Batch Norm ya mencionado, son la estandarización y la normalización Min-Max. La primera deja los datos en el rango [-1,1] con promedio igual a 0 y desviación estándar igual a 1. En la Ec. 1.15 se ve su fórmula, donde Z es la data normalizada, μ el promedio y θ la desviación estándar. La normalización Min-Max deja los valores en rango [0,1], donde X' es la data normalizada, y x la data a normalizar (Ec. 1.16).

$$Z = \frac{x-\mu}{\theta} \tag{1.15}$$

$$X' = \frac{X-\min(x)}{\max(x)-\min(x)} \tag{1.16}$$

Es relevante mencionar que los parámetros para la normalización deben provenir únicamente del dataset de entrenamiento, puesto que se busca predecir a partir únicamente de la información aprendida de esta data.

2 Datos y métodos

2.1 Área de estudio

Inicialmente, desde esta investigación solo se tiene conocimiento sobre un dataset disponible de imágenes que contienen cicatrices de incendios que utiliza satélites Landsat, provisto por el United States Geological Survey (USGS) (https://data.usgs.gov/datacatalog/data/USGS:587017d7e4b01a71ba0c5ff7). De la misma manera, proveen un producto para la estimación de áreas quemadas, sin embargo, posee amplios errores de omisión y comisión, al ser basado en un algoritmo para la predicción de las superficies quemadas. Por esto, se justifica la necesidad de utilizar otro dataset para este trabajo, como lo es el de The Landscape Fire Scars Database (Miranda et al., 2022), que contempla un margen temporal entre el 1985 y el 2018, otorgando una amplia base de datos para el estudio de los incendios de Chile.

Ahora, como se ha mencionado en la sección 1.2, el área de estudio de este trabajo se acota a las regiones de Valparaíso y BíoBio, como se ve en la Figura 2.1, mientras que un ejemplo de la visualización de la data desde GEE se ve en la Figura 2.2.

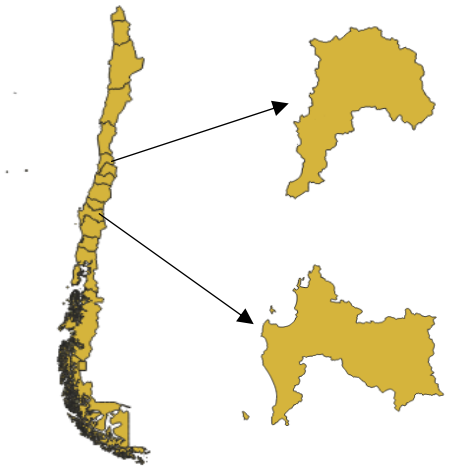


Figura 2.1 Área de estudio. Adaptado de (BCN, 2022)

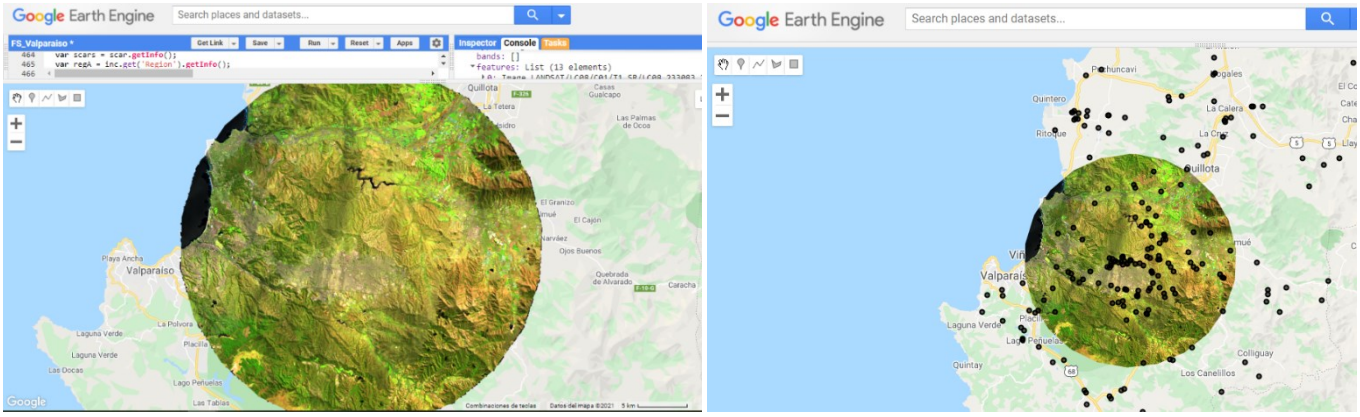


Figura 2.2 Visualización de la data en GEE. (Google, 2021)

Cabe mencionar que esta data original abarcaba normalmente valores sobre los mil pixeles de diámetro, donde se podían incluir múltiples incendios, por lo que era necesario recortarlas a un tamaño uniforme para el ingreso al modelo, lo cual se explica en la siguiente sección 2.2.

2.2 Datasets

En primer lugar, para entender de mejor manera la data y su complejidad, en el Anexo A se pueden ver los incendios característicos más relevantes que componen el dataset original. Adicionalmente, en el Anexo B se muestra el flujo de proceso de decisiones bajo el cual se recopiló la data original.

La data disponible seleccionada para este trabajo desde el dataset inicial comprendía los archivos ráster tanto de las cicatrices de incendio (binarias), como de las imágenes correspondientes pre y post-incendio (8 bandas cada una) de los satélites Landsat 5, 7 y 8, donde ambas se incluyeron para este trabajo, al escoger un enfoque multi temporal por sus beneficios en cuanto a detección de requemas y omisión de incendios antiguos, además de permitir tener más data para el modelo. Junto a esto, se contaba con los archivos vectoriales de los incendios, los cuales luego se usaron para obtener el área real quemada con mayor facilidad con el uso de la librería Geopandas de Python. Para ello y el resto de los análisis, se crearon datasets con toda la información necesaria de los registros de incendios: nombres de archivos, área quemada, tamaño, región, entre otros; con el fin de acceder fácilmente a ellos, desde lo cual se pudo estudiar y manipular la data usando Python, incluyendo sus librerías geoespaciales, como Geopandas, Gdal, Rasterio y Shapely; y otras como Numpy, Pandas y Scikit learn (Pedregosa et al., 2012), entre otras.

Otro factor para la creación de la data era el tamaño que ingresaría al modelo U Net, dado que se requiere de un tamaño fijo idéntico para todas las imágenes de entrada con su data de referencia, pudiendo tener relleno o no. Considerando lo visto en la investigación de Bem et al (2020) respecto al tamaño que mejoraba las métricas, el tamaño en el que sería óptimo

revisar la data para que no contara con otros incendios de la misma temporada, y el peso de almacenamiento para el tratamiento de los datos, se designó el tamaño de entrada como 128x128 pixeles, que corresponden a 3,84 km por eje. A pesar de esto, se debe recordar que la data puede ser igual o inferior a esta dimensión usando relleno, el cual puede consistir en rellenar con ceros por ejemplo (*zero padding*). Desde esto, y conociendo el impacto del tamaño del tamaño de los inputs en el rendimiento del modelo, se decidió formar dos datasets con un distinto tamaño y balance de clases (i.e. razón entre las clases reconocidas y etiquetadas y no etiquetadas, en este caso, área quemada y no quemada), con el objetivo de obtener los mejores resultados posibles, manipulando esta y otras variables para posteriormente seleccionar el mejor enfoque. Estos datasets fueron denominados Allsizes (AS), el cual se compuso por imágenes de distintos tamaños, del mismo tamaño de los

cuadriláteros delimitadores de los archivos ráster de las cicatrices de incendios, y 128, el cual se constó de imágenes recortadas a un tamaño estándar único de 128x128 pixeles, que se extendieron desde los cuadriláteros delimitadores de los archivos ráster hasta este tamaño. En la Figura 2.3 se puede ver a la izquierda la cicatriz original, delimitada tal como se encuentra en el archivo inicial, mientras que a la derecha se ven las imágenes pre y post recortadas para cada uno de los dos datasets. Las imágenes del dataset AS se recortaron al mismo tamaño que la cicatriz original, rellenando el resto de los pixeles con ceros hasta alcanzar el tamaño de 128x128; y las imágenes del dataset 128 se recortaron a un mayor tamaño, de 128x128 pixeles. Estas imágenes fueron recortadas con el código expuesto en el Anexo C.

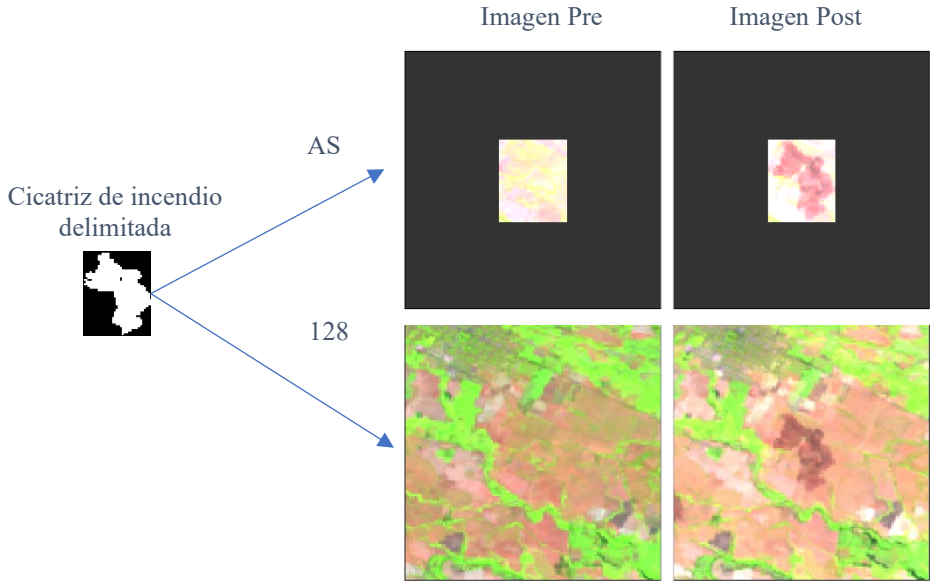


Figura 2.3 Principio de formación de los datasets AS y 128

Respecto a la proporción de entrenamiento y validación para el modelo, se escogió un 80/20%, aparte de contar con un set de testeo independiente de 100 datos.

Se testearon ambos métodos de normalización mencionados (estandarización y normalización Min-Max) de manera preliminar, de los cuales se comentan sus resultados en la sección 3.

A continuación, se caracterizará la data satelital utilizada, describiendo las bandas e índices espectrales utilizados, sus rangos de valores y el preprocesamiento aplicado, y finalmente los datasets finales utilizados elaborados a partir del primer dataset, AS y 128. Agregando a lo anterior, en este capítulo también se describen las metodologías utilizadas para el manejo de la data.

En este capítulo para caracterizar la data se presentan tablas con las columnas: count, mean, min, 25%, 50%, 75% y Max, de las cuales se detallan sus significados en la Tabla 2.1.

Tabla 2.1 Descripción columnas

	Count	mean	std	Min	25%	50%	75%	Max
Desc.	Número de datos asociado al conjunto	Promedio de la variable	Desviación estándar	Valor mín.	Valores correspondientes a los cuartiles respectivos			Valor máx.

Cabe mencionar que previo a haber finalizado los datasets oficiales presentados en esta sección, se utilizaron datasets preliminares para identificar el método de normalización más eficiente, además de comparar los resultados con y sin aumentación (duplicando la data en este caso). Estos datasets preliminares contenían algunos datos que poseían la misma área de quema repetida, lo cual afectó sus métricas marginalmente. Más detalles se presentan en la sección 3.

2.2.1 Satélites y caracterización de la data

La data usada proviene de tres satélites Landsat: LT05, LE07 y LC08 Operational Land Imager (OLI).

Más detalles de la data se pueden encontrar en las guías de producto emitidas por el USGS para el L4-L7 y L8 respectivamente (2020a, 2020b).

En primer lugar, se requieren algunas definiciones, como números digitales (DN), bits y reflectancia, además de definir las bandas e índices espectrales utilizados y los rangos de valores de la data:

2.2.1.1 DN

Los DN se definen como el valor numérico de un pixel específico, normalmente se nombran como tal previo a ser traducidos a unidades físicas. En el caso de las imágenes satelitales, en primer lugar, se convierten los DN a radiancia y esto luego se pasa a DN implementando correcciones atmosféricas y radiométricas.

2.2.1.2 Bits

Es como se define el número de niveles de grises asociados a cada pixel de la imagen, que permite la unidad fundamental de un sistema binarista, característica que significa la cantidad de valores que puede tomar determinado pixel. Por ejemplo, si una imagen tiene 4 bits, el rango de valores será de 2^4=16.

2.2.1.3 Reflectancia de la superficie

Es el valor de la reflectancia en la superficie luego de considerar los efectos de la atmósfera y sus perturbaciones.

2.2.1.4 Bandas espectrales

Cada imagen satelital a utilizar contaba con 8 bandas espectrales, donde las últimas dos de ellas fueron calculadas a partir de combinaciones de las primeras (véanse las Ecs. 2 y 2.1). En la Tabla 2.2 se define cada banda. Posteriormente, la forma de ingresar la data al modelo fue concatenar las imágenes pre-incendio luego de las imágenes post-incendio, resultando cada dato en 16 matrices conteniendo los valores de cada banda.

Tabla 2.2 Bandas data

Banda	
1	Blue
2	Green
3	Red
4	NIR
5	SWIR1
6	SWIR2
7	NDVI
8	NBR

Donde

$$NDVI = \frac{NIR - Red}{NIR + Red} \tag{2}$$

$$NBR = \frac{NIR - SWIR2}{NIR + SWIR2} * 1000 \tag{2.1}$$

El Índice de Vegetación de Diferencia Normalizada (NDVI) es uno de los índices espectrales más conocidos y usados, el cual se basa en la gran absorción de la luz roja desde la vegetación, en contraste con la alta reflexión de esta ante la luz infrarroja corta (NIR) (Fornacca et al., 2018). Debido a esto, los daños en la vegetación ocasionados por quemas generan una disminución notoria de este índice. Más específico, el NBR se usa frecuentemente para la evaluación de severidad de incendios debido a su sensibilidad específica a los cambios ocurridos con incendios, incluyendo su uso para incendios ocurridos hace años, permitiendo la supervisión de la recuperación de la vegetación. De esta manera,

se ha visto que obtiene los mejores rendimientos entre otros índices para identificación de áreas quemadas con períodos de tiempo superiores a un año, aunque otros lo superan para tiempos inferiores (Fornacca et al., 2018).

2.2.1.5 RdNBR

En la recolección de la base de datos usada, para la identificación de pixeles quemados se utilizó el índice *Relative Delta Normalized Burned Ratio* (RdNBR) (Miller & Thode, 2006), que se traduce como la diferencia relativa de la razón de quema normalizada, que cumple el objetivo de reconocer pixeles quemados usando las bandas espectrales que miden la humedad y vegetación. Se define en la Ec. 2.2.

$$RdNBR = \frac{NBR_{pre\ incendio} - NBR_{post\ incendio}}{\sqrt{\left|\frac{NBR_{pre\ incendio}}{1000}\right|}} \tag{2.2}$$

2.2.1.6 Rango de valores de la data y preprocesamiento

Para toda la data se comparten los rangos de data que se presentan en la Tabla 2.3.

Tabla 2.3 Rango de valores de la data

Banda	Rango	Rango válido	Valor de llenado	Valor saturado
1-7	-2000-16000	0-10000	-9999	20000

Fuente: Adaptado de (U.S. Geological Survey, 2020a, 2020b)

En primer lugar, los valores saturados de 20.000 presentes en la data se usan para identificar pixeles inservibles, fuera del rango de los pixeles válidos. Ocurren cuando hay altos niveles de radiancia o alta energía de partículas, implicando que los DN alcancen su máximo valor. Esto puede ocurrir por ejemplo en nubes, incendios, flujos de lava, nieve y hielos (Morfitt et al., 2015).

Es relevante mencionar que la data de los satélites L5 y L7, desde ahora L57, tienen una principal diferencia con la data del L8, que es la cantidad de bits que poseen, donde los primeros cuentan con 8 bits y el último con 16 bits, por lo que L57 pueden tener hasta 256 distintos valores por pixel, mientras que L8 hasta 65536. Esto ha llevado a que raramente se tengan valores saturados en la data del L8 (Morfitt et al., 2015; U.S. Geological Survey, 2020b), mientras que sí ocurre para la data de L57.

Por este motivo, se procedió a reemplazar los valores fuera de rango con un preprocesamiento. Inicialmente, se debieron cambiar los valores inválidos conocidos como NaN (*not a number*) que se encontraban en la data, los cuales se interpolaron con la función *k nearest neighbors* de la librería Scikit Learn de Python (Pedregosa et al., 2012), la cual aproxima los valores deseados según la cercanía a los otros puntos dentro de un radio de búsqueda. Se pueden ver algunos ejemplos de la actuación de esta interpolación en el Anexo A.

Posteriormente, como segunda parte del preprocesamiento se utilizó una imputación de datos, que consiste en agregar datos de un valor arbitrario para reemplazar otros. Esto se hizo usando una máscara para los valores fuera de rango, primero para los superiores a 10.000 para reemplazarse por los promedios de cada banda de la misma imagen, y de la misma manera, con los valores inferiores a 0 para las bandas 1-6 presentadas. En las bandas 7-8, al poseer rangos de -1 a 1 y -1000 a 1000 respectivamente, se reemplazaron los datos fuera de los intervalos por el promedio de la banda respectiva igualmente.

Para la normalización Min-Max, se debieron calcular los valores mínimos y máximos del set de entrenamiento, mientras que para la estandarización, se debieron calcular los promedios y desviaciones estándar. Para esto se usaron los mismos límites del rango válido de la data, no se consideraron los otros datos inválidos NaN, y se filtraron todos los datos anómalos o *outliers* de la distribución en sí. Para el análisis de estos datos anómalos se pueden buscar los datos anómalos leves, que se consideran al distanciarse del primer o tercer cuartil en al menos 1,5 veces la distancia entre estos cuartiles, o como se define, rango intercuartil (IQR); o bien, los datos anómalos extremos, que se definen por distanciarse al menos 3 veces el IQR. De esta manera, se establece el Límite Inferior (LI) y Límite Superior (LS). Se optó en este caso por el estudio de anómalos leves para el preprocesamiento. Se ven sus expresiones en las Ecs. 2.3 y 2.4.

$$LI = Q1 - 1.5 \cdot RIQ \tag{2.3}$$

$$LS = Q3 + 1.5 \cdot RIQ \tag{2.4}$$

En esta sección se le da énfasis a la normalización Min-Max debido a su mayor relevancia (véanse los resultados preliminares de la sección 3).

2.2.1.7 Estratificación de tamaños

Debido a las distribuciones de tamaños muy variadas para ambos dataset (AS y 128), y a la necesidad de que los lotes incluyeran data representativa del modelo, se estratificó la data de cada dataset según sus cuartiles en porcentaje de área quemada (se realizó separadamente para los dataset AS y 128 debido a la distinta cantidad de datos entre estos), desde lo cual se compuso un nuevo dataset alternando cada registro con un dato de cada uno de estos cuartiles (pequeño, mediano, grande y muy grande). Esto incluyó a la data de entrenamiento, validación y testeo, las cuales posteriormente se separaron.

2.2.1.8 Testing

Para la validación del modelo, es fundamental un dataset de testeo. En este caso, se formó un dataset de 100 datos, representativos de BioBío (52 datos) y Valparaíso (48 datos), el cual se diseñó para tener los mismos registros de evaluación para ambos modelos, pero con los tamaños correspondientes a cada dataset (i.e. datos 128x128 para el modelo 128 y usando los mismos tamaños de las cicatrices de incendios para el modelo AS).

En la Tabla 2.4 se describen los rasgos generales en cuanto a tamaño y proporción de área quemada de este dataset, donde "% área quemada" corresponde al porcentaje calculado desde el área real respecto a los 128x128 pixeles que es el tamaño de ingreso al modelo; "Y" y "X" son los pixeles que poseía la cicatriz de incendio en su cuadrilátero delimitador, y "área quem. ha" corresponde al área quemada total en hectáreas.

Se puede notar que hay una gran desviación estándar en cuanto al área quemada en hectáreas, desde lo que se justifica la estratificación por tamaños y su inclusión en cada lote de ingreso al modelo. Otro punto importante a considerar, es que el porcentaje de área quemada ingresado al modelo es en promedio de un 4% del tamaño de entrada, siendo bajo en cuanto al área total, lo cual es un posible factor de errores.

Tabla 2.4 Descripción del dataset de test general

	count	Mean	Std	Min	25%	50%	75%	Max
Y	100	41,2	27,49	5	22	33	52	126
X		46,0	30,34	3	22,5	37	59,25	117
área quem., ha		59,7	71,37	0,50	14,1	29,8	72,6	349,8
% área quemada		4,0	4,84	0,034	1,0	2,0	4,9	23,7

2.2.2 Dataset All Sizes

Para la formación de este dataset, incluyendo *training*, *validation* y *testing*, se contó con 2635 datos iniciales, constituidos por 1156 datos de la Región de Valparaíso y 1479 de la Región de BioBío.

De esta manera, se analizó la distribución de los datos de los datasets de entrenamiento y validación, como se presenta en las Tablas 2.5 y 2.6. De ellos se nota nuevamente una tendencia similar a la del dataset de testing, con una gran desviación estándar en cuanto al área quemada, y con un promedio cercano igualmente, con 51,4 y 47,9 hectáreas para los dataset de entrenamiento y validación respectivamente.

Tabla 2.5 Análisis estadístico de tamaños, training AS

	Count*	Mean	Std	Min	25%	50%	75%	Max
Y	2029	37,4	24,97	2	19	30	48	127
X		41,4	25,73	3	22	35	54	128
área quem., ha		51,4	63,53	0,494	12,8	26,4	61,3	466,2
% área quemada		3,49	4,30	0,034	0,87	1,79	4,2	31,6

Tabla 2.6 Análisis estadístico de tamaños, validation AS

	Count	Mean	Std	Min	25%	50%	75%	Max
Y	506	35,0	21,72	4	20	28	45	121
X		40,8	24,83	5	21	34	55,75	124

área quem., ha	47,9	55,70	1,2	13,6	26,6	62,0	471,7
% área quemada	3,3	3,77	0,087	0,93	1,8	4,2	31,9

2.2.2.1 Metodología 1

Esta metodología buscaba cubrir la porción más importante de la data, que es la superficie quemada, por lo que se tomaron los archivos ráster de las cicatrices de incendios existentes para recortar las imágenes post y pre-incendio con sus 8 bandas a ese tamaño, el cual era muy variable como se vio en la sección previa 2.2.2, pero cercano a las 50 hectáreas quemadas en promedio por imagen. Algunos criterios a destacar son:

i. Data inferior o igual a 128x128 pixeles con posterior *zero padding*
   Inicialmente, se escogió este tamaño debido a su simpleza para supervisar las imágenes, además de lo observado respecto a la obtención de mejores resultados con un tamaño intermedio (véase la sección 1.1.10.2). De un total de 3075 datos iniciales, se filtró un 12% de la data que era superior a este tamaño (128x128). Posteriormente todos los archivos se sometieron a *zero padding* hasta alcanzar el tamaño de 128x128 pixeles.
ii. Equilibrio relativo entre las regiones de la data
   Como se indicó previamente, el dataset de entrenamiento se compone por 2029 datos, con 1144 datos de la Región de BioBío (56%) y 885 de Valparaíso (44%). Se mantiene la misma relación para el dataset de validación, que corresponde al 20%, y finalmente en el testeo se tiene un 52% vs un 48% entre BioBío y Valparaíso.
iii. Distribución de porcentajes de quema
   De acuerdo a lo comentado en la sección 2.2.1.7, se distribuyeron los porcentajes de área quemada de forma que todos los lotes fueran representativos.

Esta metodología apunta a data que provenga de cuadriláteros delimitadores, lo cual en un futuro puede realizarse automáticamente con una correcta geolocalización de los incendios y sus bordes.

2.2.2.2 Análisis estadístico

Se realiza este análisis inicialmente para los 2535 datos correspondientes al dataset de entrenamiento y validación, que requieren de mayor estudio al ser sometidos a métodos de normalización. Por lo mismo, en la Tabla 2.7, se presenta la distribución de datos de los promedios de las bandas incluyendo sus desviaciones estándar, mínimos y máximos, y sus valores por cuartiles.

Tabla 2.7 Estadística preliminar de los datos AS, sin preprocessing

Banda	Count	Mean	Std	Min	25%	50%	75%	Max
1	2535	406	134	-181	314	406	491	1203
2		595	167	3	481	592	701	1364
3		713	225	104	549	708	858	1630

4		1651	341	455	1427	1637	1861	3231
5		1714	443	399	1394	1719	2027	3384
6		1259	384	225	988	1278	1535	5191
7		0,484	2,475	0,145	0,286	0,362	0,465	106,8
8		137	172	-278	8	104	246	733
9		422	158	38	309	419	526	1080
10		669	216	151	513	659	808	1650
11		798	330	84	558	770	998	2317
12		2243	457	703	1938	2215	2521	4481
13		1937	575	473	1523	1927	2368	3590
14		1173	413	177	867	1171	1487	2425
15		0,488	0,190	-4,52	0,372	0,469	0,590	3,78
16		334	180	-105	193	318	463	859

En la Tabla 2.8 se exhiben los datos para el preprocesamiento y la normalización, mientras que la Tabla 2.9 muestra la misma data previa pero habiendo pasado por el preprocesamiento, lo cual se evidencia en que todos los valores se encuentran dentro del rango válido.

Tabla 2.8 Límites y parámetros para preprocesamiento y normalización Min-Max AS

Banda	Mín.	LI	Máx.	LS
1	0	0	1689	1689
2	0	0	2502	2502
3	0	0	3260	3260
4	17	0	5650	5650
5	7	0	5282	5282
6	0	0	4121	4121
7	-0,0962	-0,0964	1	1
8	-598,0	-598,0	1000	1000
9	0	0	1750	1750
10	0	0	2559	2559
11	0	0	3325	3325
12	0	0	6065	6065
13	8	0	5224	5224
14	0	0	3903	3903
15	-0,0966	-0,0966	1	1
16	-392,0	-392,7	1000	1000

Tabla 2.9 Descripción promedios bandas incluyendo preprocessing AS

Banda	Count	Mean	Std	Min	25%	50%	75%	Max
1	2535	405	132	55	314	406	490	1054
2		594	166	59	481	592	701	1203
3		713	225	155	549	708	858	1630
4		1650	341	455	1427	1637	1861	3231
5		1714	443	399	1394	1719	2027	3384
6		1257	376	225	988	1277	1535	2374
7		0,385	0,128	-0,065	0,285	0,361	0,463	0,805
8		136	172	-278	8	104	246	733

9	422	157	46	309	419	525	1080
10	669	215	151	513	658	808	1650
11	797	329	84	559	769	998	2317
12	2243	457	703	1938	2215	2521	4481
13	1937	575	473	1523	1927	2369	3590
14	1172	413	177	867	1170	1487	2425
15	0,487	0,180	-4,775	0,373	0,470	0,590	0,930
16	334	180	-105	193	318	463	859

Finalmente, se presentan los valores promedio de las bandas con normalización en la Tabla 2.10.

Tabla 2.10 Promedio bandas post Min-Max AS

Banda	Valor
1	0,240
2	0,237
3	0,219
4	0,290
5	0,324
6	0,305
7	0,439
8	0,460
9	0,241
10	0,261
11	0,240
12	0,370
13	0,370
14	0,300
15	0,532
16	0,522

2.2.3 Dataset 128

En primer lugar, este dataset se compone de 2173 imágenes, donde 1032 datos corresponden a la Región de Valparaíso, y 1141 a la Región del Bío-Bío. Desde aquí, 2073 datos se destinan para el aprendizaje en el entrenamiento y validación del modelo y 100 datos se reservan para el testeo, los últimos divididos equitativamente para la Región de Valparaíso y BíoBio, que son los mismos registros usados para probar la Metodología 1, pero de tamaño 128x128 pixeles, incluyendo valores de imagen en todo aquel cuadrante, a diferencia del caso anterior en que se utiliza relleno con ceros, mientras que el tamaño es igual al del archivo ráster cuadrilátero donde se contenía la cicatriz.

Al igual que para el dataset anterior, se revisó la distribución de tamaños, con el objetivo de mantener balanceado el aprendizaje y testeo del modelo. En las Tablas 2.11 y 2.12 se presentan las distribuciones de los tamaños. Para poder comparar con los datos del modelo AS se usaron los valores de las cicatrices de incendios y no de las imágenes, las cuales son siempre de tamaño 128x128 en este dataset.

Tabla 2.11 Análisis estadístico de tamaños, training 128

	count	Mean	Std	Min	25%	50%	75%	Max
X	1620	36,8	24,10	2	19	29	47	127
Y		41,8	25,47	3	22	35	55	128
área quem., ha		52,1	63,0	0,494	13,6	27,1	63,7	471,8
% área quemada		3,5	4,3	0,0335	0,9	1,8	4,3	32,0

Tabla 2.12 Análisis estadístico de tamaños, validación 128

	count	Mean	Std	Min	25%	50%	75%	Max
X	404	39,2	26,34	6	19	31	50	124
Y		41,0	25,82	7	22	35	51,25	127
área quem., ha		54,1	66,75	1,88	14,0	27,1	61,9	384,1
% área quemada		3,7	4,53	0,127	0,9	1,8	4,2	26,1

2.2.3.1 Metodología II

Este dataset precisó de mayor revisión, puesto que no se podía permitir la existencia de más incendios en las imágenes que no estuvieran reconocidos en la data de referencia, desde lo cual se revisó toda la data. Algunos puntos relevantes:

i. Data igual a 128x128 pixeles con *zero padding en cicatrices*
   Toda la data tiene dimensiones 128x128, y solo las cicatrices de incendios pasan por el relleno con ceros.
ii. Equilibrio relativo entre las regiones de la data
   Se incluyen 840 datos de BioBío (52%) y 780 de Valparaíso (48%) en el entrenamiento, y se mantienen las proporciones mencionadas, con 20% de la data total para validación, y 52% de BioBío vs 48% de Valparaíso en el testing.
iii. Distribución de porcentajes de quema
   Al igual que en el caso anterior, se alternaron por cuartiles para mantener los lotes representativos.
iv. Aprendizaje de contexto y quemas agrícolas
   Si bien es un criterio difícil de diferenciar, tanto para humanos como para modelos de IA, al ingresar data que tenga ambas clases, incendios y quemas agrícolas, con solo una de ellas reconocida como área quemada, se puede esperar que el modelo aprenda ciertos atributos relacionados para no identificar en este caso las quemas agrícolas como incendios. Este punto se considera como adicional, puesto que para cumplir esta tarea se requiere de mayor cantidad de data y de mayor revisión, por lo que no es un objetivo principal.

Este modelo, al igual que el modelo AS, podría funcionar automáticamente recibiendo imágenes con los cuadriláteros delimitadores de los incendios, que podrían ser a su vez obtenidos de forma automática con una correcta geolocalización de los bordes del evento. Sin embargo, este enfoque aprende más información contextual, por lo que puede recibir

imágenes de mayor tamaño, como lo son 128x128 pixeles y reconocer los incendios, desde donde podría ser más útil desde este lado.

2.2.3.2 Análisis estadístico

En la Tabla 2.13 se presentan los valores sin filtrar los datos anómalos ni valores fuera de rango.

Tabla 2.13 Estadística preliminar de los datos 128, sin preprocessing

Banda	Count	Mean	Std	Min	25%	50%	75%	Max
1	2024	417	153	-158	312	415	515	2181
2		644	195	-3	500	640	773	2290
3		757	271	33	558	755	955	2400
4		2059	437	455	1778	2059	2347	3420
5		1811	500	395	1450	1827	2175	3256
6		1162	380	192	883	1186	1438	2701
7		1,03	4,38	0,0321	0,350	0,439	0,579	91,5
8		300	174	-60	158	279	429	819
9		417	149	34	311	417	511	1205
10		656	190	191	522	655	787	1590
11		760	265	132	564	755	949	1806
12		2193	423	719	1916	2198	2471	3643
13		1829	487	455	1481	1829	2177	3241
14		1137	367	203	864	1148	1402	2319
15		0,567	2,89	-3,48	0,375	0,464	0,579	104,6
16		336	165	-33	202	327	463	766

En la Tabla 2.14 se presentan los parámetros para el preprocesamiento y normalización, desde donde se calculan los datos de la Tabla 2.15 sin poseer datos anómalos.

Tabla 2.14 Límites y parámetros para preprocesamiento y normalización Min-Max 128

Banda	Mín.	LI	Máx.	LS
1	0	0	3560	3561
2	0	0	4147	4149
3	0	0	4780	4781
4	0	0	6832	6833
5	0	0	6143	6144
6	0	0	6261	6263
7	-0,674	-0,674	1	1
9	-605	-605	1000	1000
10	0	0	3765	3766
11	0	0	4398	4399
12	0	0	4866	4868
13	0	0	6930	6930
14	0	0	6157	6159
15	0	0	6167	6169
16	-0,771	-0,772	1	1

17	-572	-574	1000	1000

Tabla 2.15 Descripción promedios bandas incluyendo preprocessing 128

Banda	Count	Mean	Std	Min	25%	50%	75%	Max
1	2024	416	147	60	312	415	514	1271
2		643	192	164	502	639	773	1604
3		756	269	113	558	754	953	1897
4		2059	437	455	1778	2059	2347	3420
5		1812	498	431	1450	1827	2175	3256
6		1163	378	192	883	1186	1439	2636
7		0,455	0,146	-0,0427	0,346	0,433	0,555	0,865
8		298	173	-60	158	278	424	765
9		417	149	54	311	417	510	1205
10		656	190	191	522	655	787	1590
11		760	265	132	564	755	951	1806
12		2193	424	719	1916	2198	2470	3643
13		1829	486	455	1482	1829	2177	3241
14		1137	367	203	864	1149	1402	2319
15		0,473	0,210	-5,30	0,374	0,463	0,577	0,866
16		336	165	-41	202	326	462	766

Finalmente, en la Tabla 2.16 se ven los datos post Min-Max.

Tabla 2.16 Promedio bandas post Min-Max 128

Banda	Valor
1	0,117
2	0,155
3	0,158
4	0,301
5	0,295
6	0,186
7	0,674
8	0,563
9	0,111
10	0,149
11	0,156
12	0,317
13	0,297
14	0,184
15	0,703
16	0,578

2.3 Modelo

El modelo utilizado tuvo una base en el código de Mommert et al. (2020), el cual se basó a su vez en el de Milesi, (2017) para la red U Net usando Pytorch como marco de trabajo de Python, especializado en aplicaciones de DL.

Se destaca que la función de pérdida es la BCEWithLogitLoss (véase la sección 1.2.7.1).

El código utilizado se presenta en el Anexo C.

2.3.1 Arquitectura

El modelo U-Net, posee una arquitectura que se asemeja a la letra U, desde donde se le nombra como tal. La estructura original se presentó en la sección 1.3, en la Figura 1.3, mientras que la variación usada para este trabajo se exhibe en la Figura 2.4.

Algunos detalles diferentes a destacar respecto a la red original son el uso de *Batch Norm*alization luego de cada convolución 3x3, el uso de *padding* en estas para no disminuir el alto y ancho, y el uso de *upsampling 2x2*, el cual consta de interpolaciones bilineares en el camino de expansión, respecto del uso de convoluciones transpuestas como se usaba en la red original. Finalmente, tampoco hay capas ocultas intermedias en el último bloque del camino descendente, manteniendo el mismo número de filtros (1024).

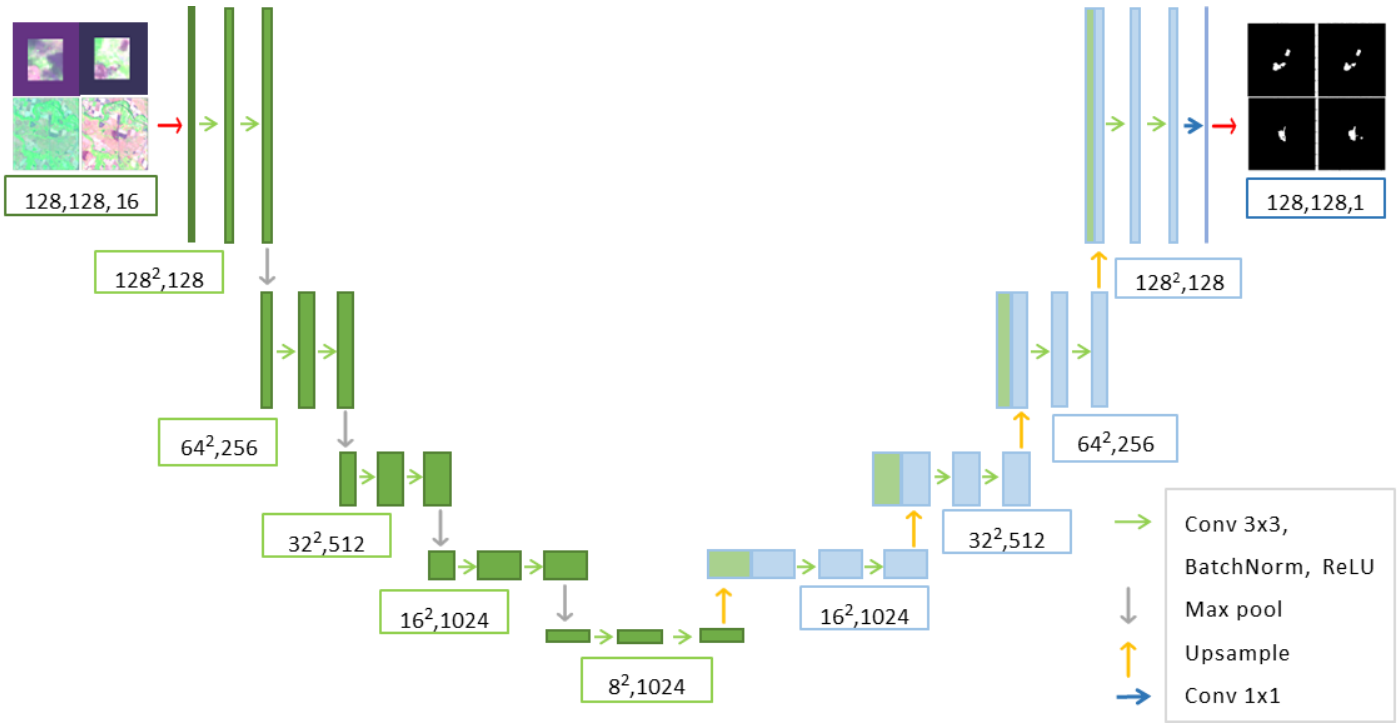


Figura 2.4 Configuración U Net seleccionada

2.3.2 Pruebas preliminares

En primer lugar, debido a la existencia de imágenes repetidas en ambos dataset, que se lograron identificar luego de la realización de las pruebas, se consideran estas como preliminares, que sirven como marco de referencia igualmente para analizar los resultados del modelo, dado su bajo impacto, con 68 datos para el dataset AS, y 16 para el dataset 128, que se incluían tanto en el dataset de entrenamiento como de validación, por lo que se alteraban estos resultados. Estas imágenes se detectaron partir del valor de sus áreas quemadas iguales, pese a tener códigos de identificación diferentes. Producto de esto, hay un leve error de sesgo en los resultados de estas pruebas.

Primeramente, se realizaron cuatro corridas sin utilización de aumentación, comparando los métodos de normalización Min-Max y estandarización. Luego, se seleccionó el mejor método, y a partir de esta prueba se procedió a realizar la optimización de hiperparámetros (HPO por sus siglas en inglés de *Hyperparameter Optimization*), los cuales debían ser ajustados. Para ello se procedió a realizar 4 pruebas adicionales para los hiperparámetros: *learning rate,* tamaño de lotes, cantidad de capas, más dos pruebas con el optimizador Adam y otras dos con aumentación de datos. Cabe señalar que la aumentación de datos consistió en repetir los registros separadamente para cada dataset (entrenamiento y validación), pero aplicando modificaciones para que los registros repetidos permitieran el aprendizaje. Por esto, se aplicaron rotaciones a los datos con 75% de probabilidad, y volteados horizontal y vertical al 50% de la data de manera aleatoria. El factor de aumentación se definió como la cantidad de veces que se repetiría la data (“mult” en el código), siendo 1 (si no se modificaba el dataset), 2 si se duplicaba o 3 si se triplicaban los datos.

3 Resultados y discusión

En principio, en la Tabla 3.1 se resumen los resultados preliminares para ambos modelos usando estandarización como método de normalización. En esta prueba no se usa aumentación de datos. Luego, en la Tabla 3.2 se ven los resultados aplicando normalización Min-Max, mejorando los resultados en ambos modelos, y posteriormente, mejorando todavía más sus resultados duplicando la data con aumentación. Se sombrea en naranjo la única métrica que empeoró cambiando de método de normalización, que es el CE en tres de las cuatro pruebas.

Tabla 3.1 Resultados preliminares con estandarización

Parámetro	*AS*	*128*
Época	25	25
Val loss	0.027	0.037
Train loss	0.015	0.018
IoU	0.781	0.635
DC	0.854	0.734
OE	0.144	0.238
CE	0.114	0.222

Tabla 3.2 Resultados preliminares con Min-Max y aumentación

		AS aumentada		*128 aumentada*	
	Parámetro	*AS*		*128*	
Training y validation	Época	25	24	25	21
	Val los	0.024	0.021	0.036	0.028
	Train los	0.014	0.014	0.018	0.018
	Val DC	0,914	0,927	0,852	0,923
	IoU	0.790	0.818	0.660	0.726
Testing	DC	0.867	0.894	0.753	0.786
	OE	0.102	0.08	0.232	0.149
	CE	0.133	0.113	0.203	0.207

Nota: Las celdas sombreadas en naranja representan los valores en que el rendimiento del método de estandarización superó al de normalización Min-Max.

A partir de esto, la configuración del caso base para las siguientes pruebas de HPO se resume en la Tabla 2.16, de acuerdo a lo visto en las pruebas pasadas.

Tabla 3.3 Configuración prueba preliminar base

Modelo	**Optimizador**	**Aumentación**	**Learning rate**	**Bs**	**Capas**	**Momentum**
AS y 128	SGD	x2	1e-3	10	128-1024	0.7

Desde aquí, se procedió a hacer la HPO, sin embargo, se realizaron estas pruebas solo para AS debido a su mejor rendimiento. Luego, se destinaron los mismos hiperparámetros y la selección del optimizador para la corrida final del modelo 128.

3.1 HPO y ajustes

Se realizaron pruebas para 5 hiperparámetros: la cantidad de capas en la red, *batch size*, *learning rate,* el factor de aumentación de la data con factor 3, y realizar 3 pruebas variando el optimizador. Se marca en negrita la condición base en cada tabla resumen (Véanse las Tablas 3.3-3.7), en las cuales no se indica el tiempo demorado ya que se realizaron en otro computador en el cual el tiempo de procesamiento era aproximadamente el doble. Así, en esta sección se muestran solo los gráficos distintos al caso base.

3.1.1 Arquitectura

Se probaron arquitecturas estándar para el ajuste de la cantidad de filtros por cada capa convolucional (Zunair & Ben Hamza, 2021). Se inició la red con una profundidad de 16, 32, 64, 128 y 160 filtros en su primera convolución, duplicándose en cada subsecuente operación en el camino de *downsampling* hasta mantenerse el número solamente en el último bloque de convolución, para luego retomar los mismos valores al final del *upsampling*.

Se nota claramente que el cambio entre 64 y 128 capas iniciales marca una gran diferencia en el rendimiento del modelo, indicando que más complejidad mejora las predicciones del modelo. A pesar de ello, el aumento hasta 160 capas iniciales no marcó una diferencia tan significativa.

Tabla 3.4 HPO capas

Capas	16/128	32/256	64/512	**128/1024**	160/1280
Val DC	0,915	0,917	0,927	0,927	0,931
Tiempo	01:03:02	01:02:15	01:04:53	-	01:12:13

Nota: En el nombre de las columnas se indica la cantidad de capas iniciales y el máximo de capas al finalizar el camino de *downsampling*. Los valores en negrita representan la configuración base.

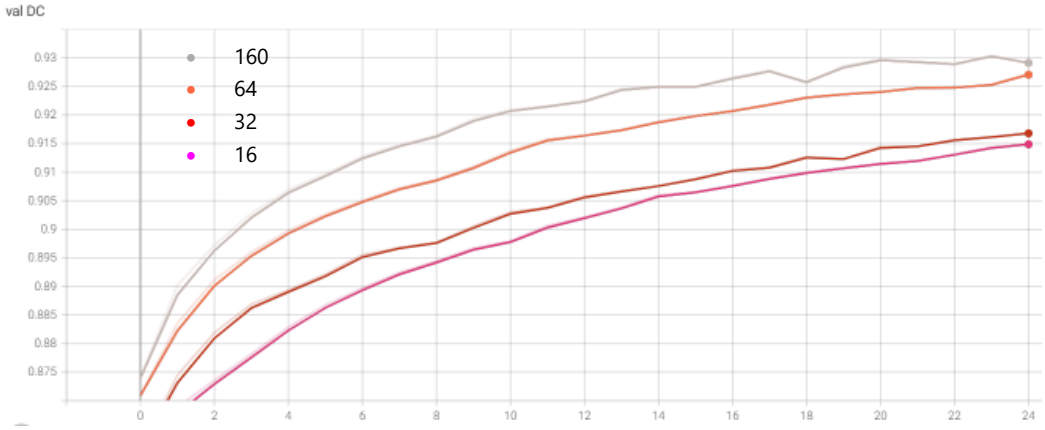


Figura 3.1 Val DC respecto al HPO capas como función de las epochs

Finalmente, debido a que la diferencia fue tan marginal entre las 128 y 160 capas, se mantuvo este hiperparámetro arbirtrariamente en 128 con el objetivo de mantener los requerimientos del sistema no tan elevados, como lo es la cantidad de parámetros, mantener relaciones similares a lo visto en la bibliografía (Zunair & Ben Hamza, 2021) y no aumentar más el tiempo de corridas.

3.1.2 Learning rate

En la Tabla 3.4 se muestran los *learning rates* utilizados. A simple vista se nota claramente la gran diferencia según el orden de magnitud de los *learning rates*. Esto se conecta directamente a la naturaleza de la función de optimización la SGD, desde donde se ven sus grandes impactos entre 1e-4 y 1e-2 como ejemplo.

Tabla 3.5 HPO Learning rate

Lr	0,0001	**0,001**	0,01	0,1	0,5
Val DC	0,893	0,927	0,944	0,951	0,950
Tiempo	01:07:05	-	01:06:41	01:05:46	01:09:48

Nota: Los valores en negrita representan la configuración base.

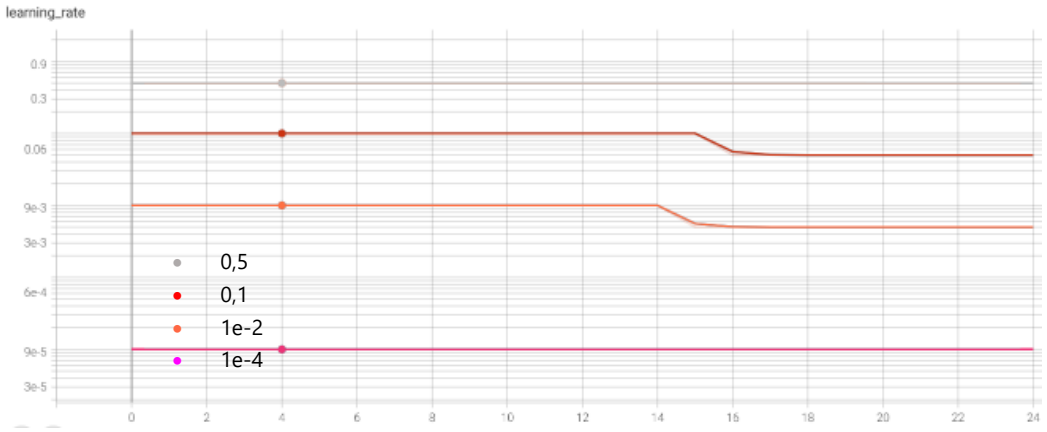


Figura 3.2 Configuraciones de learning rate vs epochs

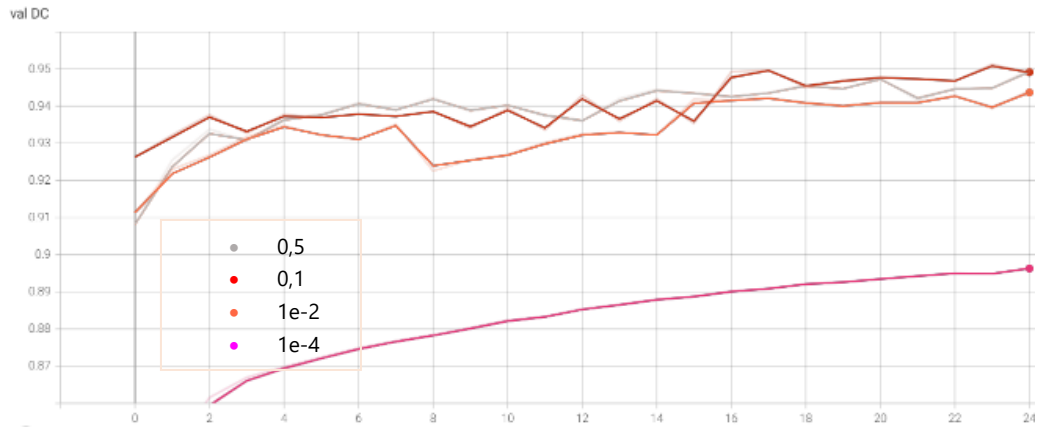


Figura 3.3 Val DC respecto al HPO learning rate

3.1.3 Batch size

El tamaño de lotes no demostró ser una variable sensible para el modelo. A pesar de ello, existieron impedimentos de memoria para realizar la prueba final con 32 datos por lote, por lo que se mantuvo el valor en 16.

Tabla 3.6 HPO Batch size

Bs	4	**10**	16	32	64
Val DC	0,930	0,927	0,929	0,93	0,9301
Tiempo	01:06:44	-	01:05:15	01:05:57	01:06:24

Nota: Los valores en negrita representan la configuración base.

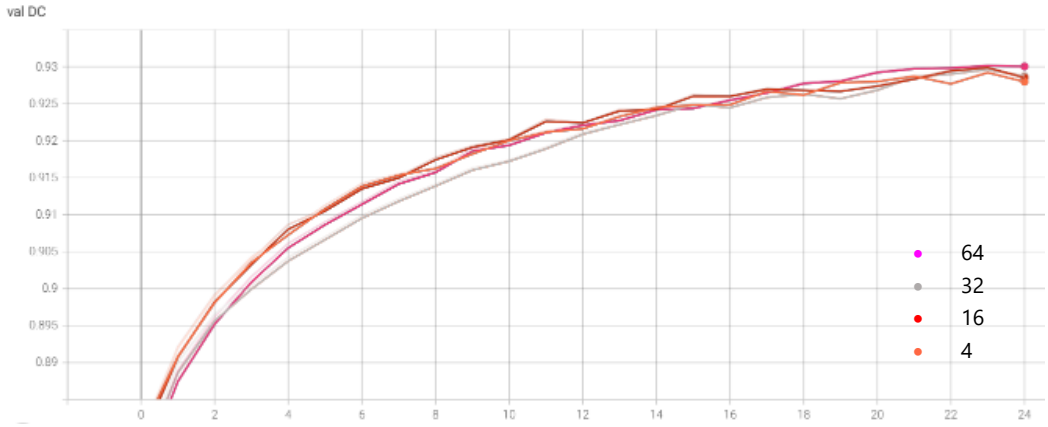


Figura 3.4 Val DC respecto al batch size

3.1.4 Aumentación

La aumentación de datos es muy importante en los modelos de CNN, sin embargo, el cambio en este caso no se ve tan elevado en cuanto al aumento del DC (0,76%). También se observa que la tendencia de la curva es a la estabilidad en las últimas épocas, por lo que se estima que no hubiese aumentado mucho más con más épocas.

Cabe señalar que no se advierte un sobreajuste ni subajuste entre la pérdida de validación y de entrenamiento de las Figuras 3.5 y 3.6 respectivamente, pese a que se encuentran en otras escalas.

Tabla 3.7 HPO factor aumentación

Aumentación	x2	**x3**
Val DC	0,927	0,934
Tiempo	-	01:47:40

Nota: Los valores en negrita representan la configuración base.

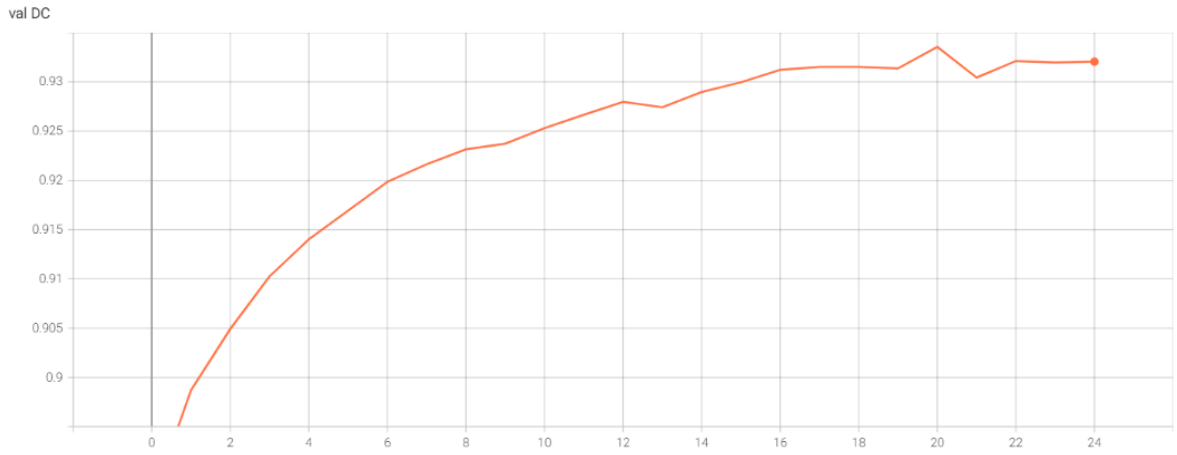


Figura 3.5 Val DC respecto a la aumentación x3

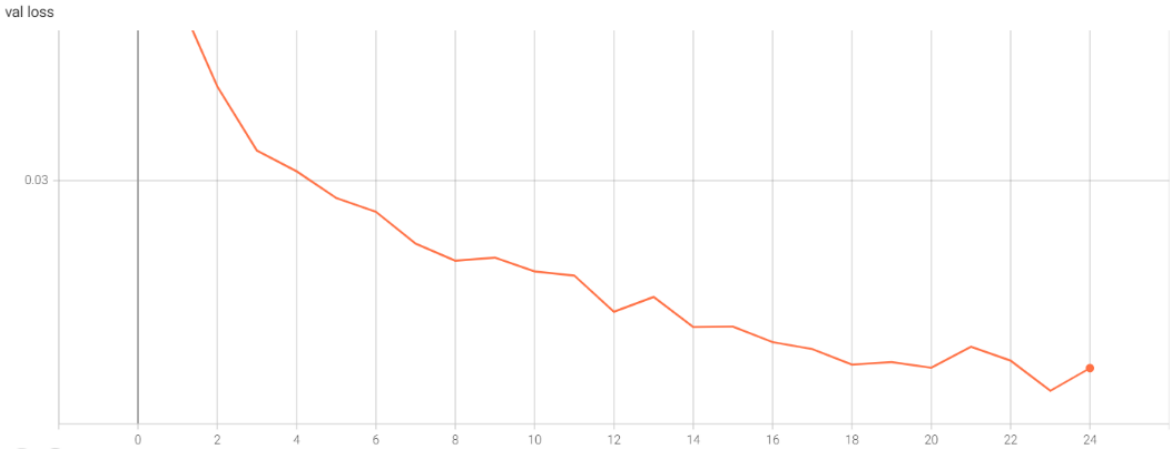


Figura 3.6 Validation loss respecto a la aumentación x3

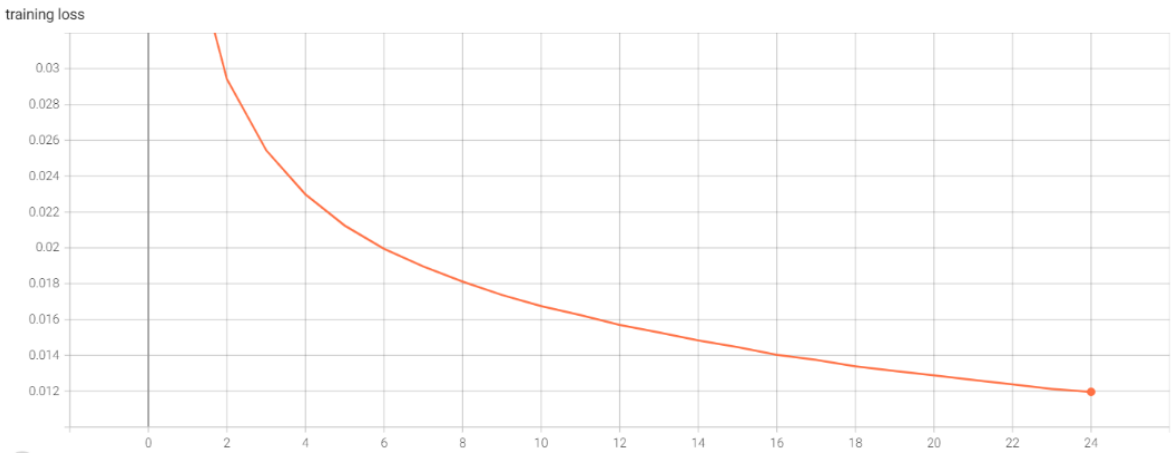


Figura 3.7 Training loss respecto a la aumentación x3

3.1.5 Optimizador Adam

Finalmente, el cambio de optimizador afectó en gran medida las métricas. Se nota que el optimizador Adam genera una muy importante mejora, indicando que el *learning rate* adaptativo es preferible para esta aplicación.

En las Figuras 3.8-3.11 se puede observar cómo afecta la disminución del *learning rate* al finalizar la corrida a mejorar las métricas en este caso, marcando la importancia del hiperparámetro para el optimizador, e indicando que es recomendable a futuro utilizar más épocas para el entrenamiento, dada su última tendencia a seguir mejorando las métricas, pese a la clara estocasticidad presente.

Tabla 3.8 Comparación optimizador Adam

Optimizer	Adam/1e-3	**SGD/Lr=0,001**	Adam/Lr=1e-4
Val los	0,013	0,021	0,011
Train los	0,009	0,014	0,009
Val DC	0,950	0,927	0,951

Nota: Los valores en negrita representan la configuración base.

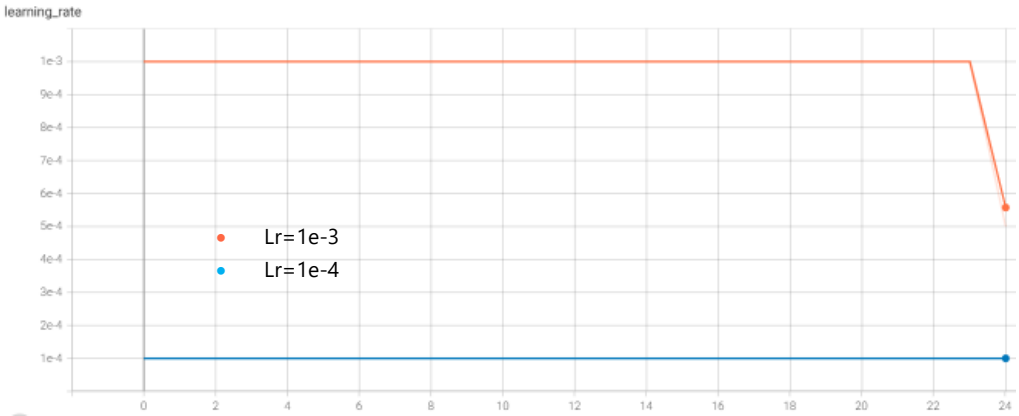


Figura 3.8 Learning rate usando el optimizador Adam

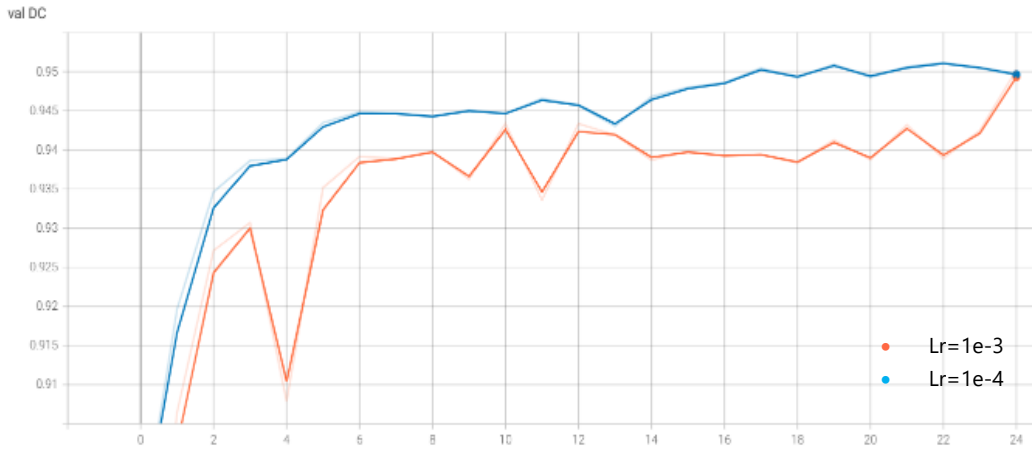


Figura 3.9 Val DC respecto al HPO del optimizador ADAM

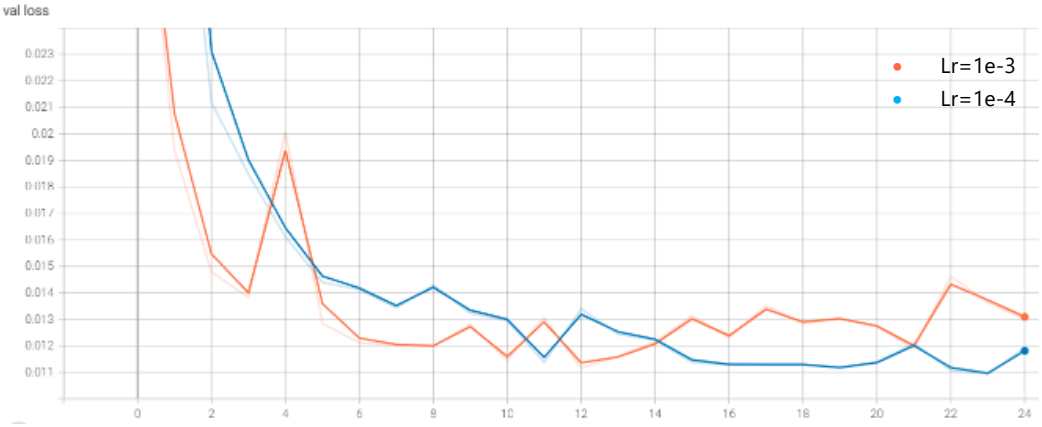


Figura 3.10 Val loss respecto al HPO del optimizador ADAM

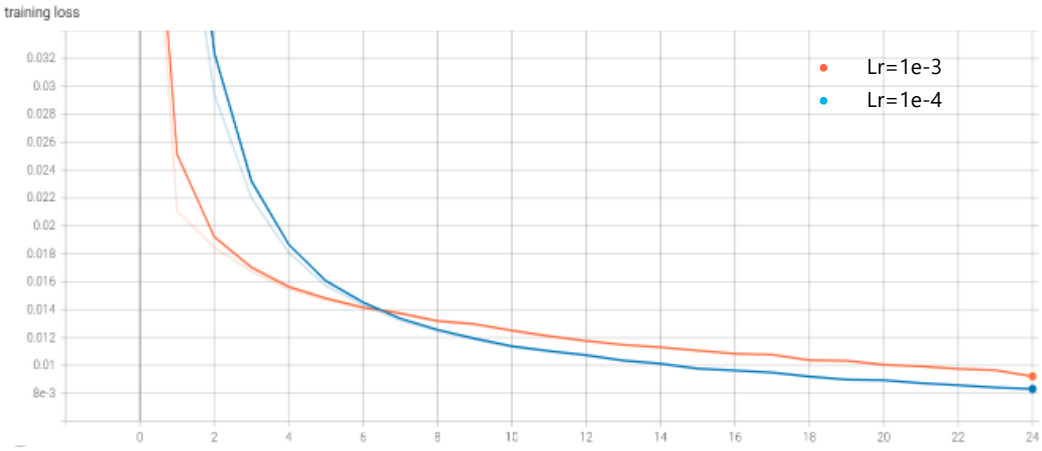


Figura 3.11 Training loss respecto al HPO del optimizador ADAM

3.2 Post-HPO

En la Tabla 3.8 se indica la configuración con los hiperparámetros finales seleccionados luego de realizar la HPO. Luego, en la Tabla 3.9 se muestra un resumen de los resultados comparando pre y post-HPO para ambos modelos.

A partir de los resultados se puede concluir que usando una menor proporción de área no perteneciente a la clase, esta se aprende de mejor manera y luego se predice igualmente con mayor precisión en un test de la misma naturaleza, con resultados sobresalientes que permitirían utilizar este modelo a mayor escala recortando las cicatrices en cuadros delimitadores. No obstante, esto no asegura que el modelo obtenga resultados similares en data de mayores dimensiones, en la cual exista mayor información contextual.

En ambos modelos se presenta un CE mayor al OE, lo cual se puede atribuir en parte al dataset original utilizado, que se formó bajo el principio de preferir omitir áreas quemadas a sobreestimarlas. Desde esto, los modelos pueden reconocer pixeles que estén efectivamente quemados y sus métricas seguirán presentando errores a causa de la data de referencia utilizada.

Se aprecia también muy poco cambio entre las métricas del modelo 128 pre y post-HPO, por lo cual sería recomendable realizar una HPO individual para este modelo.

Tabla 3.9 Configuración post-HPO

Optimizador	*Aumentación*	*Learning rate*	Bs	Capas
Adam	x3	1e-4	16	128-2048

Tabla 3.10 Resumen resultados finales

	Modelo	**As***	**AS + HPO**	**128***	**128 +HPO**
Training and validation	Val los	0,021	0,009	0,028	0,018
	Train los	0,014	0,008	0,018	0,015
	Val DC	0,927	0,952	0,923	0,924
Testing	IoU	0,818	0,880	0,726	0,772
	DC	0,894	0,932	0,786	0,843
	OE	0,080	0,049	0,149	0,131
	CE	0,113	0,076	0,207	0,143

Nota: *En estas corridas los valores son solo referenciales puesto que existe un sesgo en la data como se comentó en el Capítulo 2.

Tabla 3.11 Tiempo modelos +HPO

	AS	*128*
Tiempo 25 epochs	01:54:04	02:48:39

En la Tabla 3.10 se indican los tiempos asociados a cada corrida. Cabe señalar que para estas corridas se utilizó un computador con procesador AMD Ryzen 9 5950X, 3401 Mhz, con 16 procesadores principales y 32 procesadores lógicos, y 32 Gb de RAM. La tarjeta gráfica del computador es la NVIDIA GeForce RTX 3080 Ti. Sobresale claramente la diferencia de tiempo según el modelo, en casi una hora más para el modelo 128, debido a las interacciones que ocurren en este modelo que no ocurren en el *zero padding* del modelo AS.

Finalmente, desde la tendencia en la Figura 3.12, se recomendaría entrenar al modelo con más épocas en caso de que pudiera mejorar el rendimiento. Por otro lado, en las Figuras 3.13-3.14 se ve una clara estabilidad de las funciones de pérdida en el entrenamiento y la validación, indicando un ajuste correcto.

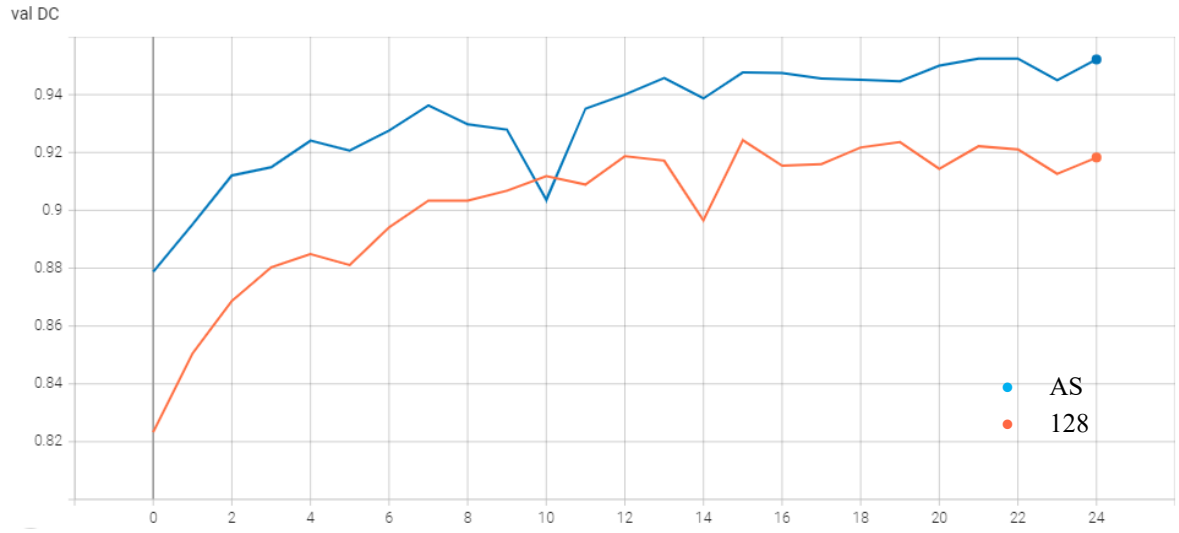


Figura 3.12 Validation DC - configuraciones finales

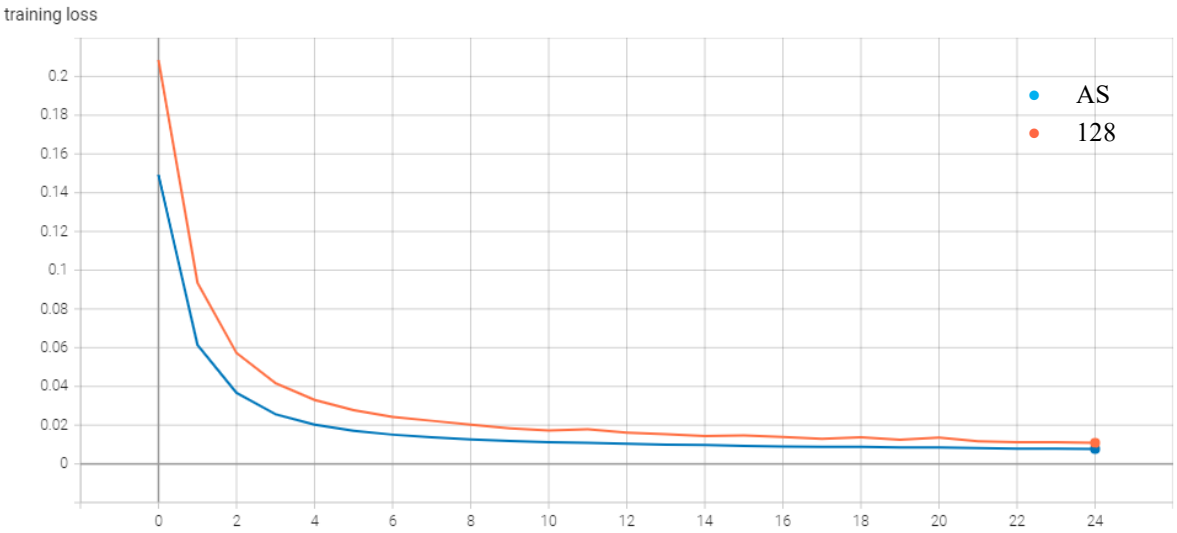


Figura 3.13 Training loss - configuraciones finales

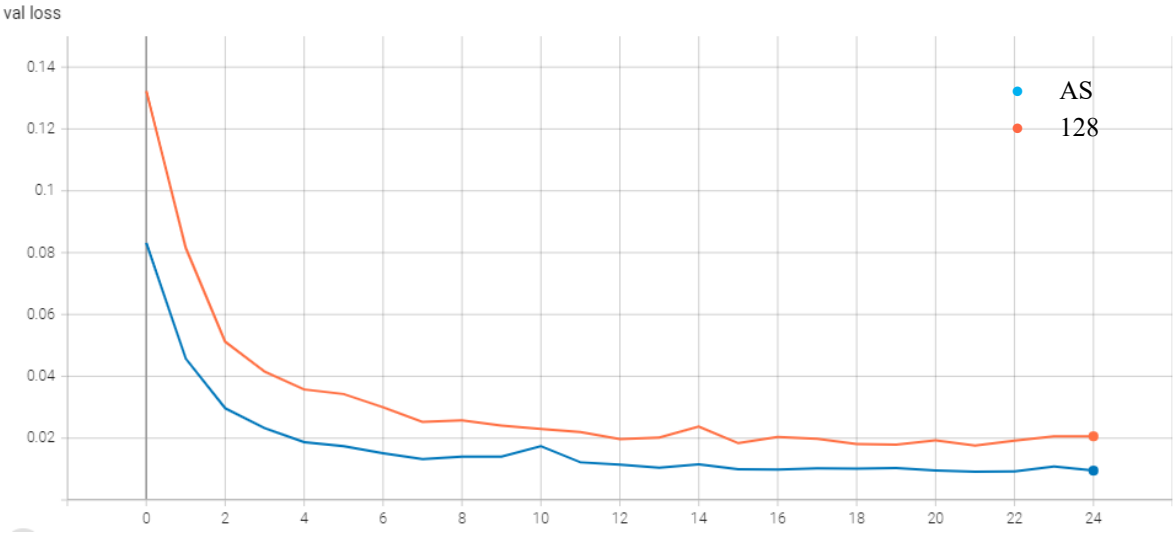


Figura 3.14 Validation loss - configuraciones finales

3.3 Análisis del testing

En esta sección se analiza la predicción de las 100 imágenes de testing del dataset. Para esto, se estudian los resultados en cuanto al DC por cada cuartil asociado a esta métrica, incluyendo los valores promedios de cada variable dentro de cada cuartil con el objetivo de representar más claramente estos.

Se hace la distinción del uso del nombre DC en las columnas de las tablas de esta sección, donde los valores corresponden a los que clasifican a cada cuartil, mientras que $\overline{\boldsymbol{DC}}$ indica el valor promedio de esta métrica del cuartil correspondiente.

3.3.1 Análisis por bandas

En las Tablas 3.12- 3.15 de esta sección se ha analizado el cambio de los promedios de cada banda en relación al DC. Se compararon los 4 cuartiles de esta métrica para comprobar si existe una relación entre los cambios espectrales entre las imágenes pre y post-incendio y la mejora del rendimiento. Además de esto, se revisa si hay una diferencia significativa dentro de los valores de una misma banda que pueda estar relacionado a una mejora del DC. En cada tabla las dos columnas bajo el nombre "Bn°", donde n° son los números de las bandas del 1 al 8, corresponden a los valores promedio de las imágenes pre y post-incendio, con los primeros en la columna inicial a la izquierda y los segundos, de la imagen post, en la segunda columna, a la derecha. Para este análisis los diferenciales entre columnas pre y post se calculan con la Ec. 3, y los deltas entre cuartiles de cada misma imagen (pre o post) con la Ec. 3.1

$$\Delta_n\% = \frac{Banda_n\ post - Banda_n\ pre}{Banda_n\ pre} \cdot 100 \tag{3}$$

$$\%\Delta_q = \frac{Q_4 - Q_1}{Q_1} \cdot 100 \tag{3.1}$$

De esta manera, en la sección se presentan en primer lugar las tablas con las estadísticas correspondientes a las bandas. En segundo lugar, se presentan las Figuras 3.15 y 3.16 con las distribuciones de los promedios de las bandas de las 100 imágenes del dataset vs el DC que se obtuvo para cada una de ellas, comparando en cada gráfico los datos pre y post-incendio, además de marcar las líneas que denotan el promedio de los datos tanto de las imágenes pre como post-incendio.

3.3.1.1 AS

Debe considerarse que en este análisis se realizó la estratificación solo por los cuartiles del DC y no por tamaños, los cuales también pueden influir de manera importante en la métrica para el modelo AS. El efecto de los tamaños se analiza en la Sección 3.3.2.

Tabla 3.12 Análisis de bandas 1-4 modelo AS vs DC

	DC	$\overline{DC}$	B1		Δ_1%	B2		Δ_2%	B3		Δ_3%	B4		Δ_4%
Q1	0,914	0,841	0,252	0,248	-1,6%	0,284	0,254	-10,3%	0,258	0,221	-14,3%	0,375	0,291	-22,3%
Q2	0,950	0,934	0,235	0,239	1,8%	0,256	0,245	-4,5%	0,228	0,218	-4,6%	0,334	0,274	-17,8%
Q4	0,976	0,967	0,225	0,238	6,0%	0,237	0,238	0,6%	0,197	0,206	4,5%	0,331	0,280	-15,4%
Q4	1,0	0,986	0,227	0,234	3,4%	0,250	0,241	-3,9%	0,216	0,213	-1,1%	0,358	0,283	-20,8%
$\%\Delta_q$	6,86%	17,2%	-9,9%	-5,4%		-11,8%	-5,4%		-16,6%	-3,8%		-4,6%	-2,7%	

Tabla 3.13 Análisis de bandas 5-8 modelo AS vs DC

	DC	$\overline{DC}$	B5		Δ_5%	B6		Δ_6%	B7		Δ_7%	B8		Δ_8%
Q1	0,914	0,841	0,380	0,324	-14,7%	0,302	0,296	-2,1%	0,512	0,437	-14,7%	0,521	0,481	-7,8%
Q2	0,950	0,934	0,362	0,330	-8,8%	0,296	0,309	4,6%	0,516	0,425	-17,6%	0,506	0,457	-9,6%
Q4	0,976	0,967	0,321	0,305	-4,9%	0,260	0,279	7,3%	0,557	0,447	-19,8%	0,540	0,484	-10,5%
Q4	1,0	0,986	0,337	0,308	-8,7%	0,267	0,295	10,5%	0,551	0,435	-21,1%	0,545	0,463	-15,0%
$\%\Delta_q$	6,86%	17,2%	-11,3%	-5,0%		-11,6%	-0,2%		7,8%	-0,3%		4,5%	-3,7%	

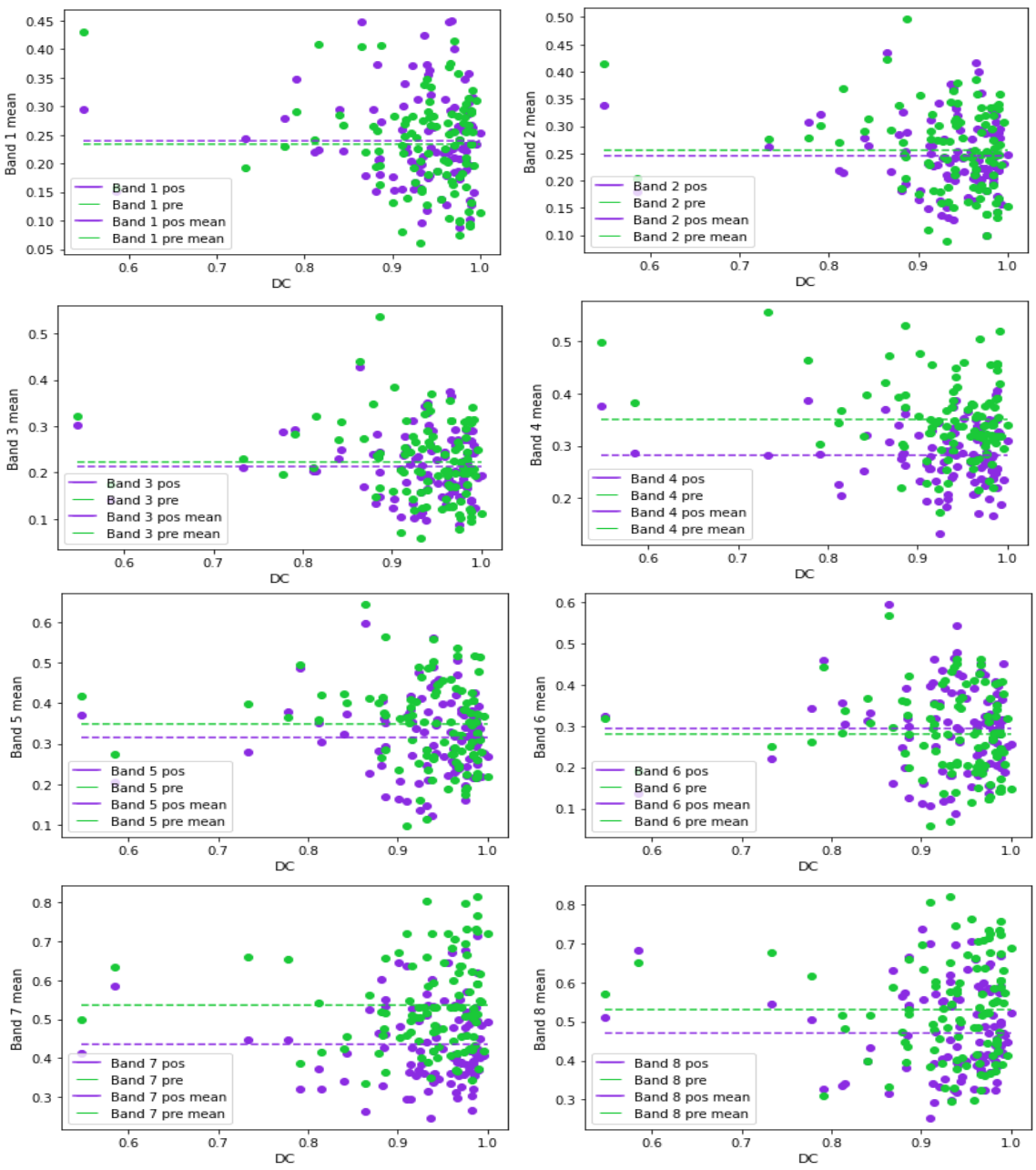


Figura 3.15 Promedios bandas testing AS vs DC

Desde los valores de las Tablas 3.12 y 3.13 se puede notar que respecto al DC, hay un mayor valor en los promedios de todas las bandas pre-incendio que en las post-incendio, esto comparando el primer y último cuartil del DC ($\%\Delta_q$), lo cual indica que el modelo es más sensible a predecir datos que contengan imágenes pre de cierto tipo con mejor precisión, que en este caso son las que tienen valores promedios de sus bandas más cercanos a los de las imágenes post en las bandas 1-6 y más lejanos en las bandas 7 y 8. Además de esto, la banda 3, correspondiente a la banda roja, es la que presenta mayor cambio para la imagen pre (-16,8%), por lo que se observa que probablemente imágenes con menores valores en esta

banda, o bien con valores más próximos a los de la imagen post, tendrán una mejor predicción. Asimismo, la banda 2, correspondiente a la banda verde, es la segunda más importante considerando la diferencia entre cuartiles de las imágenes pre (-11,8%) y post (-5,4%), por lo que también esta banda sería sensible para aportar una correcta respuesta para el modelo AS.

En cuanto a las variaciones entre los promedios de las bandas de las imágenes pre y post ($\Delta_n\%$) se nota que el mayor cambio se encuentra en la banda 4, del NIR, con una diferencia promedio de un 19,1% entre los 4 cuartiles, lo cual se aprecia claramente en el gráfico de la Figura 3.15. Se destacan también los cambios de las bandas 7 y 8, correspondientes al NDVI y al NBR respectivamente, los cuales indican que a mayor diferencia entre los valores de sus bandas pre y post, mejor será la predicción. Se debe recordar que el NDVI tiene una relación directamente proporcional con la banda 4 (NIR) e inversamente proporcional con la banda roja, mientras que el NBR también es directamente proporcional al NIR pero inversamente proporcional a la banda 6, que es el SWIR2 (Véanse las Ecs. 2 y 2.1). Otro punto a señalar, es que las bandas que poseen mayor valor de delta en el Q4 son las bandas 4, 6, 7 y 8, que corresponden a los índices y dos de sus bandas que los componen, demostrando su sensibilidad a los cambios en la vegetación y a su uso para esta aplicación.

3.3.1.2 128

Inicialmente, en la Tabla 3.14 y 3.15 se identifica que el Q1 es muy inferior al del testing del modelo AS de 0,841 en promedio, y que además se produce un salto mucho más pronunciado hasta el DC del Q2, lo cual en parte es debido al registro que fue falso negativo, con un DC de 0. A causa de lo primero, se puede decir que el modelo es más sensible a ciertos atributos de las imágenes, y especialmente entre el Q1 y Q2 por la mayor variación del DC, entre los cuales se pueden encontrar el valor de las bandas y sus relaciones entre la imagen pre y post como algunos de ellos, en adición a la proporción de área quemada y severidad, que se analiza en la siguiente sección 3.3.2.

Tabla 3.14 Análisis de bandas 1-4 modelo 128 vs DC

	DC	$\overline{DC}$	B1		$\Delta_1\%$	B2		$\Delta_2\%$	B3		$\Delta_3\%$	B4		$\Delta_4\%$
Q1	0,814	0,597	0,117	0,126	7,1%	0,163	0,165	1,1%	0,174	0,158	-8,9%	0,311	0,290	-6,8%
Q2	0,909	0,865	0,112	0,116	3,4%	0,155	0,159	2,4%	0,165	0,160	-2,7%	0,303	0,277	-8,7%
Q3	0,956	0,936	0,106	0,110	3,8%	0,142	0,145	2,3%	0,146	0,143	-2,2%	0,276	0,251	-8,9%
Q4	0,991	0,975	0,110	0,108	-2,0%	0,150	0,152	1,6%	0,152	0,151	-0,5%	0,300	0,291	-3,0%
$\%\Delta_q$	21,7%	63,5%	-6,1%	-14,1%		-8,5%	-8,1%		-12,7%	-4,7%		-3,5%	0,5%	

Tabla 3.15 Análisis de bandas 5-8 modelo 128 vs DC

	DC	$\overline{DC}$	B5		$\Delta_5\%$	B6		$\Delta_6\%$	B7		$\Delta_7\%$	B8		$\Delta_8\%$
Q1	0,814	0,597	0,312	0,284	-8,7%	0,195	0,178	-8,7%	0,675	0,660	-2,2%	0,562	0,571	1,5%

Q2	0,909	0,865	0,305	0,296	-2,8%	0,187	0,189	0,9%	0,685	0,655	-4,5%	0,561	0,543	-3,2%
Q3	0,956	0,936	0,289	0,275	-4,8%	0,180	0,175	-2,8%	0,674	0,639	-5,2%	0,534	0,520	-2,6%
Q4	0,991	0,975	0,278	0,279	0,3%	0,170	0,170	0,0%	0,700	0,676	-3,4%	0,584	0,574	-1,7%
$\%\Delta_q$	21,7%	63,5%	-10,8%	-1,9%		-12,4%	-4,2%		3,6%	2,4%		3,8%	0,4%	

Ahora no se ven cambios de igual magnitud a los que se vieron para el modelo AS ni en el delta intercuartil ($\%\Delta_q$) ni el delta entre imágenes pre y post ($\Delta_n\%$), lo cual se comprende dado el muy superior tamaño de cada imagen del dataset del modelo 128 sobre el AS, donde la proporción de área quemada es muy inferior al caso previo, y por tanto las variaciones de la imagen podrían ser menores en promedio, o mayores debido a otras variables, contemplando los cambios de uso de suelo, distintos a lo que ocurre con los incendios, o simplemente cambios de vegetación producto del tiempo o el cambio de estaciones del año.

En este caso, se observa que una disminución del valor promedio de la banda 1 (banda azul) indica una tendencia a mejorar el valor del DC, y al igual que para AS, con la banda 3 o roja ocurre el mismo comportamiento, ambas disminuyendo la brecha entre los valores de las bandas pre y post.

Respecto a las variaciones entre bandas pre y post ($\Delta_n\%$), se pudo observar en el Q4, de mejor rendimiento, que las bandas 4 y 7 son las que mantienen un mayor valor delta, con un -3,0% y -3,4% respectivamente, lo cual se explica dada la sensibilidad del NDVI a los cambios en la vegetación, y a la dependencia de este índice en la banda NIR para esto, por la característica de que la vegetación refleje estos rayos intensamente. Esto sugiere que el modelo es más sensible a predecir mejor cuando se mantienen estas diferencias, mientras que por el contrario en las otras bandas, 1-3 y 5-6, se nota una tendencia a disminuir las brechas a medida que se mejoran los resultados.

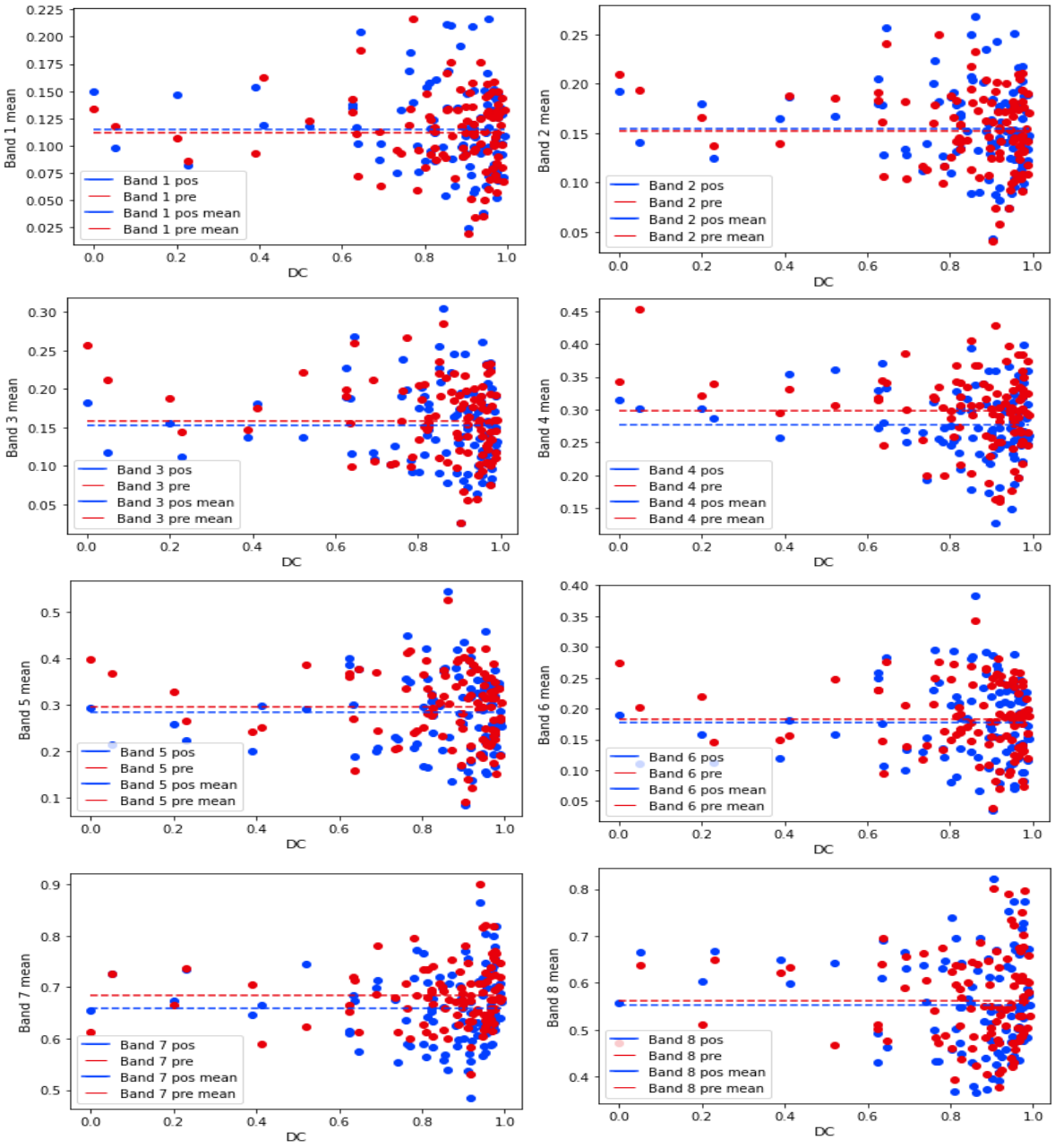


Figura 3.16 Promedios bandas testing 128 vs DC

3.3.2 Área quemada y severidad

Otro factor relevante para entender el rendimiento del modelo es el porcentaje de área quemada de cada registro y el tamaño de los archivos del dataset, ya que pese a que una imagen sea más grande en el dataset AS, no implica que las superficies quemadas sean mayores, puesto que esto depende del cuadrilátero delimitador que se haya formado. En particular, existen casos en que un cuadrilátero puede contener más de un incendio o distintas formas de estos, resultando en una menor superficie quemada y mayor área total que otro registro que tenga la situación contraria. Debido a esto, el porcentaje de área quemada

estudiado en esta sección se calcula con el área efectivamente quemada, obtenida de los vectores de los incendios sin contar el área exterior, respecto al total cuadrante de 128x128, sin embargo, el área total corresponde al tamaño del cuadro delimitador o *bounding box*, lo cual se analiza solo para el dataset AS dado que el dataset 128 es uniforme en sus tamaños.

En primer lugar, en la Tabla 3.16 se ve un resumen de los datos, mientras que en la Figura 3.17 se presentan dos gráficos de estos: el de la izquierda con el porcentaje de área quemado en el testing de AS vs el DC, y a la derecha el área total vs el DC.

Tabla 3.16 Análisis distribución de área, %Área quemada y severidad AS vs DC

	DC	$\overline{DC}$	% Área quemada	Área total, ha	RdNBR
Q1	0,914	0,841	2,93	132,4	515,8
Q2	0,950	0,934	4,74	213,2	502,1
Q3	0,976	0,967	5,39	218,5	395,2
Q4	1,0	0,986	3,29	133,4	398,5

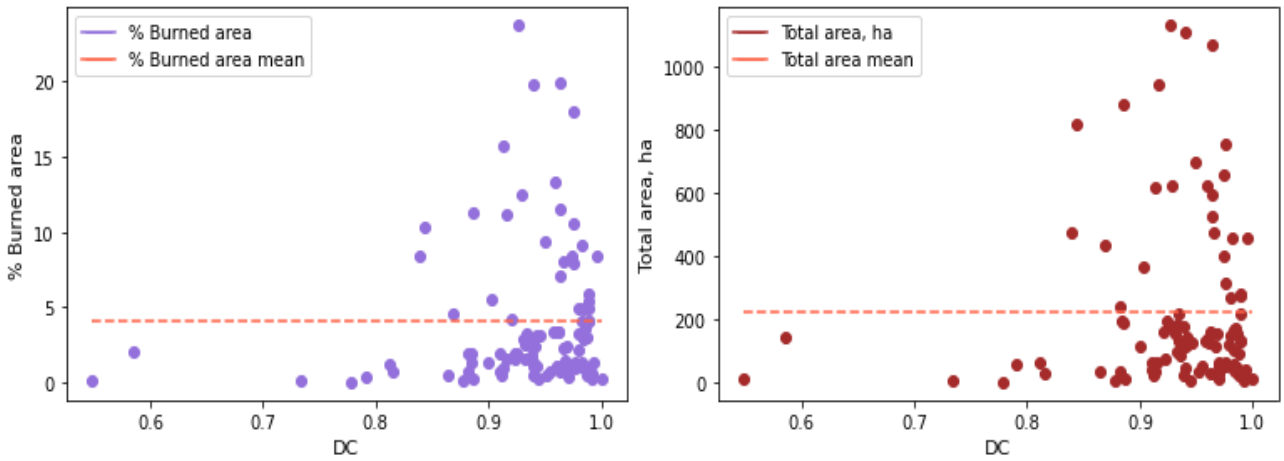


Figura 3.17 Distribución de área y su porcentaje quemado testing AS vs DC

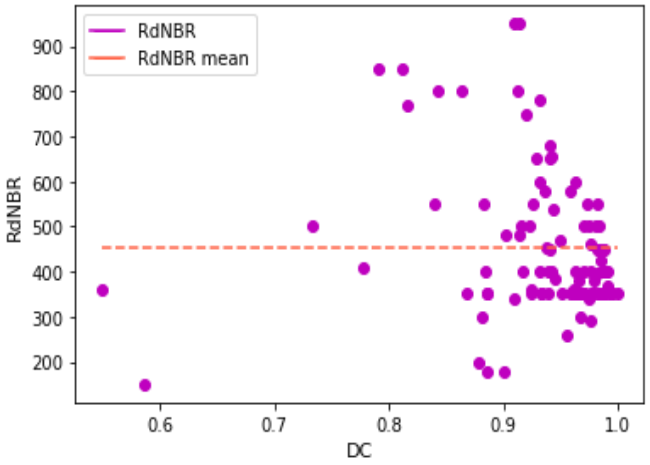


Figura 3.18 RdNBR AS vs DC

Se puede notar desde la Figura 3.17 y los datos de la Tabla 3.16 que el modelo no posee una tendencia clara respecto al porcentaje de área quemada presente en el input ni al área total de este para entregar una buena predicción, sin embargo, se puede ver que todos los datos con peores rendimientos poseen tanto el porcentaje de área quemada como su área total bajo el promedio de los datos, lo cual indica que el modelo falla más frecuentemente en datos más pequeños o con menor proporción de área quemada. Adicionalmente, se destaca que los datos poseen un tamaño promedio pequeño en comparación al área total de los 128x128 pixeles, los cuales corresponden a un total de 1474,6 ha, mientras que un 4% de esta área corresponde a 59 ha, que es aproximadamente donde se concentran los datos del testing.

Respecto a la severidad, se encuentra un mejor rendimiento en los datos del testing para valores del índice RdNBR más bajos, cercanos al valor de 400.

Los datos del testing del modelo 128 se ven en la Figura 3.19, y en la Tabla 3.17 el resumen de estos. En este caso se ve la misma tendencia, de que los datos con peor rendimiento poseen un área quemada más pequeña, bajo el promedio de los datos como se ve claramente bajo la línea del promedio de los datos del porcentaje de área quemada *(% Burned area mean*) en la Figura 3.19. Se advierte también una mayor frecuencia de datos que poseen un peor DC cuando el porcentaje de área quemada es inferior al promedio, lo cual ocurre también respecto al índice RdNBR, mejorando las predicciones en el mismo rango en que se vio para el modelo AS, con valores cercanos a 400 en los dos cuartiles superiores.

Tabla 3.17 Porcentaje de área quemada y severidad 128 vs DC

	DC	$\overline{DC}$	% Área quemada	RdNBR
Q1	0,814	0,597	2,17	481,0
Q2	0,909	0,865	3,86	492,6
Q3	0,956	0,936	6,87	444,4

Q4	0,991	0,975	3,38	393,7

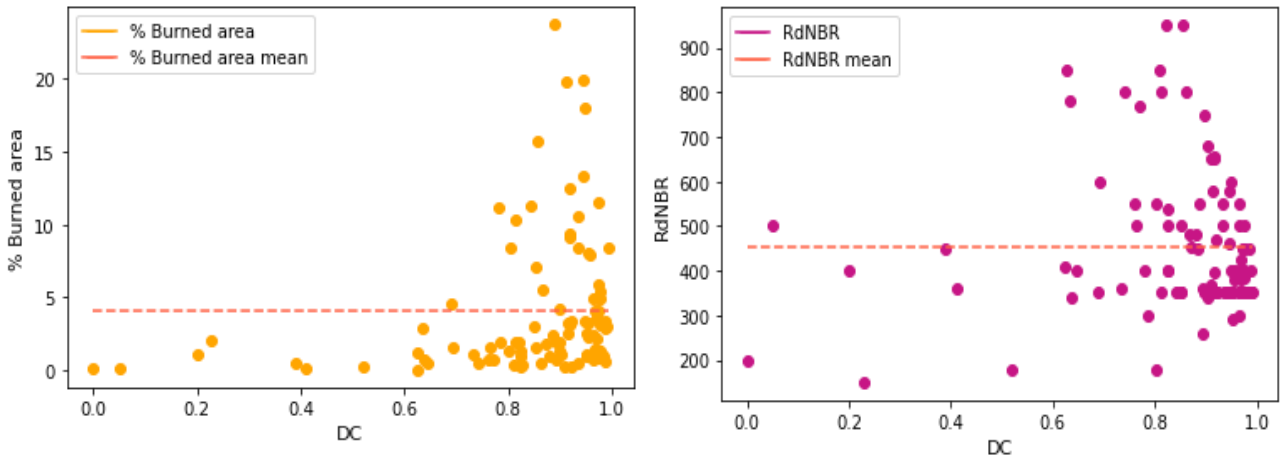


Figura 3.19 Distribución porcentaje de área quemada y severidad en el testing 128 vs DC

Otro punto a mencionar es que se puede apreciar que las distribuciones de datos de porcentaje de área quemada incluyen cada uno de los cuartiles de tamaño gracias al orden dado inicialmente al dataset, alternando cada registro de los datos ingresados al modelo según los cuartiles de las áreas reales quemadas del dataset completo, como se comenta en la sección 2.2.1.7.

3.3.3 Ejemplos testing

Se comparan a continuación los mismos incendios para ambos modelos, presentando un caso de una predicción casi perfecta, una deficiente, otro caso específico en que se presenta bandeo y un último caso respecto a la predicción de quemas agrícolas para el modelo 128.

La comparación que se puede ver en la Figura 3.20 demuestra que ambos modelos predijeron casi perfectamente la cicatriz. La predicción del modelo AS tuvo un DC de 1, un OE de 0% y un CE de 1%, mientras que la del modelo 128 tuvo un DC de 0,99 con un CE y OE de 1% en ambos. El índice RdNBR de este evento es de 350, mientras que el porcentaje de área quemada es cercano al 10%, superior al promedio de los datos de ambos datasets.

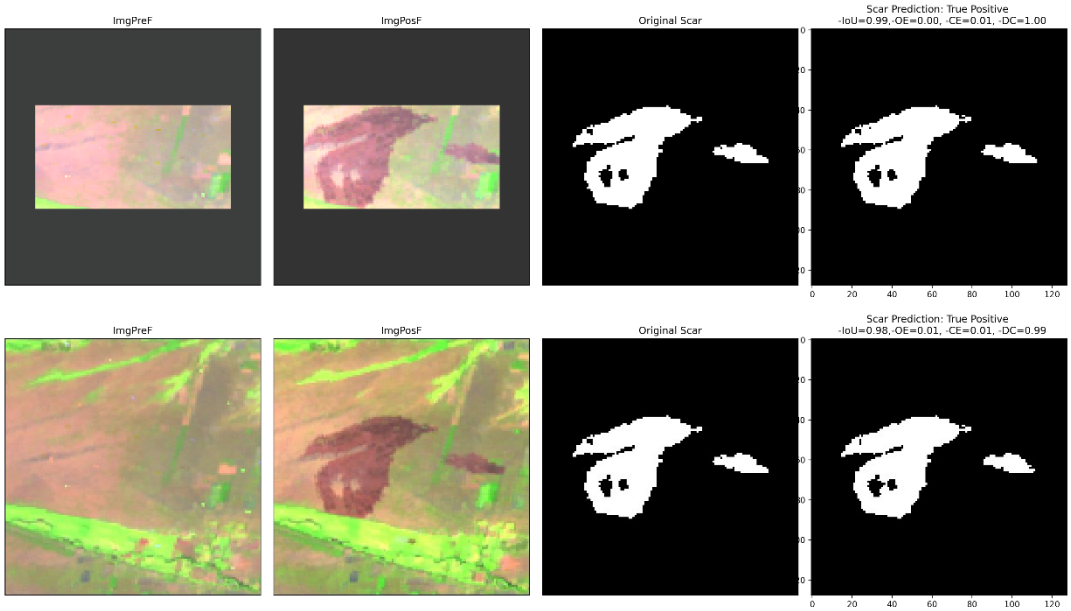


Figura 3.20 Comparación testing alto DC

En el ejemplo de mala predicción de la Figura 3.21, se presenta el único falso negativo del modelo 128, y una de las predicciones del primer cuartil en cuanto al DC del modelo AS, que representa a sus peores rendimientos, con un DC de 0,88, que se atribuye a su OE de 0,22%, mientras que, por otra parte, no tuvo CE.

Dentro de los factores a considerar para entender el rendimiento ocurrido en el mapeo de este incendio, en primer lugar, respecto al falso negativo, se encuentran los valores promedios de las bandas 7 y 8 de las imágenes 128x128, que en este caso son inversos a la tendencia regular, puesto que existe un valor mayor para el NDVI y NBR en las imágenes post que en las imágenes pre, lo cual ocurre debido al visible aumento en la vegetación de la imagen post. Además de esto, el índice RdNBR de esta imagen indica una baja severidad, igual a 200, y el porcentaje de área quemada es inferior al 2%, características que como se comenta en la sección anterior, se ven repetidamente en los bajos rendimientos del modelo.

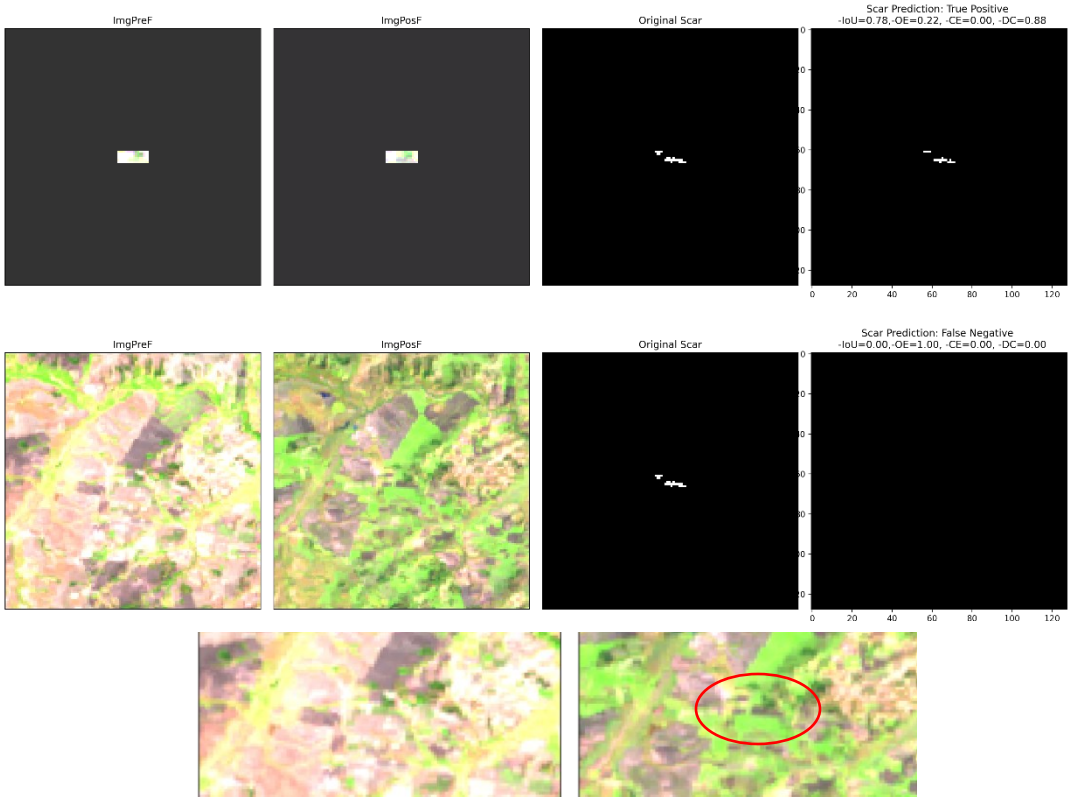


Figura 3.21 Comparación testing bajo DC

En la Figura 3.22, se ve un caso con bandeo (Revisar Anexo A), donde se obtiene un DC de 0,94 para AS, atribuido al OE de 0,11, sin CE; mientras que se obtiene un DC igual a 0,82 para el modelo 128, atribuido, al contrario, al CE de 0,3, sin OE. Se observa que el modelo AS al tener un menor tamaño de los datos de ingreso responde mejor a cicatrices de incendio pequeñas que el modelo 128. Es relevante comentar que en este caso la predicción del modelo 128 disminuye su valor del DC debido a la comisión, puesto que el área quemada contigua no es reconocida como parte de la cicatriz del incendio, pese a estar efectivamente quemada. Esta superficie pudo ser ignorada en la cicatriz original por considerarse una quema agrícola, sin embargo no se logra advertir a simple vista el motivo de su omisión, por lo que el modelo puede no entender tampoco. Por esto, el criterio para la selección de la data ingresada al modelo es muy importante, especialmente en incendios.

Finalmente, cabe mencionar que el modelo AS identifica simplemente zonas quemadas como tal, pero el modelo 128 en algunos casos logra diferenciar entre una quema agrícola y un incendio, como se muestra en la Figura 3.23. En la primera imagen se nota una correcta predicción, con un DC de 0,95, sin incluir quemas agrícolas que no representan a incendios, mientras que en la segunda imagen se ve por el contrario una incorrecta predicción del área quemada, incluyendo quemas agrícolas y omitiendo área quemada, con uno de los peores rendimientos del modelo con un DC de 0,23. Para mejorar esta clasificación, sería necesario incluir una mayor cantidad de imágenes con quemas agrícolas, o bien, incluso se podrían

agregar las quemas agrícolas como una nueva clase, pero se requeriría de un nuevo dataset que las tenga identificadas.

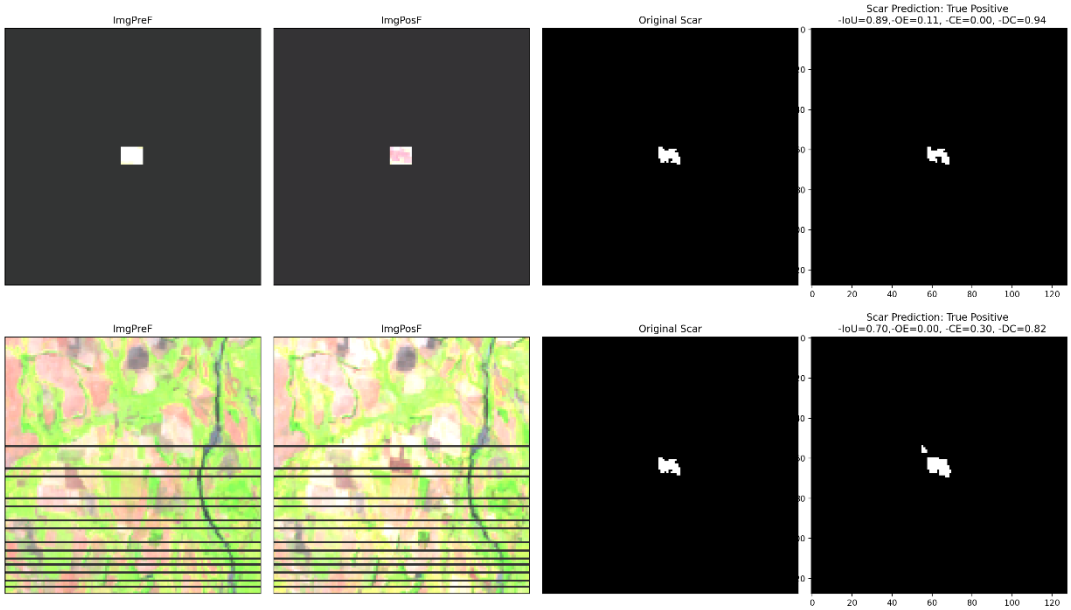


Figura 3.22 Comparación testing con bandeo

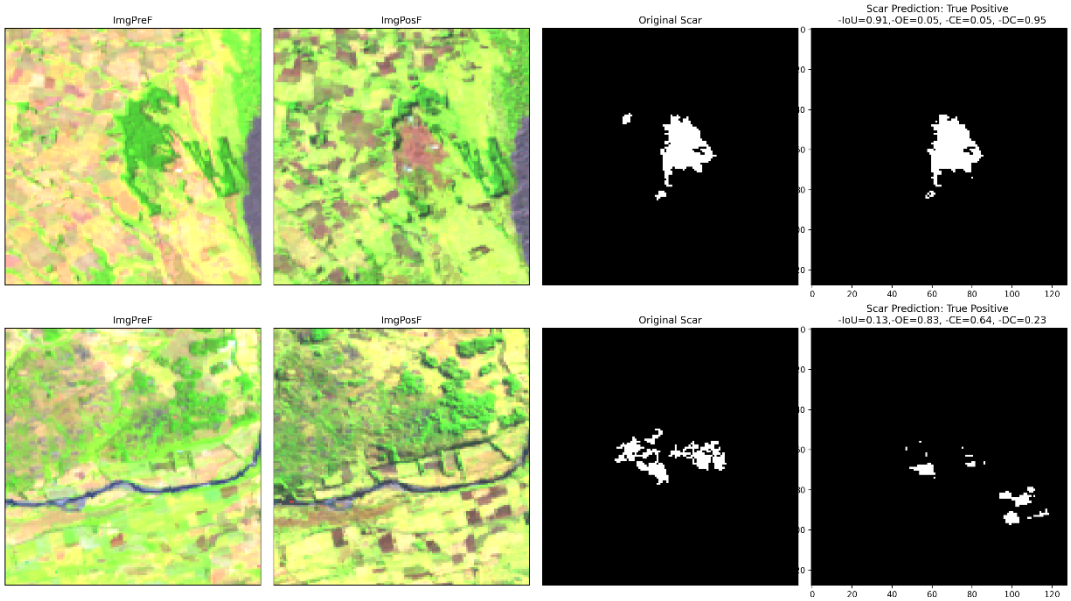


Figura 3.23 Comparación testing 128 quemas agrícolas

4 Conclusiones

En primer lugar, se reconoce que el uso de DL permite automatizar muchos procesos previamente muy demandantes en términos de tiempo, y el mapeo de áreas quemadas en imágenes satelitales no es la excepción, donde las CNN, y en particular el modelo U-Net han sobresalido notablemente, confirmándose el rendimiento de U-Net este trabajo.

Sobre el entrenamiento de los modelos, al utilizar DL se debe contar con una gran cantidad de datos, los cuales deben tener la menor cantidad de errores o sesgos. En este trabajo estos factores se consideran como parte de las limitaciones existentes, puesto que el mapeo de áreas quemadas abarca muchos diferentes escenarios. Es relevante mencionar que el principio de omisión por sobre la comisión en la obtención de la data usada para el entrenamiento de este modelo, se considera como un error inherente a la data, desde el cual se reproducen distintos errores, e incluso se miden errores de comisión que no existen en algunos casos, cuando hay área que efectivamente fue parte de un incendio y no se incluyó originalmente debido a limitaciones del programa y algoritmo usados para su obtención. Por estos motivos, debido a la gran complejidad del problema, se requiere de datos que permitan su aprendizaje incluyendo cada escenario posible, y con las proporciones de cada clase de manera representativa en cada lote alimentado al modelo, de distintos tamaños, severidades, proporciones de área quemada, entre otros.

Respecto a las imágenes obtenidas de los satélites Landsat, se puede comentar que existen distintos errores, desde los bandeos, los datos fuera de rango que tuvieron que ser preprocesados, hasta la misma resolución de los 30 metros espaciales por pixel, que generan dificultades para el reconocimiento preciso de las áreas quemadas bajo distintas severidades.

La HPO fue un paso fundamental para mejorar los resultados del modelo, donde el *learning rate*, el factor de aumentación de los datos y la cantidad de capas aportaron a mejorar el DC significativamente. Además de esto, el cambio al optimizador Adam aportó a mejorar el rendimiento de ambos modelos en gran medida también. Pese a lo anterior, el modelo 128 podría mejorar su rendimiento si se le aplica una HPO individualmente.

Para analizar el rendimiento del modelo, en el testing, se analizaron los valores promedios de las bandas, el porcentaje de área quemada, el área total de cada input y la severidad, respecto a los valores del DC obtenidos. Primero, de las bandas utilizadas, incluyendo imágenes pre y post, la banda NIR y los índices NDVI y NBR presentan las mayores diferencias de valores promedio entre las imágenes pre y post, mientras que las otras bandas tienden a tener una menor variación, lo que significaría que el modelo predice mejor cuando los mayores cambios se presentan solo en estas bandas. Segundo, sobre el porcentaje de área quemada, se vio repetidamente que los peores rendimientos se dieron en ambos modelos en bajos porcentajes, bajo el 3%, y lo mismo se vio para el área total de cada input del modelo AS. Tercero, en cuanto a la severidad, se vio una tendencia a rendir mejor con valores cercanos a un RdNBR de 400, que corresponde a una mediana severidad. Finalmente, ambos

modelos tuvieron mayores errores de comisión que de omisión en el testing, lo cual se puede deber en cierta medida a la data de referencia usada, que en una importante cantidad de data omitía pixeles quemados, como se comentaba previamente.

Con esto, se concluye que los modelos propuestos poseen un potencial para automatizar completamente el proceso de obtención de cicatrices de incendios desde imágenes satelitales para las regiones de Valparaíso y Biobío inicialmente, y con un potencial para todo Chile. Para este objetivo se aproxima más el modelo AS debido a sus mejores métricas, con el DC de 0,932 en el testing, representando una alternativa muy prometedora para reconocimiento de cicatrices dentro de un cuadrilátero delimitador o *bounding box*, lo cual se puede realizar automáticamente a partir de una geolocalización precisa de las coordenadas reportadas de cada incendio. De la misma manera, el modelo 128 podría funcionar bajo esa modalidad, con un mayor margen de error en cuanto a las coordenadas entregadas, sin embargo, requiere de ajustes previos para disminuir sus errores.

4.1 Trabajo futuro

Se propone como trabajo futuro utilizar imágenes del satélite Sentinel 2, las cuales poseen una mejor resolución espacial, de 10 metros por pixel, con el objetivo de capturar con mayor precisión las superficies de las cicatrices de incendios. En esta línea, una mayor cantidad de imágenes, seleccionadas acuciosamente permitirían mejorar el rendimiento de los modelos de manera importante. Adicionalmente, considerando que no existen muchos estudios para la distinción de incendios y quemas agrícolas, se recomienda estudiar con mayor detalle la presencia de estas últimas dentro de las imágenes, y potencialmente incluirlas como una nueva clase, con el fin de que el modelo clasifique erróneamente como parte de los incendios.

Realizar la HPO para el modelo 128 específicamente también aportaría a mejorar el rendimiento del modelo, y desde aquí, se propone usar una herramienta diseñada para la optimización automática de los hiperparámetros como lo es Autogluon (Klein et al., 2020).

Finalmente, se recomienda estudiar el uso de herramientas de post-procesamiento como *conditional random fields* (CRFs) o *weighted composite filters* (WSF) (Cheng & Liu, 2020), que podrían mejorar la segmentación ya obtenida.

5 Referencias

Alzubaidi, L., Zhang, J., Humaidi, A. J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., . . . Farhan, L. (2021). Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. *Journal of Big Data*, *8*(1), 53. https://doi.org/10.1186/s40537-021-00444-8

Arruda, V. L. S., Piontekowski, V. J., Alencar, A., Pereira, R. S., & Matricardi, E. A. T. (2021). An alternative approach for mapping burn scars using Landsat imagery, Google Earth Engine, and Deep Learning in the Brazilian Savanna. *Remote Sensing Applications: Society and Environment*, *22*, 100472. https://doi.org/https://doi.org/10.1016/j.rsase.2021.100472

Bar, S., Parida, B. R., & Pandey, A. C. (2020). Landsat-8 and Sentinel-2 based Forest fire burn area mapping using machine learning algorithms on GEE cloud platform over Uttarakhand, Western Himalaya. *Remote Sensing Applications: Society and Environment*, *18*, 100324. https://doi.org/https://doi.org/10.1016/j.rsase.2020.100324

Barsi, Á., Kugler, Z., László, I., Szabó, G., & Abdulmutalib, H. M. (2018). Accuracy Dimensions in Remote Sensing. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *42.3*, 61. https://doi.org/10.5194/isprs-archives-XLII-3-61-2018

BCN. (2022). División regional: polígonos de las regiones de Chile. In Regiones (Ed.). https://www.bcn.cl/siit/mapas_vectoriales: Biblioteca del Congreso Nacional de Chile.

Belenguer-Plomer, M. A., Tanase, M. A., Chuvieco, E., & Bovolo, F. (2021). CNN-based burned area mapping using radar and optical data. *Remote Sensing of Environment*, *260*, 112468. https://doi.org/https://doi.org/10.1016/j.rse.2021.112468

Boschetti, L., Roy, D. P., Justice, C. O., & Humber, M. L. (2015). MODIS–Landsat fusion for large area 30m burned area mapping. *Remote Sensing of Environment*, *161*, 27-42. https://doi.org/https://doi.org/10.1016/j.rse.2015.01.022

Bottou, L. (2010). Large-Scale Machine Learning with Stochastic Gradient Descent. *Proc. of COMPSTAT*. https://doi.org/10.1007/978-3-7908-2604-3_16

Cabral, A. I. R., Silva, S., Silva, P. C., Vanneschi, L., & Vasconcelos, M. J. (2018). Burned area estimations derived from Landsat ETM+ and OLI data: Comparing Genetic Programming with Maximum Likelihood and Classification and Regression Trees. *ISPRS Journal of Photogrammetry and Remote Sensing*, *142*, 94-105. https://doi.org/https://doi.org/10.1016/j.isprsjprs.2018.05.007

Cheng, X., & Liu, H. (2020). A Novel Post-Processing Method Based on a Weighted Composite Filter for Enhancing Semantic Segmentation Results. *Sensors*, *20*(19). https://doi.org/10.3390/s20195500

Chuvieco, E., Lizundia-Loiola, J., Pettinari, M. L., Ramo, R., Padilla, M., Tansey, K., . . . Plummer, S. (2018). Generation and analysis of a new global burned area product based on MODIS 250 m reflectance bands and thermal anomalies. *Earth Syst. Sci. Data*, *10*(4), 2015-2031. https://doi.org/10.5194/essd-10-2015-2018

Chuvieco, E., Mouillot, F., van der Werf, G. R., San Miguel, J., Tanase, M., Koutsias, N., . . . Giglio, L. (2019). Historical background and current developments for mapping burned area from satellite Earth observation. *Remote Sensing of Environment*, *225*, 45-64. https://doi.org/https://doi.org/10.1016/j.rse.2019.02.013

Collins, L., McCarthy, G., Mellor, A., Newell, G., & Smith, L. (2020). Training data requirements for fire severity mapping using Landsat imagery and random forest. *Remote Sensing of Environment*, *245*, 111839. https://doi.org/https://doi.org/10.1016/j.rse.2020.111839

CONAF, S. (2022). *Número de incendios forestales y superficie afectada a la fecha*

Congalton, R. G. (1991). A review of assessing the accuracy of classifications of remotely sensed data. *Remote Sensing of Environment*, *37*(1), 35-46. https://doi.org/https://doi.org/10.1016/0034-4257(91)90048-B

de Bem, P. P., de Carvalho Júnior, O. A., de Carvalho, O. L., Gomes, R. A., & Fontes Guimarães, R. (2020). Performance Analysis of Deep Convolutional Autoencoders with Different Patch Sizes for Change Detection from Burnt Areas. *Remote Sensing*, *12*(16). https://doi.org/10.3390/rs12162576

Fielding, A. H., & Bell, J. F. (1997). A review of methods for the assessment of prediction errors in conservation presence/absence models [Article]. *Environmental Conservation*, *24*(1), 38-49. https://doi.org/10.1017/S0376892997000088

Fornacca, D., Ren, G., & Xiao, W. (2018). Evaluating the Best Spectral Indices for the Detection of Burn Scars at Several Post-Fire Dates in a Mountainous Region of Northwest Yunnan, China. *Remote Sensing*, *10*(8). https://doi.org/10.3390/rs10081196

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

Google. (2021). *Google Earth Engine*. https://earthengine.google.com/

Hamilton, D., Hamilton, N., & Myers, B. (2019, 2019//). Evaluation of Image Spatial Resolution for Machine Learning Mapping of Wildland Fire Effects. Intelligent Systems and Applications, Cham.

Hamilton, D., Myers, B., & Branham, J. (2017). Evaluation of Texture as an Input of Spatial Context for Machine Learning Mapping of Wildland Fire Effects. *Signal & Image Processing : An International Journal*, *8*, 01-11. https://doi.org/10.5121/sipij.2017.8501

Hu, X., Ban, Y., & Nascetti, A. (2021). Uni-Temporal Multispectral Imagery for Burned Area Mapping with Deep Learning. *Remote Sensing*, *13*(8). https://doi.org/10.3390/rs13081509

Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.

Jain, P., Coogan, S. C. P., Subramanian, S. G., Crowley, M., Taylor, S., & Flannigan, M. D. (2020). A review of machine learning applications in wildfire science and management. *Environmental Reviews*, *28*(4), 478-505. https://doi.org/10.1139/er-2020-0019

Kingma, D., & Ba, J. (2014). Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations*.

Klein, A., Tiao, L. C., Lienart, T., Archambeau, C., & Seeger, M. W. (2020). Model-based Asynchronous Hyperparameter and Neural Architecture Search. *arXiv: Learning*.

Knopp, L., Wieland, M., Rättich, M., Martinis, S., a, a., & bB, B. (2020). A Deep Learning Approach for Burned Area Segmentation with Sentinel-2 Data. *Remote Sensing*.

Langford, Z., Kumar, J., & Hoffman, F. (2018). *Wildfire Mapping in Interior Alaska Using Deep Neural Networks on Imbalanced Datasets*. https://doi.org/10.1109/ICDMW.2018.00116

Liu, J., Heiskanen, J., Maeda, E. E., & Pellikka, P. K. E. (2018). Burned area detection based on Landsat time series in savannas of southern Burkina Faso. *International Journal of Applied Earth Observation and Geoinformation*, *64*, 210-220. https://doi.org/https://doi.org/10.1016/j.jag.2017.09.011

Lizundia-Loiola, J., Otón, G., Ramo, R., & Chuvieco, E. (2020). A spatio-temporal active-fire clustering approach for global burned area mapping at 250 m from MODIS data. *Remote Sensing of Environment*, *236*, 111493. https://doi.org/https://doi.org/10.1016/j.rse.2019.111493

Meddens, A. J. H., Kolden, C. A., & Lutz, J. A. (2016). Detecting unburned areas within wildfire perimeters using Landsat and ancillary data across the northwestern United States. *Remote Sensing of Environment*, *186*, 275-285. https://doi.org/https://doi.org/10.1016/j.rse.2016.08.023

Mellor, A., Boukir, S., Haywood, A., & Jones, S. (2015). Exploring issues of training data imbalance and mislabelling on random forest performance for large area land cover classification using the ensemble margin. *ISPRS Journal of Photogrammetry and Remote Sensing*, *105*, 155-168. https://doi.org/https://doi.org/10.1016/j.isprsjprs.2015.03.014

Milesi, A. (2017). *Pytorch-UNet*. In (Version GNU General Public License v3.0) Github. https://github.com/milesial

Millard, K., & Richardson, M. (2015). On the Importance of Training Data Sample Selection in Random Forest Image Classification: A Case Study in Peatland Ecosystem Mapping. *Remote Sensing*, *7*(7). https://doi.org/10.3390/rs70708489

Miller, D., Jay, & Thode, D., Andrea. (2006). Quantifying burn severity in a heterogeneous landscape with a relative version of the delta Normalized Burned Ratio (dNBR). *Remote Sensing of Environment*, 66-80.

Miranda, A., Mentler, R., Moletto-Lobos, I., Alfaro, G., Aliaga, L., Balbontin, D., . . . Urrutia, V. (2022). *The Landscape Fire Scars Database: mapping historical burned area and fire severity in Chile*. https://doi.org/10.5194/essd-2021-467

Mithal, V., Nayak, G., Khandelwal, A., Kumar, V., Nemani, R., & Oza, N. C. (2018). Mapping Burned Areas in Tropical Forests Using a Novel Machine Learning Framework. *Remote Sensing*, *10*(1). https://doi.org/10.3390/rs10010069

Mommert, M., Sigel, M., Neuhausler, M., Scheibenreif, L., & Borth, D. (2020). Characterization of Industrial Smoke Plumes from Remote Sensing Data. *Workshop Tackling Climate Change with Machine Learning NeurIPS 2020*. (NeurIPS 2020)

Morfitt, R., Barsi, J., Levy, R., Markham, B., Micijevic, E., Ong, L., . . . Vanderwerff, K. (2015). Landsat-8 Operational Land Imager (OLI) Radiometric Performance On-Orbit. *Remote Sensing*, *7*(2). https://doi.org/10.3390/rs70202208

Oumar, Z. (2015). Fire scar mapping for disaster response in KwaZulu-Natal South Africa using Landsat 8 imagery. *South African Journal of Geomatics*, *4*, 309-316. https://doi.org/10.4314/sajg.v4i3.11

Padilla, M., Olofsson, P., Stehman, S. V., Tansey, K., & Chuvieco, E. (2017). Stratification and sample allocation for reference burned area data. *Remote Sensing of Environment*, *203*, 240-255. https://doi.org/https://doi.org/10.1016/j.rse.2017.06.041

Padilla, M., Stehman, S. V., Ramo, R., Corti, D., Hantson, S., Oliva, P., . . . Chuvieco, E. (2015). Comparing the accuracies of remote sensing global burned area products using stratified random sampling and estimation. *Remote Sensing of Environment*, *160*, 114-121. https://doi.org/https://doi.org/10.1016/j.rse.2015.01.005

Patel, N., & Kaushal, B. (2010). Improvement of user's accuracy through classification of principal component images and stacked temporal images. *Geo-spatial Information Science*, *13*(4), 243-248. https://doi.org/10.1007/s11806-010-0380-0

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., . . . Louppe, G. (2012). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*.

Pereira, A. A., Pereira, J. M. C., Libonati, R., Oom, D., Setzer, A. W., Morelli, F., . . . De Carvalho, L. M. (2017). Burned Area Mapping in the Brazilian Savanna Using a One-Class Support Vector Machine Trained by Active Fires. *Remote Sensing*, *9*(11). https://doi.org/10.3390/rs9111161

Perez, L., & Wang, J. (2017). The Effectiveness of Data Augmentation in Image Classification using Deep Learning.

Pinto, M. M., Libonati, R., Trigo, R. M., Trigo, I. F., & DaCamara, C. C. (2020). A deep learning approach for mapping and dating burned areas using temporal sequences of satellite images. *ISPRS Journal of Photogrammetry and Remote Sensing*, *160*, 260-274. https://doi.org/https://doi.org/10.1016/j.isprsjprs.2019.12.014

Pinto, M. M., Trigo, R. M., Trigo, I. F., & DaCamara, C. C. (2021). A Practical Method for High-Resolution Burned Area Monitoring Using Sentinel-2 and VIIRS. *Remote Sensing*, *13*(9). https://doi.org/10.3390/rs13091608

Pytorch. (2022). *BCEWITHLOGITSLOSS*. Pytorch. Retrieved 12/03/2022 from https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html#bcewithlogitsloss

Ronneberger, O., Fischer, P., & Brox, T. (2015). *U-Net: Convolutional Networks for Biomedical Image Segmentation* (Vol. 9351). https://doi.org/10.1007/978-3-319-24574-4_28

Roteta, E., Bastarrika, A., Padilla, M., Storm, T., & Chuvieco, E. (2019). Development of a Sentinel-2 burned area algorithm: Generation of a small fire database for sub-Saharan Africa. *Remote Sensing of Environment*, *222*, 1-17. https://doi.org/https://doi.org/10.1016/j.rse.2018.12.011

Roy, D. P., Huang, H., Boschetti, L., Giglio, L., Yan, L., Zhang, H. H., & Li, Z. (2019). Landsat-8 and Sentinel-2 burned area mapping - A combined sensor multi-temporal change detection approach. *Remote Sensing of Environment*, *231*, 111254. https://doi.org/https://doi.org/10.1016/j.rse.2019.111254

Sertel, E., & Alganci, U. (2016). Comparison of pixel and object-based classification for burned area mapping using SPOT-6 images. *Geomatics, Natural Hazards and Risk*, *7*(4), 1198-1206. https://doi.org/10.1080/19475705.2015.1050608

Shamsoshoara, A., Afghah, F., Razi, A., Zheng, L., Fulé, P. Z., & Blasch, E. (2021). Aerial imagery pile burn detection using deep learning: The FLAME dataset. *Computer Networks*, *193*, 108001. https://doi.org/https://doi.org/10.1016/j.comnet.2021.108001

Storey, J. C., Scaramuzza, P. L., & Schmidt, G. L. (2005). LANDSAT 7 SCAN LINE CORRECTOR-OFF GAP-FILLED PRODUCT DEVELOPMENT.

Thariqa, P., Sitanggang, I., & Syaufina, L. (2016). Comparative analysis of spatial decision tree algorithms for burned area of peatland in Rokan Hilir Riau. *14*, 684-691. https://doi.org/10.12928/TELKOMNIKA.v14i1.3540

Trisakti, B., Nugroho, U., & Zubaidah, A. (2017). TECHNIQUE FOR IDENTIFYING BURNED VEGETATION AREA USING LANDSAT 8 DATA. *International*

Journal of Remote Sensing and Earth Sciences (IJReSES), *13*, 121. https://doi.org/10.30536/j.ijreses.2016.v13.a2447

U.S. Geological Survey, D. o. t. I. (2020a). Landsat 4-7 Collection 1 (C1). Surface Reflectance (LEDAPS). Product Guide. In: USGS. U.S. Geological Survey, D. o. t. I. (2020b). Landsat 8 Collection 1 (C1). Land Surface Reflectance Code (LaSRC). Product Guide. In: USGS.

Wei, S., Zhang, H., Wang, C., Wang, Y., & Xu, L. (2019). Multi-Temporal SAR Data Large-Scale Crop Mapping Based on U-Net Model. *Remote Sensing*, *11*. https://doi.org/10.3390/rs11010068

Zhang, P., Nascetti, A., Ban, Y., & Gong, M. (2019). An implicit radar convolutional burn index for burnt area mapping with Sentinel-1 C-band SAR data. *ISPRS Journal of Photogrammetry and Remote Sensing*, *158*, 50-62. https://doi.org/https://doi.org/10.1016/j.isprsjprs.2019.09.013

Zhao, F., Huang, C., & Zhu, Z. (2015). Use of Vegetation Change Tracker and Support Vector Machine to Map Disturbance Types in Greater Yellowstone Ecosystems in a 1984–2010 Landsat Time Series. *IEEE Geoscience and Remote Sensing Letters*, *12*, 1-5. https://doi.org/10.1109/LGRS.2015.2418159

Zunair, H., & Ben Hamza, A. (2021). Sharp U-Net: Depthwise convolutional network for biomedical image segmentation. *Computers in Biology and Medicine*, *136*, 104699. https://doi.org/https://doi.org/10.1016/j.compbiomed.2021.104699

6 Anexos

6.1 Anexo A

Se presentan de modo referencial distintos tipos de incendios, describiendo los más relevantes. En cada una de las figuras dispuestas, se pueden ver 6 imágenes, donde la primera corresponde a la imagen post-incendio con el vector de la cicatriz sobrepuesto (color azul), la segunda a la imagen pre, la tercera a la imagen post sin el vector de la cicatriz sobrepuesto, y usando otra combinación de bandas para su visualización, la cuarta a la imagen pre asociada, la quinta muestra la cicatriz de incendio original de la data de referencia, y la sexta calcula el índice RdNBR entre la imagen pre y post, para confirmar que la cicatriz asociada a los registros fuese correcta. Los colores de la imagen con el índice variaban aleatoriamente dentro del rango de colores entre verde amarillo y rojo, sin embargo, se produce un contraste de colores entre los pixeles que se detectan como quemados, con un mayor valor, y los que no, dependiendo de la sensibilidad del índice, que se define en la Ec. 2.2.

6.1.1 Más de un incendio

En la Figura 6.1 se presentan casos donde había más de un incendio. En estos casos, se debía eliminar el registro para la base de datos que alimentaría al modelo, puesto que este requiere aprender de imágenes acompañadas de *labels* o data de referencia donde se reconozcan todas las superficies de incendios presentes de la imagen.

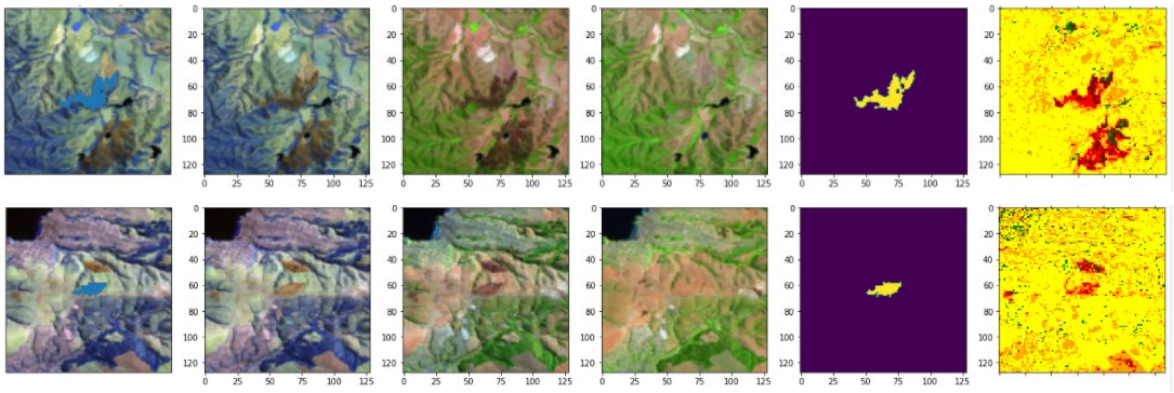


Figura 6.1 Ejemplos donde había más de un incendio

El último caso, mostrado en la Figura 6.2, es algo que también ocurría recurrentemente, en que dos incendios dentro de una imagen se obtenían en dos cicatrices distintas según su cercanía al punto central. Por el motivo ya mencionado, fueron eliminadas igualmente.

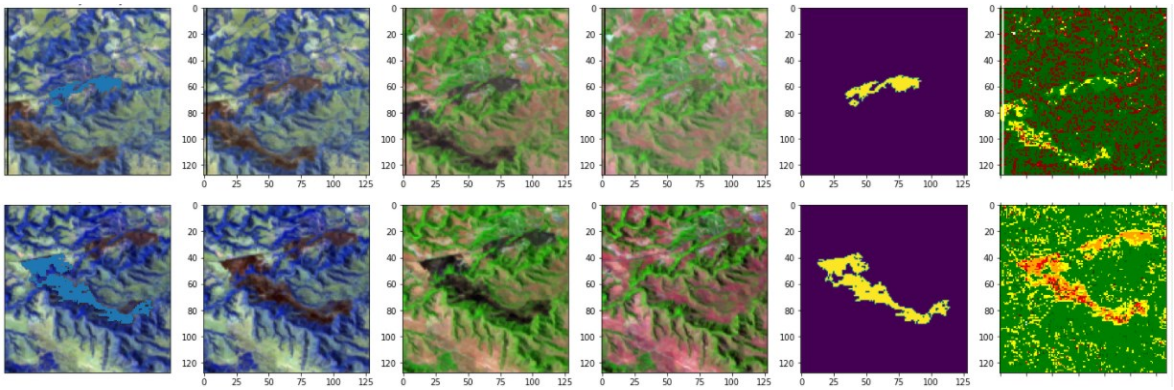


Figura 6.2 Ejemplo de más de un incendio tomados individualmente

6.1.2 Incendios con dos polígonos

En la Figura 6.3 se ilustran incendios conformados por más de un polígono y cómo todoes ellos se acogieron, lo cual se deseaba para el modelo, con el fin de que este pueda reconocer todas las áreas quemadas identificadas como incendios dentro de una imagen satelital entregada.

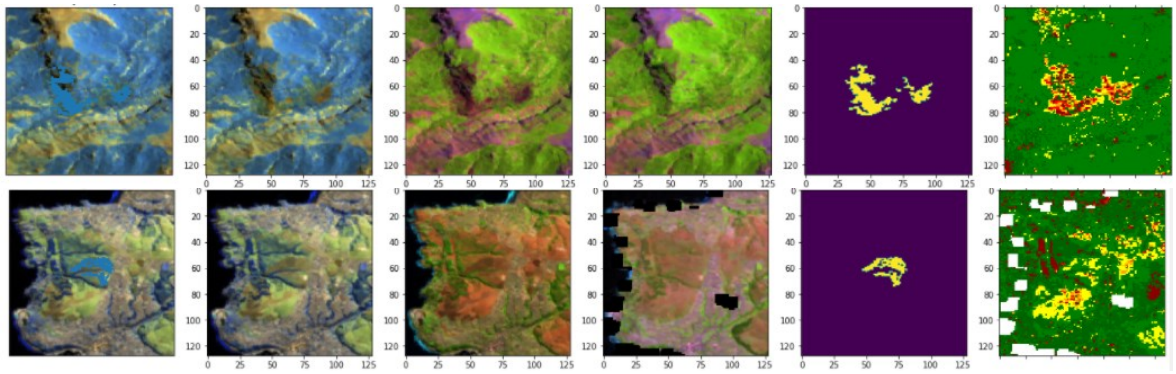


Figura 6.3 Incendios con más de un polígono

6.1.3 Requema

La Figura 6.4 presenta ejemplos de requemas, las cuales ocurren frecuentemente, y solo pueden ser captados desde un enfoque multitemporal.

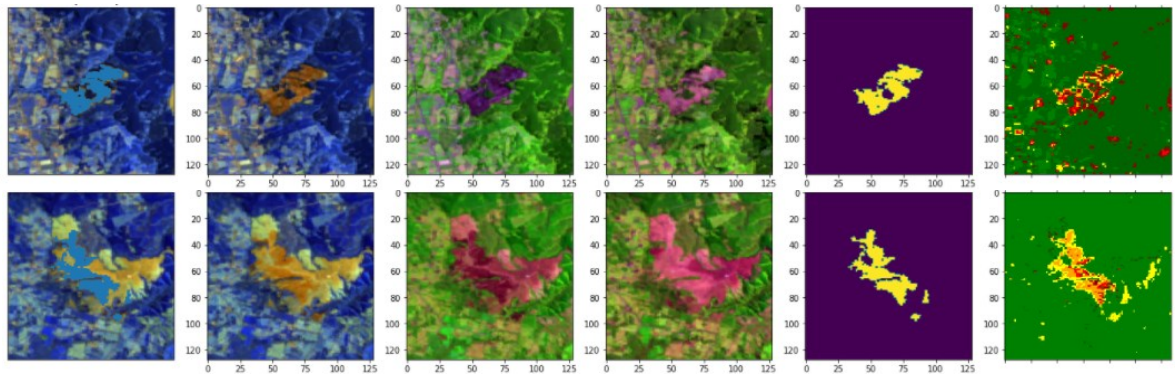


Figura 6.4 Ejemplos de requemas

6.1.4 Bandeo

Este efecto mostrado en la Figura 6.5, también se conoce como *wedge-shaped scan-to-scan gaps (Storey et al., 2005)*, o como "bandeado", que ocurre en ciertas imágenes L7. Esto se logra eliminar desplazando los histogramas parciales de la imagen, es decir, consiguiendo que el valor promedio y la desviación típica sea la misma para toda la banda, sin embargo, esto no se realizó, puesto que el ruido podría aportar a la generalización del modelo, además de que esto sucedió solo para el dataset 128 de lo que se tiene registro, y con una baja frecuencia.

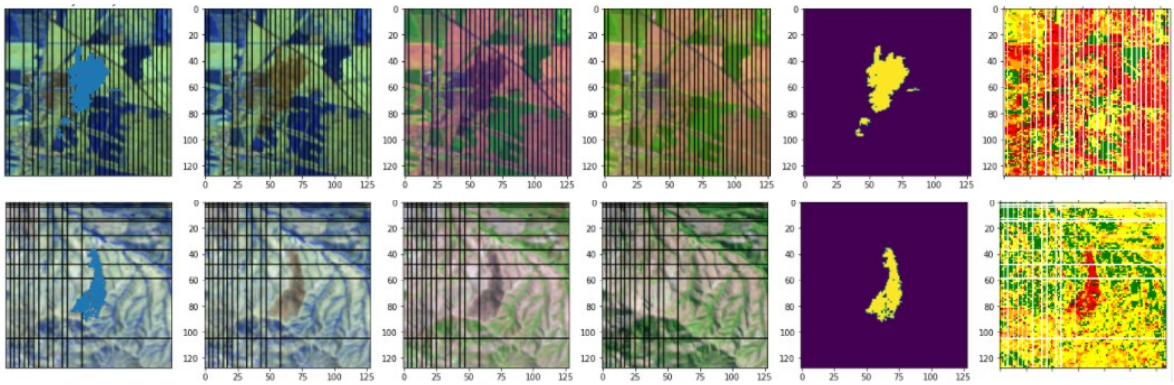


Figura 6.5 Ejemplos con bandeo

También se presentaron imágenes con menos líneas de bandeo, como se ve en la Figura 6.6.

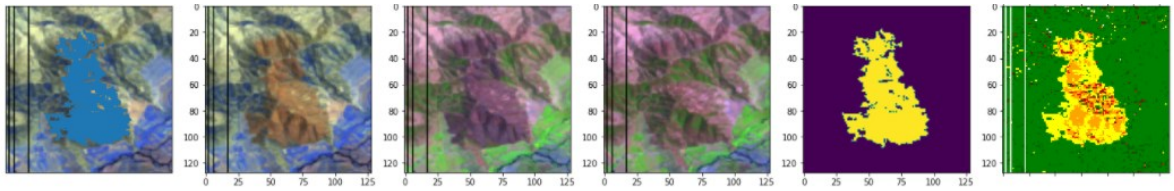


Figura 6.6 Ejemplo bandeo leve

6.1.5 Con quemas agrícolas

Otra cobertura de suelo recurrente en ambos dataset incluía quemas agrícolas, que en algunos casos se descontrolaban y producían un incendio, y en otros no se deseaba reconocer como incendio. Las quemas agrícolas se identifican por mantener la misma forma que un cultivo visible en la imagen pre-incendio, con colores uniformes. En la Figura 6.7 se ven ejemplos de esto, donde no se incluían las áreas de quema agrícola dentro de la cicatriz. Estos registros son de gran valor para generar un modelo que reconozca o no estas superficies como incendios, para lo cual se tendrían que incluir más registros de estas características o no dependiendo de lo que se desee.

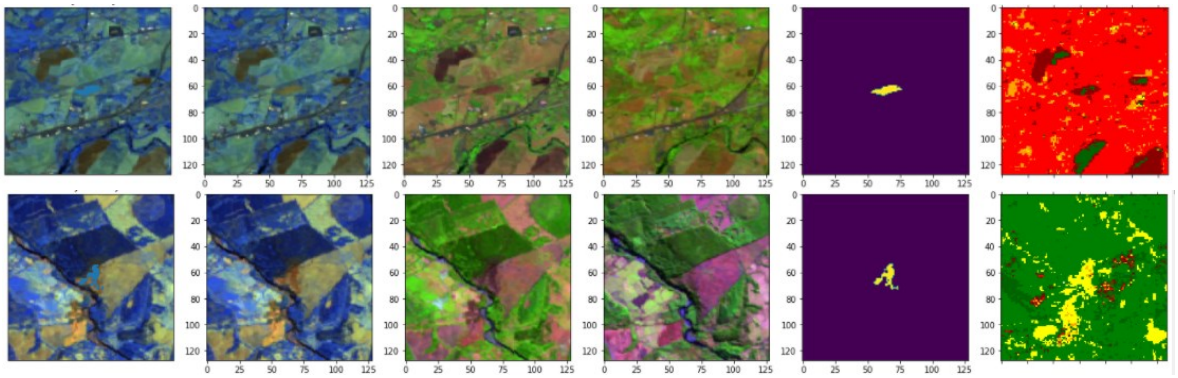


Figura 6.7 Ejemplo de quemas agrícolas

6.1.6 Interpolación realizada usando Nearest-neighbor Interpolator

La interpolación realizada con *Nearest-neighbor Interpolator*, se demuestra en la Figura 6.8 La primera imagen contiene un caso inusual, con bastante superficie sin datos, mientras que las últimas dos imágenes solo prescinden de datos en los bordes, lo cual era más recurrente.

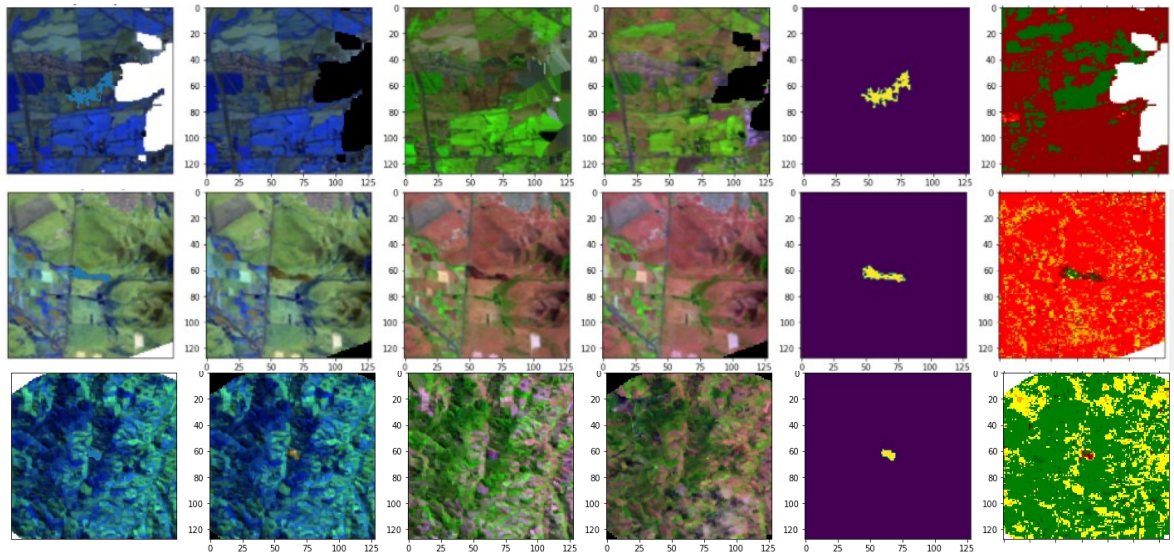


Figura 6.8 Ejemplos de la aplicación de NearestNeighbours Interpolator

6.1.7 Imágenes con recuperación de la vegetación

En este caso mostrado en la Figura 6.9 se nota que el cuadrilátero quemado de la imagen pre, en la imagen post se encuentra de nuevo con vegetación, como una zona de cultivo. Estos eventos son también interesantes para el estudio dadas las diferencias en el contexto de las imágenes, lo cual repercute en los valores de las bandas de las imágenes, esperando mayores cambios en las bandas NDVI y NBR.

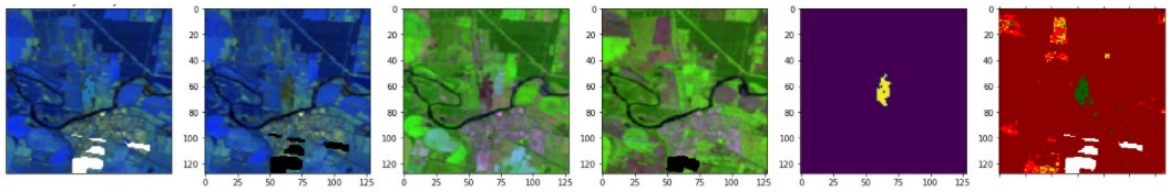


Figura 6.9 Ejemplo de recuperación de vegetación

6.1.8 Paisaje con abudante vegetación

En la Figura 6.10 se ve un ejemplo de un paisaje con abundante vegetación.

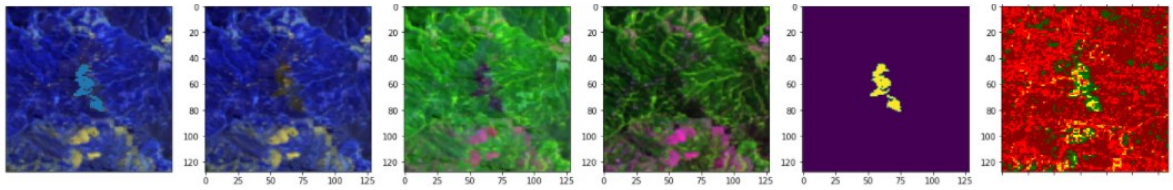


Figura 6.10 Ejemplo de bosque en la cobertura de suelo

6.1.9 Costeros

En la Figura 6.11 se ven paisajes con incendios en zonas costeras. Además de esto, se ve nuevamente la actuación del *Nearest-neighbor Interpolator*.

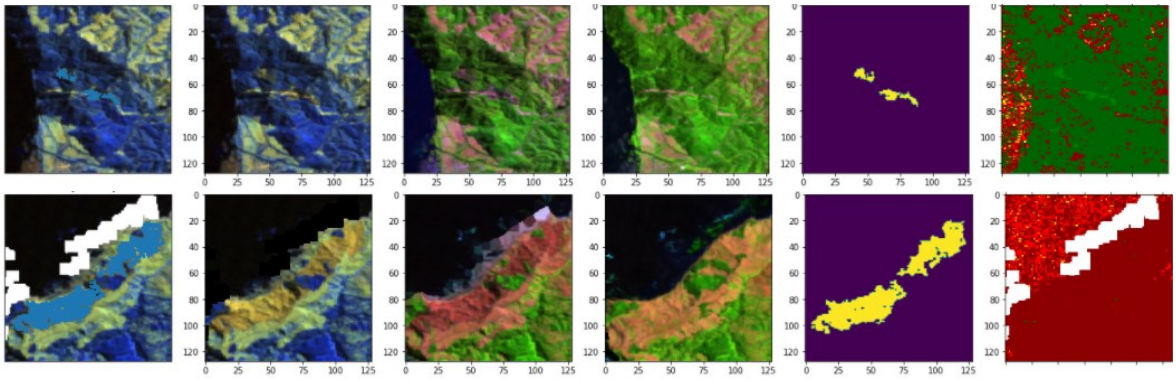


Figura 6.11 Ejemplo de incendios en zonas costeras

6.1.10 Con cuerpos de agua

Los cuerpos de agua en las imágenes que aprenda el modelo son relevantes, puesto que en reiterados casos la sequía del agua produce un alto índice RdNBR, que se asocia con quema, por lo que incluir imágenes como las de la Figura 6.12 aporta a no tener errores de reconocimiento de sequías como superficie quemada.

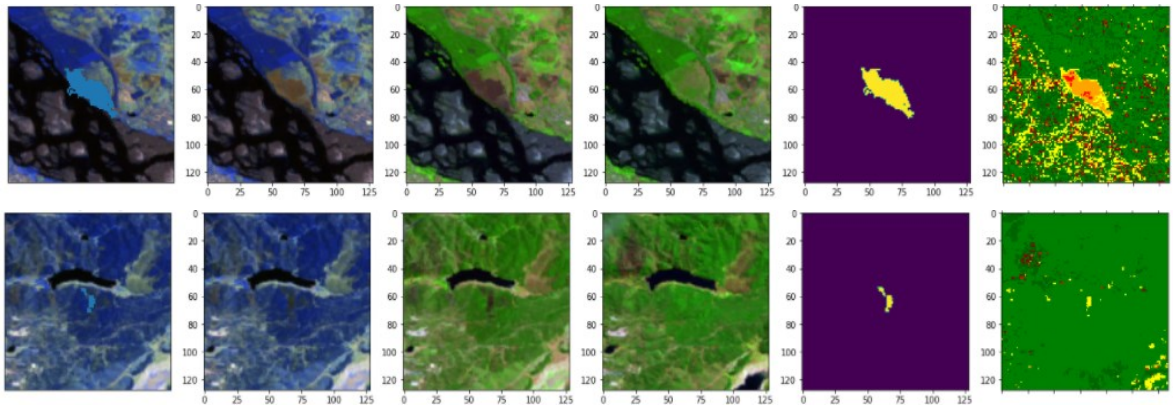


Figura 6.12 Ejemplo de incendios con cuerpos de agua en la imagen

6.1.11 Posibles errores de clasificación filtrados

La Figura 6.13 muestra un caso en que el área quemada identificada probablemente solo fue una quema agrícola. Esto se nota observando el vector triangular amarillo identificado como quema, que posee la misma forma que tenía delimatada el triángulo uniforme verde de la imagen pre-incendio. Un incendio se caracteriza por ser descontrolado, con formas y manchas heterogéneas respecto a la situación previa, lo cual en este caso no ocurre. Este error de clasificación puede originarse desde el llamado a CONAF para reportar un incendio, y que luego ellos lo hayan registrado como tal, pero que en realidad haya sido una quema agrícola.

Para el dataset 128 se eliminaron los registros que fueron identificados como posibles quemas agrícolas, pero se reconoce como un criterio frágil el que diferencia una quema agrícola de un incendio, que puede originarse desde su descontrol.

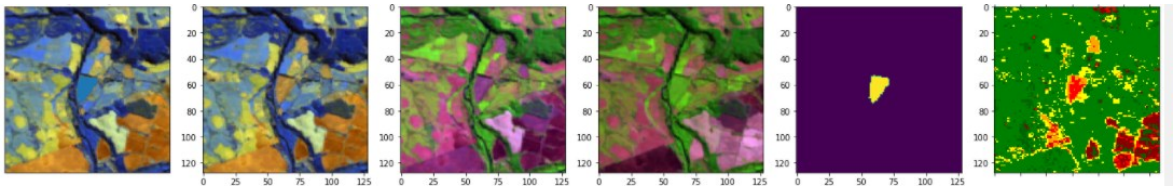


Figura 6.13 Ejemplo de clasificación errada

6.1.12 Registros con incendios previos

Algunas imágenes contenían incendios previos, como se ve en la Figura 6.14. Al igual que las requemas, estos solo se pueden detectar con un modelo multitemporal, incluyendo imágenes pre y post.

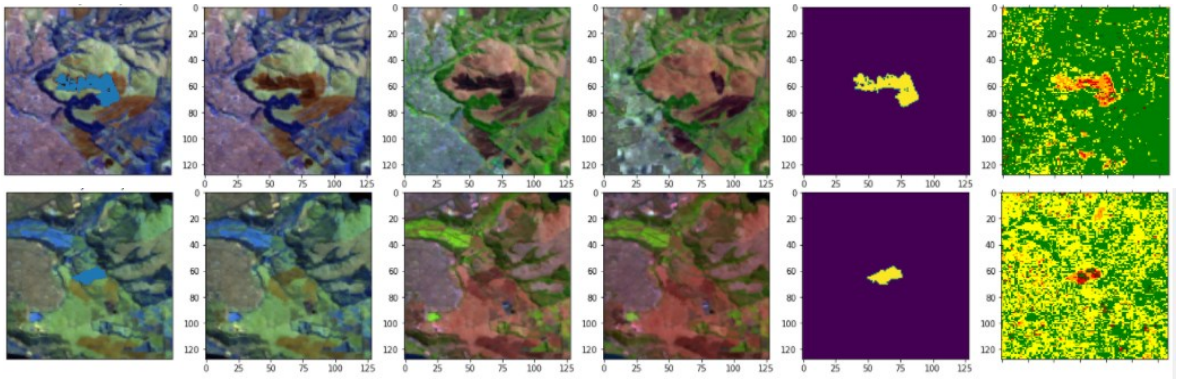


Figura 6.14 Ejemplo de imágenes con incendios previos

6.1.13 Registros con errores de omisión

Por último, como se ha mencionado, el dataset utilizado se formó bajo el principio de omitir por sobre sobreestimar las áreas quemadas, por lo que dada las limitaciones del algoritmo utilizado, en muchos casos se perdían pixeles debido a este principio, mientras que en otros se sobreestimaban igualmente, pero en menor medida. En la Figura 6.15 se ve un claro ejemplo de esto, donde posteriormente si el modelo clasifica correctamente los pixeles quemados que no fueron inicialmente considerados, se medirá como un error de comisión.

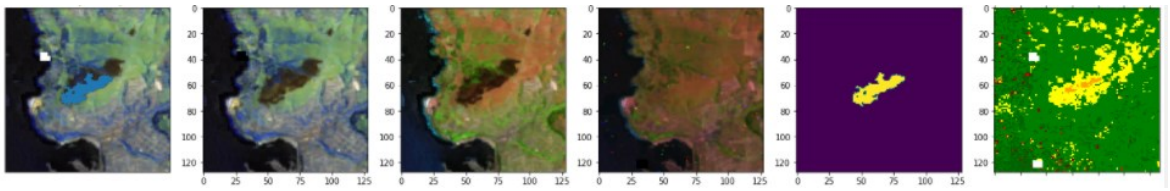


Figura 6.15 Ejemplo de registros con omisión

6.2 Anexo B

El flujo de proceso de obtención de la data inicial se presenta en la Figura 6.16.

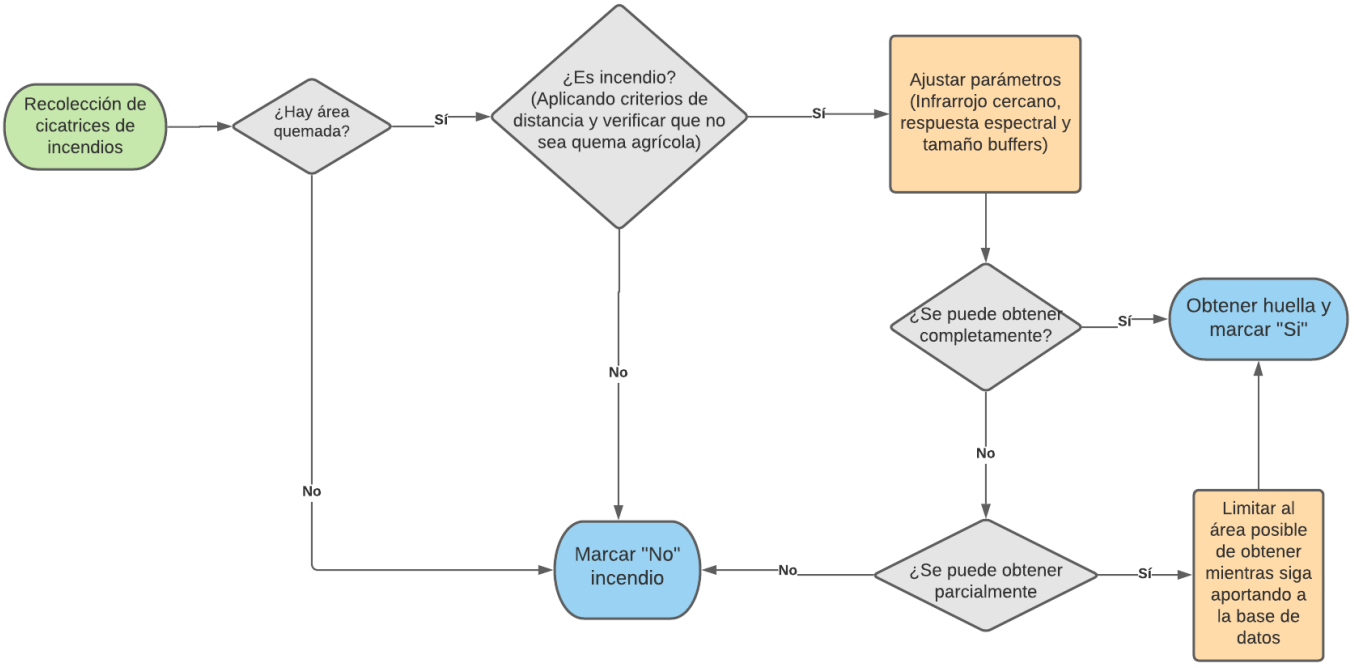


Figura 6.16 Flujo de proceso de obtención del dataset original

6.3 Anexo C

6.3.1 Código

6.3.1.1 Recorte de imágenes

6.3.1.1.1 AS

En primer lugar se presenta la función para el recorte del dataset AS en la Figura 6.17 , mientras que en la Figura 6.18 para el dataset 128.

```
import os, csv, geopandas as gpd, rasterio as rio, rioxarray as rxr
from osgeo import gdal
from pathlib import Path
```

```
def to_shp(filename,destin_folder):
    '''
    transforms the raster to shp to enable the posterior clipping
    '''
    inDs=gdal.Open(filename)
    outDs = gdal.Translate('{}.xyz'.format(destin_folder+"/"+Path(filename).stem), inDs, format='XYZ', creationOptions=["ADD_HEADER_LINE=YES"])
    outDs = None
    try:
        os.remove('{}.csv'.format(destin_folder+"/"+Path(filename).stem))
    except OSError:
        pass
    os.rename('{}.xyz'.format(destin_folder+"/"+Path(filename).stem), '{}.csv'.format(destin_folder+"/"+Path(filename).stem))
    return os.system('ogr2ogr -f "ESRI Shapefile" -oo X_POSSIBLE_NAMES=X* -oo Y_POSSIBLE_NAMES=Y*'
                     '-oo KEEP_GEOM_COLUMNS=NO {0}_2.shp {0}.csv'.format(destin_folder+"/"+Path(filename).stem))
```

```python
def clipping(filename, destin_folder, ipname, out_path):  #filename es del archivo tif del Firescar y el ipname de la Imagen Post (IP)
    '''
    Clips the satellite raster to the firescar size using the firescar binary raster for the georeference
    filename: firescar raster path
    destin_folder: new shapefile path
    ipname: image pre/post filename path
    out_path: output path destination
    '''
    to_shp(filename,destin_folder)
    fire_boundary_path = destin_folder+"/"+Path(filename).stem+"_2.shp"
    ippath=os.path.join(ipname) #ip name of raster file
    fire_boundary = gpd.read_file(fire_boundary_path)
    #Check crs
    ip_crs=rxr.open_rasterio(ipname).rio.crs
    fire_boundary.crs=ip_crs
    #clipping
    ip=rxr.open_rasterio(ippath, masked=False).squeeze()
    clip = rxr.open_rasterio(ippath).rio.clip(
        fire_boundary.geometry,
        from_disk=True).squeeze()
    #Export
    clip.rio.to_raster(out_path+Path(ipname).stem+"_clip.tif", compress='LZMA', dtype="float64")
```

Figura 6.17 Código función de recorte de imágenes iniciales AS

6.3.1.1.2 128

```python
import os, csv, geopandas as gpd, rasterio as rio, rioxarray as rxr, pandas as pd, numpy as np
from osgeo import gdal
from pathlib import Path
from geopandas import GeoDataFrame
from shapely.geometry import Point
```

```python
def clipping128(filename, ipname, destin_folder, output, destin_folder,size): #destin_folder in str
    '''
    This function clips the satellite image raster to the desired size using another raster containing only the binary firescar
    for the georeference. It uses only firescar rasters inferior to size in at least one of the two axis.

    filename: is the path of the firescar raster, containing the geospatial information required for the clipping
    ipname: stands for image pre/pos, and is the file name of the raster desired to clip to the size dimensions, in this case 128.
    destin_folder: is the destination path for the csv file created with the filename geospatial information
    output: is the path for the clipped raster
    '''
    outDs = gdal.Translate('{}.xyz'.format(destin_folder+"/"+Path(filename).stem), inDs, format='XYZ', creationOptions=["ADD_HEADER_LINE=YES"])
    outDs = None
    try:
        os.remove('{}.csv'.format(destin_folder+"/"+Path(filename).stem))
    except OSError:
        pass
    os.rename('{}.xyz'.format(destin_folder+"/"+Path(filename).stem), '{}.csv'.format(destin_folder+"/"+Path(filename).stem))
    file_=rxr.open_rasterio(filename)
    if (len(file_.y)<size or len(file_.x)<size):
        inDs=gdal.Open(filename)
        ulx, xres, xskew, uly, yskew, yres  = inDs.GetGeoTransform()
        df=pd.read_csv(destin_folder+"/"+Path(filename).stem+".csv")
        df2=pd.DataFrame(columns=["X","Y"])
        idx=0
        for i in df.values:
            df2.loc[idx, "X"]=float(i[0].split(' ')[0])
            df2.loc[idx, "Y"]=float(i[0].split(' ')[1])
```

```
        idx+=1
    # print(f"initial size: {len(df2.X.unique()), len(df2.Y.unique())}")
    if ((len(df2.X.unique())<size) or (len(df2.Y.unique())<size)):
        if (len(df2.X.unique())<size and len(df2.Y.unique())>=size):
            newX=np.linspace(df2.X.min()-((size-(len(df2.X.unique())))/2)*xres,df2.X.max()+((size-(len(df2.X.unique())))/2)*xres,size)
            newY=np.linspace(df2.Y.min()+(((len(df2.Y.unique())-size))/2)*-yres,df2.Y.max()-(((len(df2.Y.unique())-size))/2)*-yres,size)
        elif (len(df2.Y.unique())<size and len(df2.X.unique())>=size):
            newY=np.linspace(df2.Y.min()-((size-(len(df2.Y.unique())))/2)*-yres,df2.Y.max()+((size-(len(df2.Y.unique())))/2)*-yres,size)
            newX=np.linspace(df2.X.min()+(((len(df2.X.unique())-size))/2)*xres,df2.X.max()-(((len(df2.X.unique())-size))/2)*xres,size)
        elif (len(df2.Y.unique())<size and len(df2.X.unique())<size):
            newX=np.linspace(df2.X.min()-((size-(len(df2.X.unique())))/2)*xres,df2.X.max()+((size-(len(df2.X.unique())))/2)*xres,size)
            newY=np.linspace(df2.Y.min()-((size-(len(df2.Y.unique())))/2)*-yres,df2.Y.max()+((size-(len(df2.Y.unique())))/2)*-yres,size)
    # print(f"new size x,y:{len(newX),len(newY)}")
    xx, yy = np.meshgrid(newX.tolist(), newY.tolist())
    newX = np.array(xx.flatten("C"))
    newY = np.array(yy.flatten("C"))
    df_r=pd.DataFrame(columns=["X", "Y"])
    df_r["X"]=newX
    df_r["Y"]=newY
    geometry = [Point(xy) for xy in zip(df_r.X, df_r.Y)]
    df_f = df_r.drop(['X', 'Y'], axis=1)
    gdf = GeoDataFrame(df_f, crs="EPSG:4326", geometry=geometry)
    #clipping
    ippath=os.path.join(ipname) #name of the raster file
    fire_boundary= gdf
```

```
ip=rxr.open_rasterio(ippath, masked=True).squeeze()
clip = rxr.open_rasterio(ippath).rio.clip(
fire_boundary.geometry,
from_disk=True).squeeze()
print(f"size clip: {len(clip.x), len(clip.y)}")
if (len(clip.x)==128 and len(clip.y)==128):
    #export
    clip.rio.to_raster(output+"/"+Path(ipname).stem+"_clip.tif")
#for issues with the clipping when pixels weren't in the exact border. It clips 1/2 aditional pixel down or left to obtain the right clip.
elif (len(clip.x)==127 or len(clip.y)==127):
    if (len(clip.x)==127 and len(clip.y)==127):
        newX=np.linspace(df2.X.min()-((size-(len(df2.X.unique())))/2)*xres-xres*1/4,df2.X.max()+((size-(len(df2.X.unique())))/2)*xres-xres*1/4,size)
        newY=np.linspace(df2.Y.min()-((size-(len(df2.Y.unique())))/2)*-yres-(-yres*1/4),df2.Y.max()+((size-(len(df2.Y.unique())))/2)*-yres-(-yres*1/4),size)
    elif (len(clip.x)==127 and len(clip.y)==128):
        newX=np.linspace(df2.X.min()-((size-(len(df2.X.unique())))/2)*xres-xres*1/4,df2.X.max()+((size-(len(df2.X.unique())))/2)*xres-xres*1/4,size)
        newY=np.linspace(df2.Y.min()+(((len(df2.Y.unique())-size))/2)*-yres,df2.Y.max()-(((len(df2.Y.unique())-size))/2)*-yres,size)
    elif (len(clip.y)==127 and len(clip.x)==128):
        newY=np.linspace(df2.Y.min()-((size-(len(df2.Y.unique())))/2)*-yres-(-yres*1/4),df2.Y.max()+((size-(len(df2.Y.unique())))/2)*-yres-(-yres*1/4),size)
        newX=np.linspace(df2.X.min()-((size-(len(df2.X.unique())))/2)*xres,df2.X.max()+((size-(len(df2.X.unique())))/2)*xres,size)
# print(f"new size x,y:{len(newX),len(newY)}")
xx, yy = np.meshgrid(newX.tolist(), newY.tolist())
newX = np.array(xx.flatten("C"))
newY = np.array(yy.flatten("C"))
df_r=pd.DataFrame(columns=["X", "Y"])
df_r["X"]=newX
df_r["Y"]=newY
```

```
geometry = [Point(xy) for xy in zip(df_r.X, df_r.Y)]
df_f = df_r.drop(['X', 'Y'], axis=1)
gdf = GeoDataFrame(df_f, crs="EPSG:4326", geometry=geometry)
ippath=os.path.join(ipname) #ip name of the raster file
fire_boundary= gdf
# clipping
ip=rxr.open_rasterio(ippath, masked=True).squeeze()
clip = rxr.open_rasterio(ippath).rio.clip(
fire_boundary.geometry,
from_disk=True).squeeze()
# print(f"size clip_fixed: {len(clip.x), len(clip.y)}")
#Export
if (len(clip.x)==128 and len(clip.y)==128):
    clip.rio.to_raster(output+"/"+Path(ipname).stem+"_clip.tif")
```

Figura 6.18 Código función de recorte imágenes iniciales 128

6.3.1.2 U Net

El código para de la red U Net consta de 5 partes, en secuencia son: Importación de librerías, Data, Modelo, Entrenamiento y Validación. De estas secciones, la importación de librerías, el modelo y la validación son exactamente iguales para los modelos AS y 128. En las secciones de Data y Entrenamiento las diferencias son los datos usados para el preprocesamiento, el *padding*, los datos asociados al Min-Max *scaling,* y el acceso a los datos desde sus respectivos directorios. Por esto, se muestra el código que funciona para ambos modelos, indicando donde van las diferencias comentando: "#MODELO 128", "#MODELO AS", o en el caso del padding, exhibiendo los dos códigos diferentes según el modelo. Aparte de esto, los directorios son lo único que se debe cambiar dependiendo del modelo.

6.3.1.2.1 Librerías

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision
from torch.utils.data import DataLoader, random_split, RandomSampler
from torch.utils.tensorboard import SummaryWriter
from torch import nn, optim
from torch.optim import Adam
from torchvision import transforms
import gc
import argparse
from tqdm.autonotebook import tqdm
from sklearn.metrics import accuracy_score
from sklearn.metrics import jaccard_score
import pandas as pd
import rasterio as rio
import numpy as np

from rasterio.features import rasterize
from osgeo import gdal

import os
import json
from matplotlib import pyplot as plt
from scipy.interpolate import NearestNDInterpolator
```

Figura 6.19 Código importación de librerías

6.3.1.2.2 Dataset

Inicialmente, se leen los datasets y se encuentra la función de preprocesamiento, donde se diferencian los datos para cada modelo. En esta sección se mostrarán directorios de referencia del modelo 128, pero se tendrían que ajustar simplemente en caso de usar el dataset AS. Posteriormente, se indicará la diferencia en la función *__getitem__*, que es donde se realiza el relleno pertinente para cada modelo. El resto del código es igual para ambos modelos.

Data

```
#DISTINTOS DIRECTORIOS PARA DATASETS AS Y 128
biobio_dataset=pd.read_csv("firescarbiobio128/biobio_final_128_10_03.csv")
valparaiso_dataset=pd.read_csv("firescarvalpo128/valparaiso_128_01_02.csv")
```

```
biobio_dataset.drop(columns="Unnamed: 0", axis=1, inplace=True)
valparaiso_dataset.drop(columns="Unnamed: 0", axis=1, inplace=True)
dataset=pd.concat([valparaiso_dataset,biobio_dataset], axis=0, ignore_index=True)
```

```
### MODELO AS
LS_max=[1689.0, 2502.0, 3260.0, 5650.0, 5282.0, 4121.0, 1.0, 1000.0, 1750.0,
        2559.0, 3325.0, 6065.0, 5224.0, 3903.0, 1.0, 1000.0]
LI_min=[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.096415, -598.040, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
        -0.096636, -392.728]
mean_sprepro=[405.631, 594.520, 713.177, 1650.526, 1714.064, 1259.366, 0.48379, 136.774, 421.969,
              669.070, 798.016, 2243.498, 1936.592, 1172.560, 0.48800, 334.406]
### MODELO 128
LS_max=[3560.875,  4148.875,  4781.375,  6832.625,  6143.875,  6263.125, 1, 1000,
        3765.875,  4398.875,  4868.0,  6930.0,  6158.5,  6168.5,  1,  1000]
LI_min=[0.0,  0.0,  0.0,  0.0,  0.0,  0.0,  -0.67387,  -605.442,0.0,  0.0,  0.0,  0.0,  0.0,  0.0,
        -0.77162,  -574.416]
mean_sprepro=[415.578, 642.862, 755.873, 2059.070, 1810.916, 1161.494, 0.45515, 298.499, 416.960,
              656.332, 759.754, 2192.928, 1828.554, 1136.824,0.47112, 336.206]
```

```
def preprocessing(imgdata):
        if (imgdata[k-1]>LS_max[k-1]).any():
            if imgdata[k-1].mean()<LS_max[k-1]:
                imgdata[k-1][imgdata[k-1]>LS_max[k-1]]=imgdata[k-1].mean()
        elif (imgdata[k-1]<LI_min[k-1]).any():
            if imgdata[k-1].mean()>LI_min[k-1]:
                imgdata[k-1][imgdata[k-1]<LI_min[k-1]]=imgdata[k-1].mean()
    return imgdata
```

```
class firescardataset():
    def __init__(self,dataset, subset_size1, subset_size2, subset_size3,subset_size4,mult=1,transform=None):
        """:param dataset: dataset with data filenames from two different regions. The first one from the index
        subset_size1-subset_size2, and the second one from subset_size3-subset_size4. There are 3 columns with
        the required data filenames for each input: "ImPosF"=The image post Fire, "ImgPreF"=The image pre Fire,
        and "FireScar_tif"=The label, in a raster file"""
        self.transform = transform
        # list of image files (pre and post fire), and labels
        # label vector edge coordinates
        self.imgfiles = []
        self.imgprefiles=[]
        self.labels = []
        self.seglabels = []
        imgposfiles = []
        # read in segmentation
```

```
        for i in range(subset_size1,subset_size2):
            segdata = os.path.join("firescarvalpoallsizes/FireScar/", dataset.loc[i,"FireScar_tif"])
            self.seglabels.append(segdata)
            self.imgfiles.append(os.path.join("firescarvalpoallsizes/ImgPosF/",dataset.loc[i,"ImgPosF"]))
            self.imgprefiles.append(os.path.join("firescarvalpoallsizes/ImgPreF/",dataset.loc[i,"ImgPreF"]))
        for i in range(subset_size3,subset_size4):
            self.seglabels.append(os.path.join("firescarbiobioallsizes/FireScar/",dataset.loc[i,"FireScar_tif"]))
            self.imgfiles.append(os.path.join("firescarbiobioallsizes/ImgPosF/",dataset.loc[i,"ImgPosF"]))
            self.imgprefiles.append(os.path.join("firescarbiobioallsizes/ImgPreF/",dataset.loc[i,"ImgPreF"]))
        self.imgfiles = np.array(self.imgfiles)
        self.imgprefiles=np.array(self.imgprefiles)
        self.labels = np.array(self.labels)
        if mult > 1:
            self.imgfiles = np.array([*self.imgfiles] * mult)
            self.imgprefiles = np.array([*self.imgprefiles] * mult)
            self.labels = np.array([*self.labels] * mult)
            self.seglabels = self.seglabels * mult

    def __len__(self):

        return len(self.imgfiles)
```

Figura 6.20 Código Data -1

A continuación, se presentan diferencias entre ambos modelos debido al *padding* utilizado.

Padding AS:

```
    def __getitem__(self, idx):
        idx=idx-1
        imgfile = rio.open(self.imgfiles[idx])
        imgpre=rio.open(self.imgprefiles[idx])
        imgdata1 = np.array([imgfile.read(i) for i in [1,2,3,4,5,6,7,8]])
        imgdatapre=np.array([imgpre.read(i) for i in [1,2,3,4,5,6,7,8]])
        new_array=np.concatenate((imgdata1, imgdatapre), axis=0)

        ds = gdal.Open(self.seglabels[idx])
        myarray = np.array(ds.GetRasterBand(1).ReadAsArray())

        if (np.isfinite(new_array)==False).any():
            mask=np.where(np.isfinite(new_array))
            interp=NearestNDInterpolator(np.transpose(mask), new_array[mask])
            new_array=interp(*np.indices(new_array.shape))

        new_array=preprocessing(new_array,LS_max, LI_min, mean_sprepro)

        ds = gdal.Open(self.seglabels[idx])
        myarray = np.array(ds.GetRasterBand(1).ReadAsArray())

        x=imgdata1.shape[1]
        y=imgdata1.shape[2]
        imgdata=new_array

        size=128
        if (x<size or y<size):
            if (x%2==1 and y%2==1): #if it's odd
                new_array=np.pad(imgdata, ((0,0),(int((size-x)/2-1/2),int((size-x)/2+1/2)),(int((size-y)/2+1/2),int((size-y)/2-1/2))), "constant")
```

```
        elif (x%2==1 and y%2==0):
            new_array=np.pad(imgdata, ((0,0),(int((size-x)/2-1/2),int((size-x)/2+1/2)),(int((size-y)/2),int((size-y)/2))), "constant")
        elif (x%2==0 and y%2==1):
            new_array=np.pad(imgdata, ((0,0),(int((size-x)/2),int((size-x)/2)),(int((size-y)/2+1/2),int((size-y)/2-1/2))), "constant")
        elif (x%2==0 and y%2==0):
            new_array=np.pad(imgdata, ((0,0),(int((size-x)/2),int((size-x)/2)),(int((size-y)/2),int((size-y)/2))), "constant")

    x,y=myarray.shape

    if (x<size or y<size):
        if (x%2==1 and y%2==1): #if it's odd
            myarray=np.pad(myarray, ((int((size-x)/2-1/2),int((size-x)/2+1/2)),(int((size-y)/2+1/2),int((size-y)/2-1/2))), "constant")
        elif (x%2==1 and y%2==0):
            myarray=np.pad(myarray, ((int((size-x)/2-1/2),int((size-x)/2+1/2)),(int((size-y)/2),int((size-y)/2))), "constant")
        elif (x%2==0 and y%2==1):
            myarray=np.pad(myarray, ((int((size-x)/2),int((size-x)/2)),(int((size-y)/2+1/2),int((size-y)/2-1/2))), "constant")
        elif (x%2==0 and y%2==0):
            myarray=np.pad(myarray, ((int((size-x)/2),int((size-x)/2)),(int((size-y)/2),int((size-y)/2))), "constant")

    sample = {'idx': idx,
          'img': new_array,
          'fpt': myarray,
          'imgfile': self.imgfiles[idx]}
    if self.transform:
        sample = self.transform(sample)
    return sample
```

Figura 6.21 Código Data -2, padding AS

Padding 128:

```
def __getitem__(self, idx):
    idx=idx-1
    imgfile = rio.open(self.imgfiles[idx])
    imgpre=rio.open(self.imgprefiles[idx])
    imgdata1 = np.array([imgfile.read(i) for i in [1,2,3,4,5,6,7,8]])
    imgdatapre=np.array([imgpre.read(i) for i in [1,2,3,4,5,6,7,8]])
    imgdata=np.concatenate((imgdata1, imgdatapre), axis=0)

    if (np.isfinite(imgdata)==False).any():
        mask=np.where(np.isfinite(imgdata))
        interp=NearestNDInterpolator(np.transpose(mask), imgdata[mask])
        imgdata=interp(*np.indices(imgdata.shape))

    ds = gdal.Open(self.seglabels[idx])
    myarray = np.array(ds.GetRasterBand(1).ReadAsArray())

    x=imgdata1.shape[1]
    y=imgdata1.shape[2]

    #FireScar padding to 128 in case is not that size
    x,y=myarray.shape
    #only to equalize to 128x128 images or it could be to image size
    ulx_i, lry_i, lrx_i, uly_i=imgfile.bounds
    ulx, xres, xskew, uly, yskew, yres  = ds.GetGeoTransform()
    lrx = ulx + (ds.RasterXSize * xres)
    lry = uly + (ds.RasterYSize * yres)
    left=round((ulx-ulx_i)/xres)    #np.pad(a, up, down, left, right)
    right=round((lrx_i-lrx)/xres)
    up=round((uly-uly_i)/yres)
    down=round((lry_i-lry)/yres)
```

```
        myarray=np.pad(myarray,((up, down),(left,right)),"constant")

        imgdata=preprocessing(imgdata,LS_max, LI_min, mean_sprepro)

        sample = {'idx': idx,
              'img': imgdata,
              'fpt': myarray,
              'imgfile': self.imgfiles[idx]}
        if self.transform:
            sample = self.transform(sample)
        return sample
```

Figura 6.22 Código Data -2, padding 128

Resto de la sección Data:

Luego de aquel cambio, el resto del código es igual en esta sección para ambos modelos, salvo los valores de mínimos y máximos como se indicará en la misma imagen.

```
class ToTensor(object):
    """Convert ndarrays in sample to Tensors."""
    def __call__(self, sample):
        """
        :param sample: sample to be converted to Tensor
        :return: converted Tensor sample
        """
        out = {'idx': sample['idx'],
        'img': torch.from_numpy(sample['img'].copy()),
        'fpt': torch.from_numpy(sample['fpt'].copy()),
        'imgfile': sample['imgfile']}
        return out
class Randomize(object):
    """Randomize image orientation including rotations by integer multiples of
       90 deg, (horizontal) mirroring, and (vertical) flipping."""
    def __call__(self, sample):
        """
        :param sample: sample to be randomized
        :return: randomized sample
        """
        imgdata = sample['img']
        fptdata = sample['fpt']
        idx=sample["idx"]
        # mirror horizontally
        mirror = np.random.randint(0, 2)
        if mirror:
            imgdata = np.flip(imgdata, 2)
            fptdata = np.flip(fptdata, 1)
        # flip vertically
        flip = np.random.randint(0, 2)
        if flip:
```

```
        if flip:
            imgdata = np.flip(imgdata, 1)
            fptdata = np.flip(fptdata, 0)
        # rotate by [0,1,2,3]*90 deg
        rot = np.random.randint(0, 4)
        if rot:
            imgdata = np.rot90(imgdata, rot, axes=(1,2))
            fptdata = np.rot90(fptdata, rot, axes=(0,1))

        return {'idx': sample['idx'],
                'img': imgdata.copy(),
                'fpt': fptdata.copy(),
                'imgfile': sample['imgfile']}
class Normalize(object):
    """Normalize pixel values to the range [0, 1] measured using minmax-scaling"""
    def __init__(self):
        #MODELO AS
        self.channel_min=np.array([0.0, 0.0, 0.0, 17.0, 7.0, 0.0, -0.096153, -597.968,
                                   0.0, 0.0, 0.0, 0.0, 8.0, 0.0, -0.096629, -392.023])
        self.channel_max=np.array([1689.0, 2502.0, 3260.0, 5650.0, 5282.0, 4121.0, 1.0, 1000.0, 1750.0,
                                   2559.0, 3325.0, 6065.0, 5224.0, 3903.0, 1.0, 1000.0])
        #MODELO 128
        self.channel_min=np.array([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.67379, -605.416,
                                   0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.77142, -572.139])
        self.channel_max=np.array([3560.0, 4147.0, 4780.0, 6832.0, 6143.0, 6261.0, 1.0, 1000.0,
                                   3765.0, 4398.0, 4866.0, 6930.0, 6157.0, 6167.0, 1.0, 1000.0])
    def __call__(self, sample):
        """
        :param sample: sample to be normalized
        :return: normalized sample
        """
```

```
        sample['img'] = (sample['img']-self.channel_min.reshape(
            sample['img'].shape[0], 1, 1))/(self.channel_max.reshape(
            sample['img'].shape[0], 1, 1)-self.channel_min.reshape(
            sample['img'].shape[0], 1, 1))
        return sample

def create_dataset(*args, apply_transforms=True, **kwargs):
    """
    :param apply_transforms: if `True`, apply available transformations
    :return: data set"""
    if apply_transforms:
        data_transforms = transforms.Compose([
            Normalize(),
            Randomize(),
            ToTensor()
           ])
    else:
        data_transforms = None

    data = firescardataset(*args, **kwargs,
                                        transform=data_transforms)

    return data
```

Figura 6.23 Código Data -3

6.3.1.2.3 Modelo

Modelo

```
torch.cuda.empty_cache()
gc.collect()
```

```
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
```

```
%%capture
class DoubleConv(nn.Module):
#     """(convolution => [BN] => ReLU) * 2"""

    def __init__(self, in_channels, out_channels, mid_channels=None):
        super().__init__()
        if not mid_channels:
            mid_channels = out_channels
        self.double_conv = nn.Sequential(
            nn.Conv2d(in_channels, mid_channels, kernel_size=3, padding=1),
            nn.BatchNorm2d(mid_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(mid_channels, out_channels, kernel_segment=3, padding=1),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True)
        )

    def forward(self, x):
        return self.double_conv(x)

class Down(nn.Module):
#     """Downscaling with maxpool then double conv"""
```

```
    def __init__(self, in_channels, out_channels):
        super().__init__()
        self.maxpool_conv = nn.Sequential(
            nn.MaxPool2d(2),
            DoubleConv(in_channels, out_channels)
        )

    def forward(self, x):
        return self.maxpool_conv(x)


class Up(nn.Module):
#     """Upscaling then double conv"""

    def __init__(self, in_channels, out_channels, bilinear=True):
        super().__init__()

        # if bilinear, use the normal convolutions to reduce the number of channels
        if bilinear:
            self.up = nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True)
            self.conv = DoubleConv(in_channels, out_channels, in_channels // 2)
        else:
            self.up = nn.ConvTranspose2d(in_channels , in_channels // 2, kernel_size=2, stride=2)
            self.conv = DoubleConv(in_channels, out_channels)


    def forward(self, x1, x2):
        x1 = self.up(x1)
        # input is CHW
        diffY = x2.size()[2] - x1.size()[2]
        diffX = x2.size()[3] - x1.size()[3]
```

```
        x1 = F.pad(x1, [diffX // 2, diffX - diffX // 2,
                        diffY // 2, diffY - diffY // 2])
        x = torch.cat([x2, x1], dim=1)
        return self.conv(x)

class OutConv(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(OutConv, self).__init__()
        self.conv = nn.Conv2d(in_channels, out_channels, kernel_size=1)

    def forward(self, x):
        return self.conv(x)

class UNet(nn.Module):
    def __init__(self, n_channels, n_classes, bilinear=True):
        super(UNet, self).__init__()
        self.n_channels = n_channels
        self.n_classes = n_classes
        self.bilinear = bilinear
        self.inc = DoubleConv(n_channels, 128)
        self.down1 = Down(128, 256)
        self.down2 = Down(256, 512)
        self.down3 = Down(512, 1024)
        factor = 2 if bilinear else 1
        self.down4 = Down(1024, 2048 // factor)
        self.up1 = Up(2048, 1024 // factor, bilinear)
        self.up2 = Up(1024, 512 // factor, bilinear)
        self.up3 = Up(512, 256 // factor, bilinear)
        self.up4 = Up(256, 128, bilinear)
        self.outc = OutConv(128, n_classes)
```

```
    def forward(self, x):
        x1 = self.inc(x)
        x2 = self.down1(x1)
        x3 = self.down2(x2)
        x4 = self.down3(x3)
        x5 = self.down4(x4)
        x = self.up1(x5, x4)
        x = self.up2(x, x3)
        x = self.up3(x, x2)
        x = self.up4(x, x1)
        logits = self.outc(x)
        return logits


model = UNet(n_channels=16, n_classes=1)
model.to(device)
```

Figura 6.24 Código Modelo U Net

6.3.1.2.4 Entrenamiento

```
print('running on...', device)
```

Training

```
def dice2d(pred, targs):
    pred = pred.squeeze()
    targs = targs.squeeze()
    return 2. * (pred*targs).sum() / (pred+targs).sum()
```

```
def train_model(model, epochs, opt, loss, batch_size):
    """
    :param model: model instance
    :param epochs: (int) number of epochs to be trained
    :param opt: optimizer instance
    :param loss: loss function instance
    :param batch_size: (int) batch size
    :param mult: (int) augmentation factor that amplifies the data * mult"""

    #MODELO AS
    data_train = create_dataset(dataset=dataset,subset_size1=0,subset_size2=885,subset_size3=1106,subset_size4=2250, mult=3)
    data_val = create_dataset(dataset=dataset,subset_size1=885,subset_size2=1106,subset_size3=2250,subset_size4=2535, mult=3)

    #MODELO 128
    data_train = create_dataset(dataset=dataset,subset_size1=0,subset_size2=780,subset_size3=975,subset_size4=1815, mult=3)
    data_val = create_dataset(dataset=dataset,subset_size1=780,subset_size2=975,subset_size3=1815,subset_size4=2024, mult=3)

    train_dl = DataLoader(data_train, batch_size=16, num_workers=0, pin_memory=True) #drop_last=True)
    val_dl = DataLoader(data_val, batch_size=16, num_workers=0, pin_memory=True) # drop_last=True)

    filename=""   # ending of the model filename
```

```
    best_model={}
    best_model["val_loss_total"]=100
    best_dc={}
    best_dc["val_DC"]=0
    # start training
    for epoch in range(epochs):
        model.train()
        #metrics
        dicec_train_acc=[]
        FN_train=[]
        TP_train=[]
        FP_train=[]
#         train_acc_total=0
        train_loss_total = 0
        train_ious = []
        progress = tqdm(enumerate(train_dl), desc="Train Loss: ",
                        total=len(train_dl))
        for i, batch in progress:
            # try:
            x = batch['img'].float().to(device)
            y = batch['fpt'].float().to(device)

            output = model(x)

            # derive binary segmentation map from prediction
            output_binary = np.zeros(output.shape)
            output_binary[output.cpu().detach().numpy() >= 0] = 1

            # derive IoU values
            for j in range(y.shape[0]):
```

```
                z = jaccard_score(y[j].flatten().cpu().detach().numpy(),
                                  output_binary[j][0].flatten())
                if (np.sum(output_binary[j][0]) != 0 and
                    np.sum(y[j].cpu().detach().numpy()) != 0):
                    train_ious.append(z)
                    TP_train.append((output_binary.squeeze()*y.cpu().detach().numpy().squeeze()).sum())
                    FN_train.append(((output_binary.squeeze()==0) & (y.cpu().detach().numpy().squeeze()==1)).sum())
                    FP_train.append(((output_binary.squeeze()==1) & (y.cpu().detach().numpy().squeeze()==0)).sum())
                    dicec_train_acc.append(dice2d(output_binary,y.cpu().detach().numpy()))

            # derive scalar binary labels on a per-image basis
            y_bin = np.array(np.sum(y.cpu().detach().numpy(),
                                    axis=(1,2)) != 0).astype(int)
            pred_bin = np.array(np.sum(output_binary,
                                       axis=(1,2,3)) != 0).astype(int)

            # derive image-wise accuracy for this batch
#             train_acc_total += accuracy_score(y_bin, pred_bin)
            # derive loss
            loss_epoch = loss(output, y.unsqueeze(dim=1))
            train_loss_total += loss_epoch.item()
            progress.set_description("Train Loss: {:.4f}".format(
                train_loss_total/(i+1)))

            # learning
            opt.zero_grad()
            loss_epoch.backward()
            opt.step()

        # logging
        writer.add_scalar("training DC", np.average(dicec_train_acc),epoch)
```

```
        writer.add_scalar("training CE",  np.mean(FP_train)/(np.mean(TP_train)+np.mean(FP_train)), epoch)
        writer.add_scalar("training OE",  np.mean(FN_train)/(np.mean(TP_train)+np.mean(FN_train)), epoch)
        writer.add_scalar("training loss", train_loss_total/(i+1), epoch)
        writer.add_scalar("training iou", np.average(train_ious), epoch)
#         writer.add_scalar("training acc", train_acc_total/(i+1), epoch)
        writer.add_scalar('learning_rate', opt.param_groups[0]['lr'], epoch)
        torch.cuda.empty_cache()

        # evaluation
        model.eval()
        val_loss_total = 0
        val_ious = []
#         val_acc_total = 0

        dicec_eval_acc=[]
        FN_eval=[]
        TP_eval=[]
        FP_eval=[]

        progress = tqdm(enumerate(val_dl), desc="val Loss: ",
                        total=len(val_dl))

        for j, batch in progress:
            x = batch['img'].float().to(device)
            y = batch['fpt'].float().to(device)
            output = model(x)

          # derive loss
            loss_epoch = loss(output, y.unsqueeze(dim=1))
            val_loss_total += loss_epoch.item()
```

```
          # derive binary segmentation map from prediction
            output_binary = np.zeros(output.shape)
            output_binary[output.cpu().detach().numpy() >= 0] = 1

          # derive IoU values
            ious = []
            for k in range(y.shape[0]):
                z = jaccard_score(y[k].flatten().cpu().detach().numpy(),
                        output_binary[k][0].flatten())
                if (np.sum(output_binary[k][0]) != 0 and
                    np.sum(y[k].cpu().detach().numpy()) != 0):
                    val_ious.append(z)
                    TP_eval.append((output_binary.squeeze()*y.cpu().detach().numpy().squeeze()).sum())
                    FN_eval.append(((output_binary.squeeze()==0) & (y.cpu().detach().numpy().squeeze()==1)).sum())
                    FP_eval.append(((output_binary.squeeze()==1) & (y.cpu().detach().numpy().squeeze()==0)).sum())
                    dicec_eval_acc.append(dice2d(output_binary,y.cpu().detach().numpy()))

          # derive scalar binary labels on a per-image basis
            y_bin = np.array(np.sum(y.cpu().detach().numpy(),
                                    axis=(1,2)) != 0).astype(int)
            pred_bin = np.array(np.sum(output_binary,
                                       axis=(1,2,3)) != 0).astype(int)

          # derive image-wise accuracy for this batch
#             val_acc_total += accuracy_score(y_bin, pred_bin)

            progress.set_description("val Loss: {:.4f}".format(
             val_loss_total/(j+1)))

        # logging
        writer.add_scalar("val DC", np.average(dicec_eval_acc),epoch)
```

```
        writer.add_scalar("val CE",  np.mean(FP_eval)/(np.mean(TP_eval)+np.mean(FP_eval)), epoch)
        writer.add_scalar("val OE",  np.mean(FN_eval)/(np.mean(TP_eval)+np.mean(FN_eval)), epoch)
        writer.add_scalar("val loss", val_loss_total/(j+1), epoch)
        writer.add_scalar("val iou", np.average(val_ious), epoch)
#         writer.add_scalar("val acc", val_acc_total/(j+1), epoch)

        print(("Epoch {:d}: train loss={:.3f}, val loss={:.3f}, train iou={:.3f}, val iou={:.3f}, "
               "DC training={:.3f}, val DC={:.3f}").format(epoch+1, train_loss_total/(i+1),
               val_loss_total/(j+1), np.average(train_ious), np.average(val_ious),
               np.average(dicec_train_acc), np.average(dicec_eval_acc)))

        if (val_loss_total/(j+1))<best_model["val_loss_total"]:
            best_model["val_loss_total"]=(val_loss_total/(j+1))
            best_model["epoch"]=epoch
        if (np.average(dicec_eval_acc))>best_dc["val_DC"]:
            best_dc["val_DC"]=np.average(dicec_eval_acc)
            best_dc["epoch"]=epoch

        if epoch % 1 == 0:                                  #comment to not save the model files
            torch.save(model.state_dict(),
            'U_Net/runs/ep{:0d}_lr{:.0e}_bs{:02d}_{:03d}_{}.model'.format(
                args.ep, args.lr, args.bs, epoch, filename))

        writer.flush()
        scheduler.step(val_loss_total/(j+1))
        torch.cuda.empty_cache()
    print("best model: epoch (file): {}, val loss: {}".format(best_model["epoch"], best_model["val_loss_total"]))
    print("best model_dc: epoch (file): {}, val dc: {}".format(best_dc["epoch"], best_dc["val_DC"]))
    return model
```

```
# setup argument parser
parser = argparse.ArgumentParser()
parser.add_argument('-f')
parser.add_argument('-ep', type=int, default=25,
                    help='Number of epochs')
parser.add_argument('-bs', type=int, nargs='?',
                    default=16, help='Batch size')
parser.add_argument('-lr', type=float,
                    nargs='?', default=0.0001, help='Learning rate')
# parser.add_argument('-mo', type=float,nargs='?', default=0.7, help='Momentum')  #for SGD optimizer
args = parser.parse_args()

# setup tensorboard writer
writer = SummaryWriter('U_Net/runs/'+"ep{:0d}_lr{:.0e}_bs{:03d}/".format(args.ep, args.lr, args.bs))

# initialize loss function
loss = nn.BCEWithLogitsLoss()

# initialize optimizer
# opt = optim.SGD(model.parameters(), lr=args.lr, momentum=args.mo)
opt = optim.Adam(model.parameters(), lr=args.lr)

# initialize scheduler
scheduler = optim.lr_scheduler.ReduceLROnPlateau(opt, 'min',factor=0.5, threshold=1e-4, min_lr=1e-6)
```

```
# # run training
# model.load_state_dict(torch.load(
#  "U_Net/runs/ep25_lr1e-03_bs10_mo0.7.model" , map_location=torch.device('cpu')))
train_model(model, args.ep, opt, loss, args.bs)
writer.close()
```

Figura 6.25 Código de entrenamiento U Net

6.3.1.2.5 Evaluación

Evaluation

```
# # model.load_state_dict(model_path)
```

```
%%capture
np.random.seed(3)
torch.manual_seed(3)

# load data
data_val = create_dataset(dataset=evald,subset_size1=0,subset_size2=50,subset_size3=50,subset_size4=100, mult=1)

batch_size = 1 # 1 to create diagnostic images, any value otherwise
all_dl = DataLoader(data_val, batch_size=batch_size)#, shuffle=True)
progress = tqdm(enumerate(all_dl), total=len(all_dl))

dicec_eval_acc=[]
FN_eval=[]
TP_eval=[]
FP_eval=[]
comission=[]
omission=[]
cont=0
model.eval()

# define loss function
loss_fn = nn.BCEWithLogitsLoss()

# run through test data
all_ious = []
# all_accs = []
```

```
test_df=pd.DataFrame(columns=["ImgPosF","iou","DC","CE","OE"])
for i, batch in progress:
    x, y = batch['img'].float().to(device), batch['fpt'].float().to(device)
    idx = batch['idx']

    output = model(x).cpu()

    # obtain binary prediction map
    pred = np.zeros(output.shape)
    pred[output >= 0] = 1

    # derive Iou score
    cropped_iou = []
    for j in range(y.shape[0]):
        z = jaccard_score(y[j].flatten().cpu().detach().numpy(),
                          pred[j][0].flatten())
        if (np.sum(pred[j][0]) != 0 and
            np.sum(y[j].cpu().detach().numpy()) != 0):
            cropped_iou.append(z)

    all_ious = [*all_ious, *cropped_iou]

    # derive scalar binary labels on a per-image basis
    y_bin = np.array(np.sum(y.cpu().detach().numpy(),
                            axis=(1,2)) != 0).astype(int)
    prediction = np.array(np.sum(pred,
                                 axis=(1,2,3)) != 0).astype(int)
    # derive image-wise accuracy for this batch
#     all_accs.append(accuracy_score(y_bin, prediction))

    # derive binary segmentation map from prediction
```

```
output_binary = np.zeros(output.shape)
output_binary[output.cpu().detach().numpy() >= 0] = 1

if batch_size == 1:

    if prediction == 1 and y_bin == 1:
        res = 'true_pos'
    elif prediction == 0 and y_bin == 0:
        res = 'true_neg'
    elif prediction == 0 and y_bin == 1:
        res = 'false_neg'
    elif prediction == 1 and y_bin == 0:
        res = 'false_pos'
    #scores fix
    TP_eval.append((output_binary.squeeze()*y.cpu().detach().numpy().squeeze()).sum())
    FN_eval.append(((output_binary.squeeze()==0) & (y.cpu().detach().numpy().squeeze()==1)).sum())
    FP_eval.append(((output_binary.squeeze()==1) & (y.cpu().detach().numpy().squeeze()==0)).sum())
    dicec_eval_acc.append(dice2d(output_binary,y.cpu().detach().numpy()))
    test_df.loc[cont,"OE"]=FN_eval[cont]/(TP_eval[cont]+FN_eval[cont])
    test_df.loc[cont,"CE"]=FP_eval[cont]/(TP_eval[cont]+FP_eval[cont])
    test_df.loc[cont,"DC"]=dice2d(output_binary,y.cpu().detach().numpy())
    test_df.loc[cont,"ImgPosF"]=(batch['imgfile'][0].split("/")[2])
    OE=FN_eval[cont]/(TP_eval[cont]+FN_eval[cont])
    this_iou = jaccard_score(y[0].flatten().cpu().detach().numpy(),
                             pred[0][0].flatten())
    test_df.loc[i,"iou"]=this_iou


     # create plot
    f, (ax1, ax2, ax3,ax4) = plt.subplots(1, 4, figsize=(20,20))
    x=x.cpu()
```

```
    # false color plot Image prefire
    ax1.imshow(0.2+1.5*(np.dstack([x[0][12], x[0][11], x[0][10]])-np.min([x[0][12].numpy(),
                        x[0][11].numpy(), x[0][10].numpy()]))/(np.max([x[0][12].numpy(),
                        x[0][11].numpy(), x[0][10].numpy()])-np.min([x[0][12].numpy(),
                        x[0][11].numpy(), x[0][10].numpy()])), origin='upper')

    ax1.set_title("ImgPreF",fontsize=12)
    ax1.set_xticks([])
    ax1.set_yticks([])
    #Image Pos-Fire
    ax2.imshow(0.2+1.5*(np.dstack([x[0][4], x[0][3], x[0][2]])-np.min([x[0][4].numpy(),
                        x[0][3].numpy(), x[0][2].numpy()]))/(np.max([x[0][4].numpy(),
                        x[0][3].numpy(), x[0][2].numpy()])-np.min([x[0][4].numpy(),
                        x[0][3].numpy(), x[0][2].numpy()])), origin='upper')

    ax2.set_title("ImgPosF",fontsize=12)
    ax2.set_xticks([])
    ax2.set_yticks([])

    # segmentation ground-truth and prediction
    ax3.imshow(y[0], cmap='Greys_r', alpha=1)
    ax4.imshow(pred[0][0], cmap='Greys_r', alpha=1)
    ax3.set_title("Original Scar",fontsize=12)
    ax3.set_xticks([])
    ax3.set_yticks([])
    ax3.annotate("IoU={:.2f}".format(this_iou), xy=(5,15), fontsize=15)
```

```
ax4.set_title({'true_pos': 'Scar Prediction: True Positive \n  -IoU={:.2f},'
               '-OE={:.2f}, -CE={:.2f}, -DC={:.2F}'.format(this_iou, test_df.loc[cont,"OE"],test_df.loc[cont,"CE"],test_df.loc[cont,"DC"]),
       'true_neg': 'Scar Prediction: True Negative \n  -IoU={:.2f},'
    '-OE={:.2f}, -CE={:.2f}, -DC={:.2F}'.format(this_iou,test_df.loc[cont,"OE"],test_df.loc[cont,"CE"],test_df.loc[cont,"DC"]),
       'false_pos': 'Scar Prediction: False Positive   -IoU={:.2f},'
     '-OE={:.2f}, -CE={:.2f}, -DC={:.2F}'.format(this_iou, test_df.loc[cont,"OE"],test_df.loc[cont,"CE"],test_df.loc[cont,"DC"]),
        'false_neg': 'Scar Prediction: False Negative \n  -IoU={:.2f},'
    '-OE={:.2f}, -CE={:.2f}, -DC={:.2F}'.format(this_iou,test_df.loc[cont,"OE"], 0,test_df.loc[cont,"DC"])}[res],
              fontsize=12)
```

```
        cont+=1
        f.subplots_adjust(0.05, 0.02, 0.95, 0.9, 0.05, 0.05)

        # plt.savefig("U_Net/output/"+(os.path.split(batch['imgfile'][0])[1]).\
        #             replace('.tif', '.png').replace(':', '_'),
        #              dpi=200)

        plt.close()     #comment to display

print('iou:', len(all_ious), np.average(all_ious))
```

Test analysis

```
# test_df.to_csv("U_Net/runs/test_df_128_final1103.csv")
```

```
test_df["DC"].mean(), test_df["OE"].mean(),test_df["CE"].mean()
```

Figura 6.26 Código de evaluación U Net