ESCOLA TÉCNICA SUPERIOR
DE ENXEÑARÍA

Trabajo de Fin de Grado

Estudio de esquemas de clasificación de imágenes hiperespectrales basados en redes neuronales y técnicas de segmentación en superpíxeles

Autor:
Álvaro Goldar Dieste

Tutores:
Francisco Santiago Argüello Pedreira
Jorge Alberto Suárez Garea

Grao en Enxeñaría Informática

Escola Técnica Superior de Enxeñaría

Universidade de Santiago de Compostela

25 de junio de 2021

Trabajo de Fin de Grado presentado en la Escola Técnica Superior de Enxeñaría de la Universidade de Santiago de Compostela para la obtención del Grao en Enxeñaría Informática

ESCOLA TÉCNICA SUPERIOR
DE ENXEÑARÍA

D. Francisco Santiago Argüello Pedreira, Profesor del Departamento de Electrónica e Computación de la Universidade de Santiago de Compostela, y **D. Jorge Alberto Suárez Garea**, Investigador del Centro Singular en Tecnoloxías Intelixentes de la Universidade de Santiago de Compostela,

INFORMAN:

Que la presente memoria, titulada *Estudio de esquemas de clasificación de imágenes hiperespectrales basados en redes neuronales y técnicas de segmentación en superpíxeles*, presentada por **D. Álvaro Goldar Dieste** para superar los créditos correspondientes al Trabajo de Fin de Grado de la titulación de Grao en Enxeñaría Informática, se realizó bajo nuestra tutoría en el Departamento de Electrónica e Computación de la Universidade de Santiago de Compostela.

Y para que así conste a los efectos oportunos, expiden el presente informe en Santiago de Compostela, a 25 de junio de 2021:

Tutor,	Cotutor,	Alumno,
Firmado por ARGUELLO PEDREIRA FRANCISCO SANTIAGO - 32643463T el día 25/06/2021 con un certificado emitido por AC FNMT Usuarios	Firmado por SUAREZ GAREA JORGE ALBERTO - 33414810L Día 25/06/2021 Certificado emitido por AC FNMT Usuarios	Firmado por GOLDAR DIESTE ALVARO - 35485442F el día 25/06/2021 con un certificado emitido por AC FNMT Usuarios
Francisco Santiago Argüello Pedreira	Jorge Alberto Suárez Garea	Álvaro Goldar Dieste

Índice general

Siglas

AA Average Accuracy

Adam Adaptive Moment Estimation

CNN Convolutional Neural Network

CRS Countour Relaxed Superpixels

ETPS Extended Topology Preserving Segmentation

GAN Generative Adversarial Networks

HSI HyperSpectral Imaging

LSTM Long Short–Term Memory

OA Overall Accuracy

PCA Principal Component Analysis

ResNet Residual Network

RNN Recurrent Neural Network

SEEDS Superpixels Extracted via Energy–Driven Sampling

SLIC Simple Linear Iterative Clustering

SVM Support Vector Machine

Resumen

En el campo de la clasificación de imágenes multi e hiperespectrales, durante la última década se ha introducido y popularizado el uso de esquemas de clasificación basados en *deep learning*, gracias a las excelentes precisiones que consiguen alcanzar por su capacidad de modelar la naturaleza no lineal que conforma las imágenes hiperespectrales. Concretamente, las redes neuronales convolucionales han resultado ser el modelo de *deep learning* más popular en la actualidad para resolver estos problemas de clasificación multiclase.

No obstante, las mayores precisiones que los esquemas de *deep learning* consiguen alcanzar implican pagar costes computacionales mucho mayores que las técnicas de aprendizaje automático que se empleaban anteriormente. Esto limita en gran medida la aplicación de modelos de *deep learning* complejos para la clasificación hiperespectral, y prácticamente el uso de cualquier modelo de *deep learning*, sea más sencillo o más complejo, para escenarios que requieran procesamiento en tiempo real.

Para tratar de reducir el impacto computacional de estos nuevos esquemas, es posible emplear técnicas de segmentación en superpíxeles sobre las imágenes a tratar. Gracias a ellas, se puede reducir la cantidad de elementos a procesar a la hora de clasificar una imagen, sin incurrir en una pérdida de la información contenida en la misma, dado que los superpíxeles agrupan conjuntos de píxeles semejantes para reducir la redundancia de la información a procesar. Esto posibilita realizar una sola predicción de categoría por cada superpíxel, y asignársela a todos sus píxeles, en lugar de realizar una predicción por cada píxel de la imagen.

Mediante este trabajo, se analiza el impacto de incorporar una etapa de segmentación con Simple Linear Iterative Clustering antes de la clasificación hiperespectral con los esquemas de *deep learning* más relevantes en la literatura de este campo. Las observaciones experimentales recogidas permiten comprobar cómo es posible acelerar la clasificación de una imagen con estos esquemas de forma muy significativa –en el orden de cientos de veces en los experimentos llevados a cabo–, sin incurrir en ningún tipo de degradación de las precisiones de clasificación.

Como los esquemas seleccionados y analizados representan las ideas que se encuentran detrás de los esquemas basados en *deep learning* más populares en la actualidad para la clasificación hiperespectral, las anteriores implicaciones son extrapolables a un gran número de clasificadores propuestos en la literatura por otros autores. Esto posibilita acelerar muchos esquemas publicados actualmente, de cara a poder utilizarlos en aplicaciones de prácticamente tiempo real, y proponer esquemas más complejos basados en *deep learning* que ya pudieran resultar excesivamente costosos de tratar a nivel de píxel.

Palabras clave: imágenes hiperespectrales, clasificación, comparativa, redes neuronales, aprendizaje profundo, CNN, RNN, ResNet, segmentación en superpíxeles, SLIC

1 | Introducción

En el campo científico del sensado remoto, las cámaras de imagen hiperespectral –HyperSpectral Imaging (HSI)– capturan la interacción de la luz con los objetos de una escena a un gran nivel de detalle, dando lugar, habitualmente, a imágenes con decenas o cientos de bandas, correspondiendo una longitud de onda de luz diferente a cada una [1]. Esta gran cantidad de información hace que este tipo de imágenes permitan analizar con gran precisión las escenas observadas, en aplicaciones tales como la detección y clasificación de materiales y estructuras.

En los últimos años, un problema de gran interés a resolver con las HSI es la clasificación de imágenes: asignar a cada píxel de la imagen una categoría que representa lo que se puede encontrar en él [2]. De entre las muchas potenciales aplicaciones de la clasificación, una de ellas sería la identificación de los diferentes tipos de vegetación presentes en una escena natural. La clasificación de imágenes se diferencia de la clasificación de escenas en que, en esta última, se asigna una única categoría a la imagen en su conjunto, y no a cada píxel de forma individual.

Inicialmente, para abordar el problema de la clasificación HSI se recurrió a métodos de aprendizaje automático tradicionales, como la Máquina de Soporte de Vectores –Support Vector Machine (SVM)– [3]. Sin embargo, la reciente explosión del campo del aprendizaje profundo (*deep learning*) despertó el interés de la comunidad en emplear métodos de nueva generación basados en estos conceptos [4]. Tras el esfuerzo de muchos investigadores, hoy en día es comúnmente aceptado que los métodos de clasificación basados en *deep learning* permiten obtener resultados de mayor precisión en la clasificación HSI, en comparación con los métodos tradicionales [5]-[7].

Sin embargo, cabe destacar que el foco de la comunidad científica ha sido, principalmente, el incremento de la precisión de clasificación, mientras que el incremento del uso de recursos computacionales por parte de métodos basados en *deep learning* ha quedado relegado a un segundo plano [8]. Esto es un gran problema a afrontar en la clasificación HSI moderna, al dar lugar a un fuerte compromiso entre si priorizar la precisión de clasificación, o la rapidez de la misma, ya que esta última puede ser crucial en la práctica, sobre todo por aplicaciones de tiempo real.

Una las técnicas que se han empleado en los últimos años para reducir el uso de recursos computacionales en la clasificación HSI es la incorporación de técnicas de segmentación en superpíxeles [9]. Los superpíxeles permiten agregar conjuntos de píxeles con características similares, de cara a tratarlos como uno único en posteriores procesamientos [10]. De este modo, al realizar una segmentación antes de la clasificación HSI, simplemente será necesario obtener la categoría de un píxel dado, y a partir de aquí se podrá asignar esta misma categoría a los demás píxeles de su superpíxel. Esto podría suponer notables reducciones en el tiempo de clasificación total de una imagen.

A pesar de que las técnicas de superpíxeles se han aplicado a algunos esquemas de *deep learning*, tanto de clasificación de escenas como de imágenes, es un nicho que todavía no ha sido explorado con detenimiento. Existen algunas propuestas como [9], [11], pero habitúan ser escasas. Además, en estos casos, la segmentación se

estudia exclusivamente sobre el esquema de clasificación propuesto en la publicación. No existen comparativas del impacto de su aplicación a la multitud de esquemas de *deep learning* que se han popularizado para la clasificación HSI durante estos últimos años.

Por ello, mediante la realización de este Trabajo de Fin de Grado, se pretende incorporar, evaluar y comparar el impacto de una segmentación en superpíxeles en los esquemas de redes basados en *deep learning* más relevantes para la clasificación de imágenes hiperespectrales, en el campo del sensado remoto. En concreto:

1. Recopilar, de la bibliografía, las familias de esquemas de clasificación basados en *deep learning* más populares para la clasificación HSI.

2. Aplicar una fase de segmentación en superpíxeles a los tipos de esquemas seleccionados en el punto anterior, como etapa inicial de extracción de información espacial.

3. Realizar un estudio de los esquemas implementados, en cuanto a la precisión de la clasificación y consumo de recursos computacionales, para determinar el impacto de incorporar la etapa de segmentación.

4. Reducir todo lo posible el impacto de incorporar esta etapa de segmentación, en cuanto a consumo de recursos computacionales.

2 | Estado del conocimiento

A lo largo de la última década, las redes neuronales han experimentado un aumento vertiginoso de su aplicación en una multitud de problemas del mundo real, como la visión por computador o el procesamiento de lenguaje natural. La motivación principal para ello es que han resultado ser una alternativa viable para resolver problemas cada vez más complejos, ante los que técnicas de aprendizaje automático tradicional les cuesta hacer frente [12].

Las redes neuronales pertenecen a un subcampo del aprendizaje automático, comúnmente denominado *deep learning*, en el cual se trata de replicar el funcionamiento del cerebro humano para realizar una multitud de tareas complejas [12]. Generalmente, estas estructuras que son las redes neuronales se componen de una multitud de componentes computacionales denominados *neuronas*, que se agrupan en diferentes *capas*. Las capas se conectan unas tras otras para dar lugar a un modelo que transmite información entre sus diferentes componentes, al igual que lo hacen las conexiones sinápticas de un cerebro.

Por norma general, cada neurona se encuentra caracterizada por dos valores: su peso, y su sesgo. Una neurona aplica estos valores sinápticos a una determinada información de entrada, filtrándose su respuesta con una función de activación no lineal, y el resultado será el que se propague hacia posteriores capas. Siendo W_l los pesos de la capa l, b_l los sesgos, y X_{l-1} la información de entrada dada, la capa da lugar a una salida $X_l = f_l(X_{l-1}, W_l, b_l)$.

El objetivo general es ajustar los diferentes parámetros de la red –pesos y sesgos de sus componentes– de modo que la sucesión de operaciones no lineales de la red sea capaz de dar lugar a una salida deseada ante una información de entrada dada. Por ejemplo, que sea capaz de indicar la categoría a la que pertenece un píxel HSI dado. Para este último caso, la red debería aprender a extraer e identificar características de los diferentes elementos de la imagen HSI que se le presente, para ser capaz de discernir las categorías existentes.

De entre todas las arquitecturas de *deep learning* que se han puesto a prueba en la clasificación HSI durante los últimos años, la más popular de todas ellas es la Red Neuronal Convolucional –Convolutional Neural Network (CNN) [13]–. Este tipo de red neuronal incorpora el concepto de aplicar convoluciones a la información de entrada para extraer características de ella, lo que ha permitido alcanzar grandes avances en una variedad de campos en los últimos años, como en la visión por computador, o en el procesamiento auditivo [14].

Dentro del mundo de la clasificación HSI, es común categorizar las CNNs en función de si extraen exclusivamente características espectrales del píxel a clasificar [15] –denominado píxel–vector habitualmente–, o si también consideran información de píxeles vecinos –información espacial de un parche– para tratar de extraer una mayor información de contexto con la que alcanzar un mejor desempeño. Hoy en día, se ha comprobado de forma repetida cómo una CNN que extrae información espacial [16]-[18] será capaz de realizar una clasificación más precisa [5]-[7], y es por ello que este trabajo girará principalmente en torno a este tipo de redes.

Cuanto más profunda sea una arquitectura neuronal, podrá extraer caracterís-

ticas más complejas y discriminativas de la información de entrada para realizar mejor su tarea. Ahora bien, para tratar de mejorar todavía más el desempeño de las arquitecturas basadas en CNNs, no es suficiente con encadenar un gran número de capas unas tras otras, dado que esto suele dar diversos problemas que impiden a la red aprender a realizar su tarea [19], [20]. Es por ello que surgió el concepto de *aprendizaje residual* [19], que introduce "atajos" de capas de bajo nivel a capas de alto nivel para evitar el deterioro de la información a medida que se propaga a lo largo de las redes muy profundas, dando lugar a las arquitecturas habitualmente conocidas como Redes Residuales –Residual Network (ResNet)–.

Esto ha posibilitado conseguir algunos de los modelos de redes neuronales convolucionales más profundos y precisos en tareas de visión por computador [19] y, por supuesto, la tendencia se ha trasladado también al ámbito de la clasificación HSI [21], [22]. Es por ello que también se prestará atención a este paradigma a lo largo de este trabajo.

En último lugar, cabe destacar la presencia de las Redes Neuronales Recurrentes –Recurrent Neural Network (RNN)– en la clasificación HSI. Estas se tratan de redes neuronales que presentan bucles en su arquitectura, de modo que la salida de la red en un paso de tiempo i depende de cuál fuese la salida en el anterior paso de tiempo $i-1$ [23]. Esto otorga una cierta capacidad de "memoria" a la red, al poder analizar semejanzas entre una información dada e información que le haya sido presentada anteriormente.

En el ámbito de la clasificación HSI, dada la gran complejidad espectral de este tipo de imágenes, las RNNs pueden ser utilizadas para analizar la multitud de bandas de diferentes píxel–vectores, y determinar semejanzas entre ellos. Es por ello que se han propuesto esquemas para clasificación que tratan de incorporar módulos recurrentes, para intentar encontrar estos parecidos. Al final, su objetivo sería apoyar el trabajo de una CNN utilizando esta información de semejanza, para poder realizar mejores predicciones [24], [25].

A raíz de la gran variedad de herramientas de *deep learning* que se han aplicado a la clasificación HSI, muchos autores han tratado de ofrecer a la comunidad comparativas destacando las ventajas y desventajas de cada unas de ellas, para tratar de facilitar el conocimiento y entendimiento de las técnicas de mayor interés en el momento [5]-[7].

Por otra parte, la aplicación de técnicas de segmentación en superpíxeles a esquemas de clasificación HSI de *deep learning* es un nicho que todavía no ha sido explorado con demasiado detenimiento. Existen algunas propuestas en la literatura sobre la incorporación de esta etapa de segmentación antes de proceder a un determinado esquema de clasificación, como en [9], [11], pero son relativamente escasas.

Lo que sí que no existe –a nuestro mejor saber– es ningún tipo de publicación que analice el impacto de aplicar segmentaciones en superpíxeles sobre la variedad de arquitecturas de *deep learning* populares para la clasificación HSI, para poder:

- Determinar la medida en que cada tipo de arquitectura se ve afectada por la segmentación.
- Comparar los desempeños de las diferentes arquitecturas de *deep learning* al trabajar con superpíxeles, dado que hasta el momento solo existen comparativas a nivel de píxeles.

Estas serían las dos contribuciones, desde el punto de vista investigador, que resultan del cumplimiento de los objetivos propuestos para este Trabajo de Fin de Grado.

3 | Métodos de segmentación de imágenes en superpíxeles

A la hora de capturar una escena real y almacenarla como una imagen digital, siempre se genera una estructura matricial para ello. La imagen –matriz– tiene un alto H, y un ancho W, y cada píxel –elemento de la matriz– se encuentra caracterizado por un cierto valor en cada canal de color –o banda de la imagen–, siendo B el total de bandas. Por lo tanto, todo programa informático que trabaje con imágenes se diseñará de partida, con casi total seguridad, para trabajar con este tipo de estructura.

Esto es perfectamente natural en primera instancia: leer la información tal cual se almacena. Sin embargo, también cabe preguntarse por qué trabajar con las imágenes a nivel de píxel, en lugar de ir un paso más allá [10]. Al final, este concepto de *píxel* no es más que un artefacto que se genera al discretizar información del mundo real, y por lo tanto no es más que un modo de acceder a dicha información.

Además, el número de píxeles presentes en una imagen supera rápidamente el orden de millones en resoluciones consideradas estándar hoy en día; por ejemplo, los teléfonos móviles capturan fotos de varios millones de píxeles (megapíxeles). Consecuentemente, muchos algoritmos de visión por computador resultan rápidamente intratables en la práctica, al tener que trabajar con tantos elementos. Esto es algo que se ve todavía más pronunciado en el campo de la HSI, por la gran complejidad espectral que pueden llegar a tener las imágenes, con decenas o cientos de bandas.

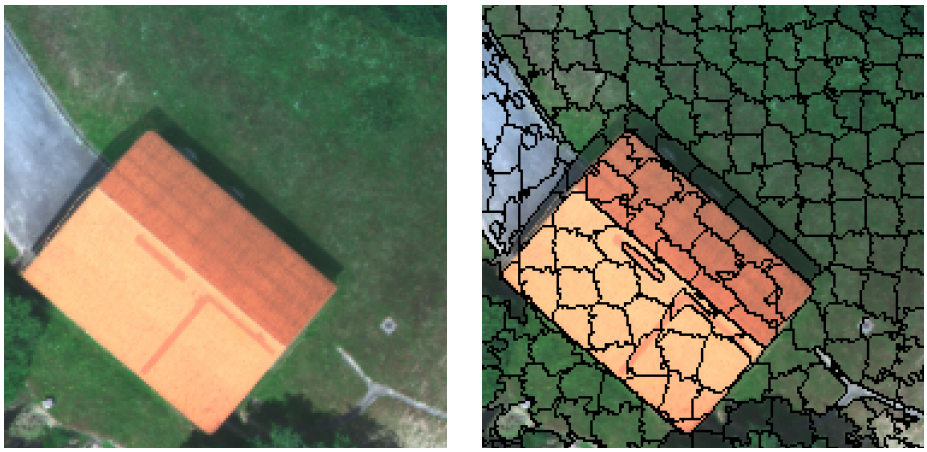
A raíz de estas consideraciones es como surge el concepto de *superpíxel*. Este pretende ser un artefacto más natural sobre el que procesar una imagen –sobre el que *acceder* a la información del mundo real que se pretende capturar–.

3.1. Qué es un superpíxel

Un superpíxel trata de agrupar píxeles contiguos con características similares, para dar lugar a una región homogénea. Esta no tiene por qué tener una forma concreta, sino que puede adaptarse, por ejemplo, a la forma de un tejado para agrupar parte de los píxeles pertenecientes a este, mientras que los píxeles del suelo se podrán agrupar en otros superpíxeles; es decir, los superpíxeles se adaptan en tamaño y forma a los objetos que aparecen en la escena a segmentar. Esta constituye una diferencia fundamental frente a métodos de segmentación más triviales, como simplemente particionar la imagen en una cuadrícula de regiones rectangulares.

Ahora bien, esto no implica que simplemente se pueda generar un superpíxel por cada objeto de la escena, ya que las diferencias de tamaño entre estos podrán ser notablemente grandes. Por ejemplo, el tejado de una casa podría ser mucho más pequeño que un gran jardín trasero. En realidad, los superpíxeles consideran las similitudes espectrales de los píxeles, pero también la proximidad espacial, de modo que píxeles semejantes a nivel espectral –como los píxeles del jardín– se podrán encontrar en superpíxeles diferentes en caso de estar suficientemente distanciados

a nivel espacial –si el jardín es muy grande–. Esta característica dará lugar a que los superpíxeles mantengan un tamaño relativamente semejante a lo largo de la imagen, además de dejar una cierta libertad a adaptarse a los objetos de la escena. Por claridad, la Figura 3.1 refleja este preciso fenómeno sobre una imagen de prueba que ha sido segmentada en base a superpíxeles.



(a) Región de la imagen de entrada.

(b) Región con la segmentación resultante superpuesta.

Figura 3.1: Resultado de una imagen segmentada en superpíxeles. Nótese cómo los superpíxeles presentan un tamaño semejante, pero también tratan de adaptarse a los objetos presentes en la escena.

Al capturar la redundancia de la imagen por agrupar píxeles similares, se consigue reducir en gran medida el número de elementos totales a tratar, permitiendo aminorar en consecuencia la carga computacional para posteriores programas que vayan a trabajar con la imagen segmentada. Es decir, un mapa de segmentación en superpíxeles se puede emplear como reemplazo de la estructura matricial original de la imagen, gracias a que presenta un menor número de elementos, pero que siguen conservando la información del mundo real que se pretende procesar.

3.2. SLIC: un popular algoritmo de segmentación en superpíxeles

A lo largo del tiempo, se han aplicado una multitud de técnicas basadas en superpíxeles en numerosos problemas de visión por computador [26]-[29]. De entre todas ellas, Simple Linear Iterative Clustering (SLIC) resulta ser una técnica de segmentación en superpíxeles notablemente popular, al ser capaz de conseguir segmentaciones de buena calidad, a la vez que conlleva costes computacionales reducidos [26], [30]. Su desempeño global se encontraría al nivel de otros algoritmos de segmentación en superpíxeles, como Countour Relaxed Superpixels (CRS) [31], Extended Topology Preserving Segmentation (ETPS) [32], o Superpixels Extracted via Energy–Driven Sampling (SEEDS) [33]. Además, permite personalizar la regularidad deseada para los superpíxeles –que sean más o menos rígidos– mediante un factor de compacidad que se presentará más adelante. Es por todo ello que se escogerá para los casos en que los experimentos a realizar durante este trabajo requieran segmentar imágenes HSI.

En esencia, este algoritmo toma inspiración a su vez del algoritmo de agrupa-

miento *k–medias* [34]. Concretamente, SLIC se basa en un conjunto de iteraciones de *k–medias* que trata de agrupar los píxeles de una imagen no solo en función de sus firmas espectrales, sino también introduciendo a mayores un factor espacial, de modo que se pondera la semejanza espectral de los píxeles junto con su proximidad espacial, para determinar a qué superpíxel pertenece cada píxel.

A continuación, se detallará la estructura global del algoritmo, tal y como se implementará para la posterior experimentación que se realice, dado que se trata de una variación sobre cómo fue propuesto originalmente [30]:

1. **Inicialización:** En primer lugar, el algoritmo distribuye tantos superpíxeles –*centroides*, según la notación original de *k–medias*– como sean necesarios a lo alto y ancho de la imagen, separándolos en una distancia de S píxeles unos de otros, hasta cubrirla al completo. Sería como particionar la imagen en una cuadrícula, por la cual cada centroide abarca un área de $S \times S$ píxeles. La etiqueta inicial de cada píxel se corresponde con el número de centroide bajo el cual ha resultado encontrarse; por ejemplo, comenzando a contar desde 0 con el primer centroide que se posicione en la imagen.

   Cada centroide se representa en un espacio $\mathbb{R}^{B+2}$, siendo B el número de bandas de la imagen. El valor inicial de cada centroide consiste en la media espectral de los píxeles que le han sido asignados, así como en la posición espacial (x, y) sobre la que ha sido colocado –que coincidiría con la media espacial de los píxeles que abarque–.

2. **Proceso iterativo:**

   a) A continuación, se recorre cada píxel de la imagen, buscando aquellos que se encuentren en los bordes de superpíxeles. Esto es, que un píxel vecino al píxel iterado pertenezca a un superpíxel diferente. Se emplea una conectividad de 4, de modo que se comprueban los píxeles norte, sur, oeste y este de cada píxel.

   b) Si se da la anterior condición, se calcula la distancia del píxel tanto a su centroide actual, como al centroide del vecino. Como se comentaba anteriormente, esta sería una ponderación de la semejanza espectral y espacial con cada centroide.

   Dado un centroide c y un píxel p, la distancia espacial entre ellos se determina en base a las coordenadas (x, y) correspondientes:

$$d_{espacial} = \sqrt{(x_c - x_p)^2 + (y_c - y_p)^2}. \tag{3.1}$$

   Por otra parte, la distancia espectral se determina con sus valores en cada banda de la imagen:

$$d_{espectral} = \sqrt{\sum_{i=1}^{B} (s_c^i - s_p^i)^2}. \tag{3.2}$$

   Y estos dos factores se ponderan del siguiente modo para dar lugar a la distancia final entre el centroide y el píxel:

$$d = d_{espectral} + \frac{m}{S} \cdot d_{espacial}, \tag{3.3}$$

   siendo m un factor de compacidad personalizable, que determina el peso de la semejanza espacial frente a la espectral. Cuanto menor sea su valor, más se podrán adaptar los superpíxeles a los bordes de elementos de la

escena capturada, aunque dejarán de presentar por lo tanto una forma tan regular [1].

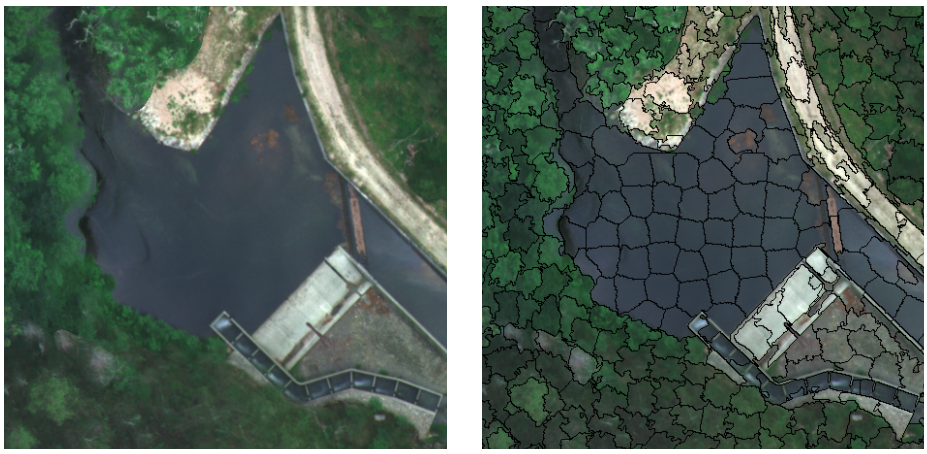
c) En caso de que la distancia del píxel al centroide vecino sea menor que hasta su centroide actual, se altera la etiqueta del píxel de modo que pertenezca ahora al primero. Consecuentemente, es necesario alterar el valor de ambos centroides –medias espectrales y espaciales dadas por los píxeles que les pertenecen–, para reflejar en uno el hecho de que el píxel ha dejado de pertenecerle, y para reflejar en el otro lo contrario.

d) Al final de la actual iteración, se computa la diferencia entre los valores de los centroides al inicio y al final de la misma. En caso de que la diferencia sea mayor que un umbral de parada dado, se repite el proceso iterativo hasta la convergencia del algoritmo, siempre y cuando no se haya alcanzado el número máximo de iteraciones permitidas.

3. **Arreglo de conectividad:** En último lugar, se aplica un post–procesado que revisa todos los superpíxeles resultantes, asegurándose de los píxeles pertenecientes a cada uno se encuentren conectados –que cada superpíxel sea una componente conectada–, dado que SLIC no asegura esta propiedad de por sí.

   En caso de que alguna serie de píxeles se haya quedado apartada, se les asigna la etiqueta de alguno de los superpíxeles que los rodeen.

Al final, el mapa de segmentación resultante para la imagen se encontraría representado por la etiqueta que haya sido asignada a cada píxel tras todo el procedimiento; es decir, la etiqueta de un pixel indicará el superpíxel al que pertenecerá. Por ejemplo, en la Figura 3.2 se ejemplifica cuál sería el resultado de generar una segmentación con SLIC sobre una región de una imagen de prueba.



(a) Región de la imagen de entrada.

(b) Región con la segmentación resultante superpuesta.

Figura 3.2: Resultado de generar una segmentación con SLIC sobre una región de una imagen de prueba. Se ha especificado un tamaño medio de 784 píxeles/superpíxel ($S \times S = 784$) –lo que equivale a superpíxeles de 28 píxeles de lado aproximadamente ($S = \sqrt{784} = 28$)–, y un factor de compacidad de 40 ($m = 40$).

[1] Es importante hacer un matiz sobre la elección de este valor. Dado que las HSI pueden contar con un número variable de bandas espectrales –desde decenas hasta cientos de ellas– no es posible escoger un valor m que se pretenda que funcione del mismo modo con cualquier HSI. Al contrario, será necesario ajustarlo manualmente para cada imagen, hasta obtener los resultados deseados.

Esta implementación optimizada de SLIC que se ha realizado difiere de la original [30], y tiene el propósito de incrementar la carga computacional útil que se realiza en cada iteración.

Tal y como fue propuesto inicialmente, SLIC permite que cualquier píxel de la imagen cambie de superpíxel en cada iteración, en lugar de limitar este fenómeno a los píxeles que se encuentren en los bordes de superpíxeles. Concretamente, para cada píxel de la imagen se determinarían todos los superpíxeles presentes en un rango de búsqueda de $2S \times 2S$ píxeles centrado en el píxel iterado en cada momento, y se comprobaría si alguno de ellos es más cercano que el superpíxel actual del píxel.

Particularmente, se ha decidido alterar esta política de actualización de la pertenencia de los píxeles en el algoritmo SLIC implementado por dos razones:

- Por una parte, los píxeles de los bordes son los más probables de pasar de un superpíxel a otro en todo momento.

  Al comenzar con una asignación arbitraria, como repartir al inicio los superpíxeles formando una cuadrícula, es cierto que muchos píxeles "querrán" cambiar de superpíxel para permitir que estos se vayan adaptando a las formas de los objetos de la imagen. Sin embargo, a medida que pasen múltiples iteraciones, como estas formas se comenzarán a estabilizar, cada vez más píxeles de los superpíxeles dejarán de "verse interesados" en cambiar de superpíxel, y las actualizaciones quedarán más bien relegadas a los píxeles de los bordes para acabar de ajustar lo mejor posible las formas de los superpíxeles a los objetos.

  Con esta nueva política de actualización, al permitir que menos píxeles cambien de superpíxel entre iteraciones, es esperable que la segmentación evolucione más despacio, y que por lo tanto requiera algunas iteraciones más para alcanzar el mismo resultado. A cambio de ello, se intenta minimizar el número de operaciones de poca utilidad esperable que se hacen en cada iteración, al limitar en todo momento los cómputos de distancias píxel–centroide a tan solo aquellos píxeles que mantengan unas probabilidades significativas de cambiar de pertenencia.

  Este último factor permitirá reducir de forma muy significativa la carga computacional de cada iteración, al compensar la realización de algunas iteraciones más con el hecho de que estas sean mucho más veloces.

- Por otra parte, es muy importante que se haya eliminado la necesidad de realizar una multitud de búsquedas para determinar qué píxeles deben de cambiar los superpíxeles a los que están asignados.

  Dado el nuevo criterio para que un píxel pueda cambiar de superpíxel, ya no tiene sentido emplear el rango de búsqueda para determinar superpíxeles candidatos. Esto era coherente al permitir que cualquier píxel de la imagen lo hiciese. En cambio, como ahora se tendrá la certeza de que un candidato a cambio se encontrará en un borde, es prácticamente inmediato obtener el superpíxel respecto al cual calcular la distancia alternativa, en lugar de tener que recorrer un área de $2S \times 2S$ píxeles a propósito.

  Esto permite aliviar todavía más la carga computacional de cada iteración.

Con todo ello, se pretende conseguir que las iteraciones más livianas de esta versión optimizada de SLIC permitan compensar el sobrecoste de realizar algunas iteraciones más, y que lo hagan además de forma holgada para reducir incluso el tiempo de segmentación total.

3.3. Acelerando la clasificación HSI con los superpíxeles

Finalmente, es necesario concretar cómo segmentar una imagen permite mejorar el rendimiento computacional de una tarea de clasificación HSI.

La reducción del número de primitivas a tratar –procesar superpíxeles en lugar de píxeles– se puede traducir en una reducción del número de muestras de la imagen que un clasificador debe tratar. En un problema de clasificación supervisada, una muestra se compone de una cierta información que la caracteriza, así como la categoría a la que pertenece.

La forma habitual de proceder al trabajar con los clasificadores HSI a nivel de píxel es:

- Para generar la información que caracteriza la muestra:
  - Si el clasificador procesa un simple píxel–vector, se toma directamente el píxel actual.
  - Por el contrario, si trabaja con *parches* –múltiples píxeles–, se toma una subregión de la imagen de $N \times N$ píxeles centrada sobre el píxel actual, siendo N un valor de vecindad dado.
- Por otra parte, la categoría de la muestra es simplemente la que haya sido asignada al píxel actual en la información de referencia.

No obstante, trabajar en base a superpíxeles abre un nuevo abanico de posibilidades, al poder emplear múltiples píxeles para conformar una sola muestra. Por ello, es necesario concretar cómo generar muestras sobre superpíxeles a partir de la información de los píxeles que contienen. En este caso, se partirá de la propuesta de generación de muestras de [29], tal que:

- Para generar la información que caracteriza la muestra, se empleará parte de los píxeles del superpíxel.

  Concretamente, se determinará el cuadrilátero mínimo que encierra el superpíxel, y el punto central del mismo indicará el *píxel central del segmento*. Para tener en cuenta los pequeños matices que pueda haber entre los diferentes miembros del superpíxel, siempre se considerarán los píxeles vecinos que se encuentren en un determinado rango de búsqueda de tamaño $N \times N$ píxeles centrado sobre el píxel central. Todo este procedimiento se representa en la Figura 3.3 por claridad.

  En función de qué tipo de entrada requiera el clasificador:

  - Si trabaja con píxel–vectores, se computará la media espectral de los píxeles contenidos en el rango de búsqueda para conformar un único píxel–vector que los represente.
  - Si el clasificador trabaja con parches, se tomará directamente como entrada la subregión de la imagen abarcada por el rango de búsqueda.
- Por otra parte, para determinar la categoría a la que pertenecerá el superpíxel, se tendrán en cuenta las categorías a las que pertenecen todos los píxeles del mismo.

  Concretamente, se escogerá como categoría del superpíxel aquella que más se repita entre todos los píxeles, siguiendo un procedimiento de voto mayoritario.

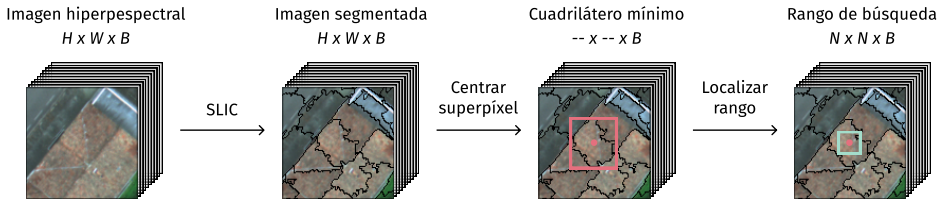


Figura 3.3: Proceso de posicionamiento del rango de búsqueda sobre un superpíxel dado. Nótese que la marca roja central representa la posición del píxel central encontrado.

Por supuesto, todo esto es generalizable a un problema de aprendizaje no supervisado. Simplemente es necesario obviar la generación de la categoría del superpíxel, al ser un dato inexistente en las muestras de este paradigma.

Una vez concretado cómo generar muestras a partir de los superpíxeles, es posible analizar dónde se podrá obtener una aceleración en el uso de un clasificador HSI. Todo clasificador de aprendizaje automático pasa por dos etapas en su vida, que son susceptibles de beneficiarse de la reducción de muestras a tratar: una fase de entrenamiento –o aprendizaje–, y una fase de prueba –o evaluación–.

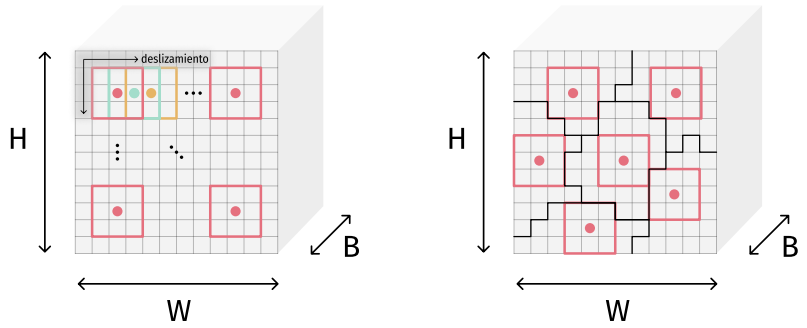
En primera instancia, no tiene mucho sentido tomar partido de la reducción de muestras en la etapa de entrenamiento. Al final, es en esta en donde el clasificador extrae información de los datos suministrados para aprender a realizar su tarea. Por ello, si como consecuencia de segmentar una imagen se suministran menos muestras al clasificador durante su entrenamiento, en comparación a procesar la imagen a nivel de píxeles, es esperable que esta etapa se vea acelerada, pero a cambio de sacrificar el desempeño final del clasificador por haber reducido la riqueza y variedad de la información a la que se ve expuesto.

Por lo tanto, se pretenderá en todo caso que un clasificador reciba el mismo número de muestras durante su etapa de entrenamiento, se haya segmentado o no la imagen, para que las condiciones de partida sean justas.

Ahora bien, la etapa de prueba es una candidata perfecta para ser acelerada. En esta, se evalúa el desempeño del clasificador resultante determinando en qué medida es capaz de predecir correctamente las categorías de muestras que no haya visto durante su entrenamiento. Aunque esta tarea se puede llevar a cabo perfectamente a nivel de píxel, también sería coherente realizarla a nivel de superpíxel. Al final, estos son regiones homogéneas –contienen píxeles similares–, por lo cual es coherente preguntarse hasta qué punto merece el esfuerzo predecir la categoría de cada píxel por separado, si se presupone que sus compañeros son parecidos y, consecuentemente, deberían pertenecer a la misma categoría. Es decir, podría predecirse una única categoría para el superpíxel, y asignársela a todos los píxeles que contenga.

Por claridad, en la Figura 3.4 se representan ambos enfoques sobre el procedimiento de clasificación de una imagen. Suponiendo un clasificador que trabaja con parches, en el caso de procesar la imagen a nivel de píxel –Figura 3.4(a)– se puede observar cómo se genera un parche a partir de cada píxel a clasificar, realizándose al final tantas predicciones como píxeles contenga la imagen. En cambio, en el caso de trabajar a nivel de superpíxel –Figura 3.4(b)– simplemente es necesario realizar una predicción por superpíxel, asignando la categoría predicha en cada uno a todos sus píxeles. Consecuentemente, el número de predicciones a realizar para clasificar toda la imagen es considerablemente menor.

Por todo lo explicado anteriormente, para la fase experimental de este trabajo se propondrá determinar en qué medida la precisión de un clasificador se podrá ver



(a) Clasificación de una imagen trabajando a nivel de píxel.

(b) Clasificación de una imagen trabajando a nivel de superpíxel.

Figura 3.4: Representación del procedimiento de clasificación de una imagen desde dos enfoques diferentes; trabajando a nivel de píxel, y trabajando a nivel de superpíxel. Nótese cómo el número total de predicciones a realizar es mucho menor al trabajar con la imagen segmentada en superpíxeles.

alterada por realizar este tipo de predicción conjunta durante la fase de prueba, en lugar de una predicción a nivel de píxel. Siendo $S \times S = S^2$ el tamaño medio de píxeles/superpíxel, además será esperable observar una aceleración de esta etapa de prueba en el orden de S^2, ya que se pasará a realizar, aproximadamente, una sola predicción a nivel de superpíxel por cada S^2 predicciones a nivel de píxel.

4 | Métodos de clasificación de imágenes hiperespectrales

Dentro de los múltiples paradigmas del aprendizaje automático, el aprendizaje supervisado resulta ser el más popular en el ámbito de la clasificación HSI, al ser capaz de proporcionar, en general, un mejor desempeño que los modelos basados en aprendizaje no supervisado [5].

El aprendizaje supervisado consiste en suministrar muestras categorizadas a un clasificador, de modo que aprenda a extraer características discriminativas de la información que las caracteriza, y sea capaz de asociarlas a sus categorías –o clases– determinadas de antemano. Es decir, dadas unas muestras de entrenamiento $\mathcal{D}_{entrenamiento} = \{x_i, y_i\}$, siendo x_i la información que caracteriza la muestra i, y siendo y_i la categoría que le ha sido asignada, el clasificador trata de adaptarse para poder modelar lo mejor posible la relación $x_i \Leftrightarrow y_i$ de todas las muestras.

Tras esta etapa de aprendizaje, el clasificador resultante habitúa ser puesto a prueba ante nuevas muestras que no ha visto durante el entrenamiento, con el objetivo de evaluar su capacidad de generalizar el conocimiento adquirido. Para ello, debe tratar de predecir la categoría correspondiente a estos nuevos datos, y su desempeño será medido en función de los aciertos y fallos cometidos.

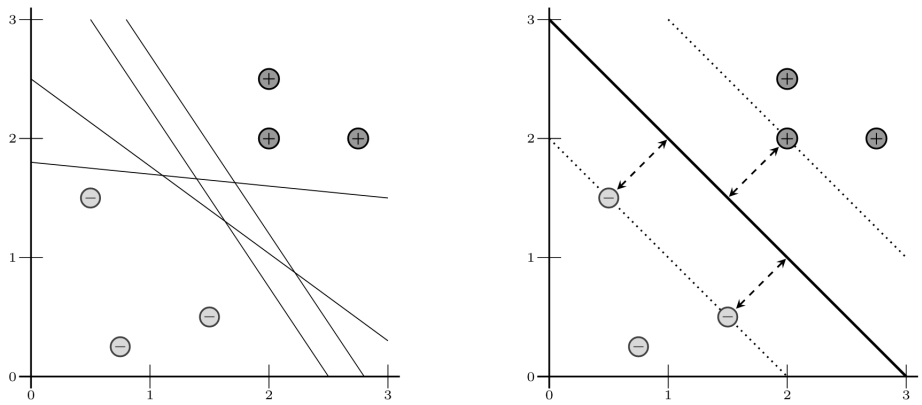
A continuación, este cuarto capítulo del trabajo se encargará de introducir poco a poco al lector las ideas de *deep learning* más presentes en el campo de la clasificación HSI en la actualidad, y se presentarán sobre ellas diferentes esquemas de clasificación supervisados y basados en *deep learning*. Estos esquemas serán los que se evalúen durante la fase experimental del trabajo, para comprobar las implicaciones de incorporar la etapa de segmentación en superpíxeles.

4.1. Antes del *deep learning*: la fiabilidad de SVM

Antes de la popularización de técnicas de *deep learning* en la clasificación HSI, uno de los métodos más comunes a lo largo de los años para esta tarea ha sido la Máquina de Soporte de Vectores –SVM–. Esta nació originalmente como un clasificador para problemas binarios, cuya intención es tratar de encontrar el hiperplano separador de máximo margen entre las muestras de dos categorías[1] [36], tal y como se ejemplifica en la Figura 4.1.

Con el tiempo, las capacidades de aplicación de SVM también se extendieron a problemas multiclase, como la clasificación HSI [3]. En este caso, SVM toma cada píxel–vector a clasificar como una muestra, y trata de discernir las C diferentes categorías de la imagen empleando la riqueza de la información espectral de las HSI. Para problemas multiclase se pueden seguir diferentes estrategias [3], como:

[1] Las categorías no tienen por qué ser linealmente separables. SVM evolucionó con el paso del tiempo para ser capaz de hacer frente a situaciones en las que no se cumple esta condición, intentando minimizar en todo caso el error cometido por muestras clasificadas erróneamente [35].



(a) Conjunto de datos linealmente separables.

(b) Hiperplano separador de máximo margen.

Figura 4.1: Búsqueda del hiperplano separador de máximo margen que es capaz de diferenciar dos conjuntos de datos [37]. Nótese que las muestras que se cruzan con las líneas punteadas constituyen los vectores de soporte seleccionados.

- **Estrategia *Uno–Contra–Todos*:** se emplean C clasificadores binarios, con la intención de que el clasificador i separe la categoría c_i de las demás.
- **Estrategia *Uno–Contra–Uno*:** se emplean $C \cdot (C-1)/2$ clasificadores binarios –tantas parejas de categorías como es posible formar–, y se lleva a cabo una predicción con cada clasificador, asignando a la muestra la categoría que más se repita.

Esta última estrategia habitúa ser la más popular, al presentar tiempos de entrenamiento más reducidos [38].

Además de ello, cabe destacar que una técnica habitual en SVM es emplear kernels (núcleos) para proyectar los datos a tratar a un espacio con mayor dimensionalidad, de modo que sean más fácilmente separables [3]. Algunos ejemplos de ello lo son el kernel polinomial, o el kernel de función de base radial.

A lo largo de los años, la fiabilidad de este algoritmo le ha permitido ganarse una reputación de ser una buena opción a la hora de afrontar un problema de clasificación HSI, siendo capaz de reportar precisiones relativamente comparables incluso a algunos modelos más modernos de *deep learning* [5]. Es por ello que se seleccionará como un representante del aprendizaje automático tradicional, frente al cual comparar las técnicas más modernas que se introduzcan a continuación.

4.2. La llegada del *deep learning*: la popular CNN

Dentro del campo de la visión por computador, una de las herramientas más populares del *deep learning* en los últimos años es la Red Neuronal Convolucional –CNN– [13]. Para facilitar su entendimiento, es conveniente introducir en primer lugar el concepto de aplicar una convolución a una imagen.

En esencia, esta operación consiste en utilizar un filtro sobre una imagen de entrada, con el objetivo de:

- Alterar la visualización de la imagen con diferentes efectos, como un difuminado. Esta es una funcionalidad muy habitual en el software de edición fotográfica.

- Identificar características de los elementos presentes en una imagen; por ejemplo, podría aplicarse un filtro de delineado para identificar bordes de objetos. Es una utilidad usada principalmente en la visión por computador y el aprendizaje automático asociado.

A modo de ejemplificación, en la Figura 4.2 puede observarse cuál sería el resultado visual de aplicar una convolución a una imagen. Para generar un píxel de la imagen de salida, sería necesario centrar el filtro sobre un píxel de la imagen de entrada, y calcular la contribución de cada elemento del filtro con su píxel de entrada correspondiente –se involucran píxeles vecinos del píxel central, por posición espacial–. Como será necesario hacer esta operación para cada píxel de la imagen de entrada, se estará desplazando el filtro a lo alto y ancho de la misma: esto es lo que se denominaría *convolución 2D*.

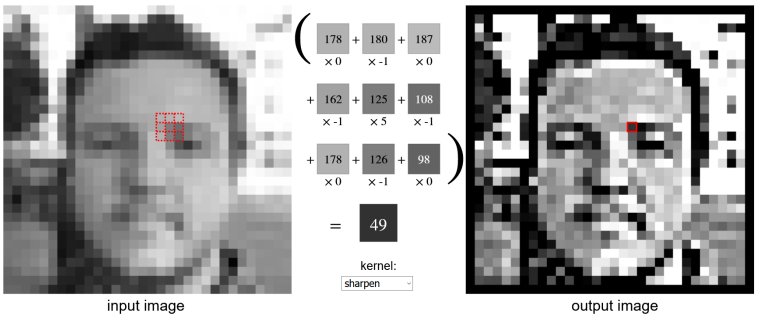


Figura 4.2: Aplicación de una matriz de convolución correspondiente a un efecto de nitidez a una imagen en escala de grises [39].

Retomando la CNN, el bloque básico de construcción de este tipo de red es la capa convolucional. En esta, se toma una determinada información de entrada x, como bien podría ser una imagen, o parte de ella, y se le aplican diferentes filtros para dar lugar a la salida y. De modo más formal, sean N los diferentes filtros de la capa convolucional, cada uno con sus correspondientes pesos W y sesgos b, la operación definida por la capa sería

$$y = \{W_i * x + b_i\}_{i=1,2,\ldots,N}, \tag{4.1}$$

siendo $*$ la operación de convolución. En un caso como el de la Figura 4.2, al aplicar N diferentes filtros sobre una imagen en escala de grises –una única banda– se daría lugar a N diferentes imágenes de salida o, lo que es lo mismo, una información de salida con N bandas. En el caso de que x tuviese más de una banda, la operación de convolución con el filtro i debería recorrer y procesar conjuntamente todas las diferentes bandas de x antes de dar lugar a la banda de salida i.

Emplear una capa convolucional para procesar información visual, en lugar de una capa densa tradicional de las redes neuronales[2], aporta una serie de ventajas para nada despreciables:

- Por simplicidad, supóngase el caso más básico de querer generar una única banda de salida. Al emplear una capa densa sería necesario definir una neurona por cada píxel de la información de entrada, cada una con sus pesos y sesgos correspondientes. En comparación, una capa convolucional define un

[2]Una capa densa se compone de una cantidad dada de neuronas que no se conectan entre sí. En cambio, las salidas de todas las neuronas de la capa i están conectadas con las entradas de todas las neuronas de la capa $i + 1$.

filtro que se comparte a lo largo de toda la información, y que conllevará un número mucho menor de parámetros –pesos y sesgos– a ajustar durante el entrenamiento.

Esta reutilización de parámetros permite acelerar el aprendizaje de la red, así como ayuda a evitar la posiblidad de que se den fenómenos no deseados como el sobreajuste –u *overfitting*–.

- Además, para procesar la información de entrada con la capa densa sería necesario aplanarla, concatenando los valores de los diferentes píxel–vectores unos detrás de otros, mientras que una convolución permite conservar la estructura espacial original. Consecuentemente, esto permite a una capa convolucional el analizar y extraer información espacial de la entrada, además de la información espectral pertinente.

  Cabe tener presente que esta es una limitación no solo presente en capas densas, sino también en métodos tradicionales como SVM, para el cual una muestra en la clasificación HSI se caracteriza por un único píxel–vector –una secuencia de valores sin estructura espacial–.

Otro bloque de gran importancia en una CNN es la capa de reducción de muestreo –también llamada *pooling*–. Esta se encarga de reducir la dimensionalidad espacial de una determinada información de entrada. Para ello, se particiona la información en diferentes regiones, y se aplica una operación a los elementos de cada región para dar lugar a una salida con tantos elementos como regiones.

Por ejemplo, en la Figura 4.3 se representa la aplicación de una operación de *max–pooling*. Como bien su nombre indica, el elemento resultante de cada región se trata del máximo valor que se pueda encontrar en ella. Por supuesto, existen otros muchos tipos de operaciones de *pooling*, como el *average–pooling*, que calcula la media de los elementos de cada región.

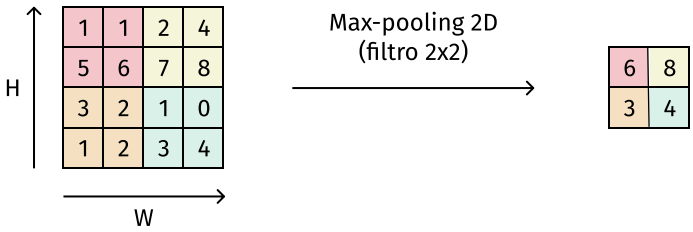


Figura 4.3: Representación de la aplicación de una operación de *max–pooling* a una información de entrada con una única banda [40].

La utilidad de este tipo de capa reside en que la reducción de dimensionalidad permite:

- Generalizar la información dada, a modo de filtrar posible ruido y conservar las características de la información que resulten más representativas.

  Con la reducción de dimensionalidad se da menos importancia a pequeñas peculiaridades que haya en una posición concreta de la información, y se preservan principalmente características dominantes que se pueden observar a lo largo de la entrada, con relativa independencia de la posición espacial.

- Y, por supuesto, reducir la carga computacional de las siguientes capas, al reducir el número de elementos de la información.

La combinación de capas convolucionales y de capas de *pooling* constituye el plano más básico para construir una CNN. Como norma general, es común formar parejas con una capa de cada tipo, y encadenar múltiples parejas que van procesando la información de entrada desde su forma original, extrayendo características cada vez más abstractas y discriminativas de la misma gracias a la combinación de convoluciones y operaciones de *pooling*. Esta se podría considerar como una primera subred presente en la CNN.

Tras ello, la CNN habitúa incorporar una segunda subred compuesta exclusivamente por capas densas. La tarea de estas sería tomar todas las características extraídas por la primera subred, y procesarlas para clasificar la información de entrada dada, asignándole una categoría de entre todas las posibles. Para este último paso, se aplica la función softmax sobre un vector con tantos elementos como posibles categorías, y asigna una probabilidad de pertenencia a cada categoría de modo que la suma de todas las probabilidades sea 1:

$$P(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{C} e^{x_j}}, \tag{4.2}$$

$$\sum_{i=1}^{C} P(x_i) = 1, \tag{4.3}$$

siendo C el total de categorías. La categoría asignada es aquella que resulte tener la mayor probabilidad –mayor valor en su correspondiente posición del vector–.

En la práctica, toda la CNN se trata como un elemento monolítico durante el proceso de entrenamiento. Los filtros de las diferentes capas convolucionales se inicializan a valores aleatorios, al igual que los pesos y sesgos de las capas densas. Mediante retropropagación, se van ajustando todos los pesos y sesgos de ambas subredes, de cara a que la CNN pueda aprender a realizar su tarea de clasificación. El ajuste de los filtros de las capas convolucionales puede interpretarse como un proceso mediante el cual averiguar qué filtros conviene emplear para extraer información lo más discriminativa posible de la imagen de entrada, de modo que permita realizar la clasificación con mayor certeza, en lugar tener que predefinir filtros a mano que se crea que vayan a funcionar de forma óptima.

Tomando como base todos los conceptos presentados hasta el momento, se comenzará ahora a sugerir una serie de esquemas de clasificación sobre los cuales experimentar con la incorporación de la etapa de segmentación en superpíxeles.

Esquema A: CNN 2D

El primer esquema consistirá en una sencilla CNN con filtros 2D, basada en la propuesta en [29]. La Figura 4.4 aporta una representación visual de la estructura de esta CNN.

Como se puede apreciar en la figura, la entrada de la red es un parche –una muestra– de la HSI a clasificar. En este caso concreto, se puede observar además cómo el parche se extrae de un superpíxel identificado en la imagen, aunque el esquema también sería perfectamente aplicable al caso de trabajar con la imagen a nivel de píxel, recordando las explicaciones del Apartado 3.3 en la página 10. Entonces, el objetivo será determinar la categoría perteneciente a la muestra. Esta categoría corresponderá a la del píxel central en caso de procesar la imagen a nivel de píxel, o sería la que se asigna al superpíxel en caso de haber segmentado la imagen.

La primera subred contiene dos pares de capas, constando cada par de una capa convolucional seguida de una capa de *max–pooling*, para realizar la extracción

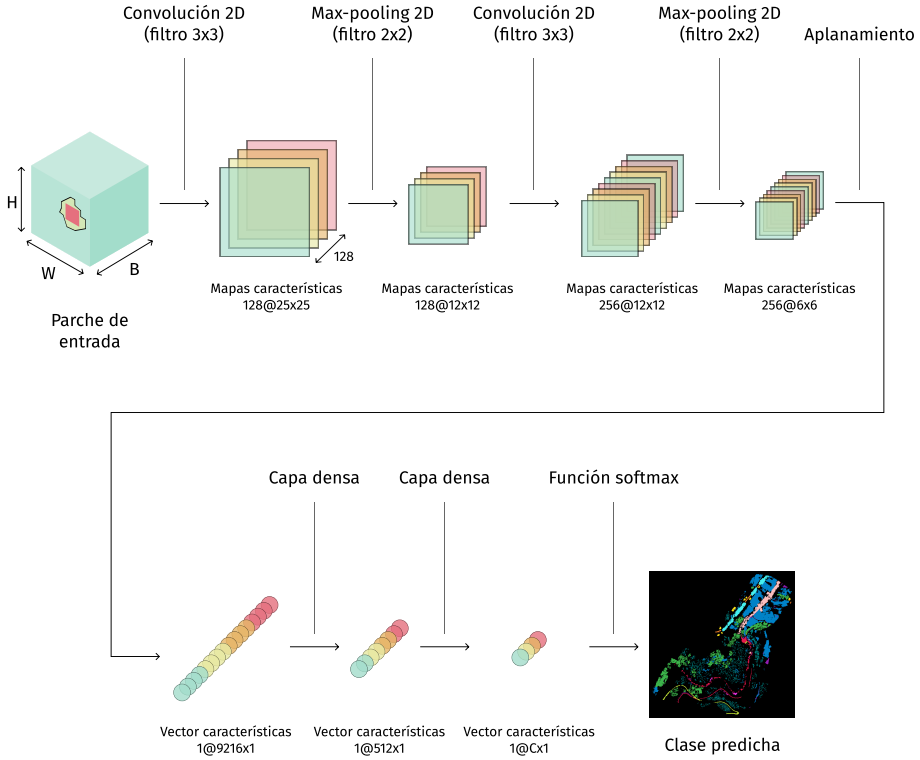


Figura 4.4: Esquema de la arquitectura de la CNN 2D descrita. Las dimensiones son las correspondientes a una entrada con dimensiones $H \times W \times B = 25 \times 25 \times B$. Nótese que cada banda de la salida de una capa convolucional se denomina *mapa de características*. El conjunto de mapas de características resultantes de una capa es la entrada de la siguiente capa.

de características cada vez más abstractas de la información de entrada. Todas estas características son procesadas por la segunda subred, que consta de dos capas densas, para realizar la clasificación final.

El Cuadro 4.1 detalla las características de cada capa de la red. Es importante destacar el uso de la función de activación ELU tras las capas convolucionales y la primera densa, dado que ofrece determinadas ventajas sobre otras funciones de la popular familia ReLU [41] –muy empleada en la clasificación HSI basada en *deep learning*–, así como un entrenamiento más veloz, y mejores precisiones [42].

Cuadro 4.1: Detalles de la arquitectura de la CNN 2D descrita. Las dimensiones de salida son las correspondientes a una entrada con dimensiones $H \times W \times B = 25 \times 25 \times B$. Antes de cada convolución se aplica un relleno en las dimensiones H y W de la información, para preservar el tamaño de las mismas.

#	Tipo	Dim. salida	Activación	Tam. filtro	Desplazamiento
1	Conv. 2D	$25 \times 25 \times 128$	ELU	3×3	1×1
2	Max-pool. 2D	$12 \times 12 \times 128$	–	2×2	2×2
3	Conv. 2D	$12 \times 12 \times 256$	ELU	3×3	1×1
4	Max-pool. 2D	$6 \times 6 \times 256$	–	2×2	2×2
5	Aplanamiento	9216×1	–	–	–
6	Densa	512×1	ELU	–	–
7	Densa	$C \times 1$	Softmax	–	–

Esquema B: CNN 3D

En el subapartado anterior se presentó una red que aplica convoluciones 2D sobre la información de entrada. En esta operación:

- Para generar un elemento de la salida, se sitúa el filtro sobre un elemento de la entrada, y se aplica el filtro sobre todos los elementos abarcados. En caso de que la entrada contenga múltiples bandas, el filtro recorre todas ellas para generar una única banda en la salida.
- Al considerar todas las bandas de entrada al mismo tiempo, el grado de libertad de la convolución es de 2, al poder desplazarse a lo alto y ancho de la entrada.

Ahora bien, este concepto es perfectamente generalizable a una dimensión más. En lugar de que cada filtro recorra todas las bandas a la vez, se puede limitar su rango de actuación en un determinado momento a un subconjunto de ellas. Consecuentemente, la convolución puede desplazarse ahora en la dimensión de profundidad, además de lo alto y ancho, por lo que la salida de la operación también contendrá una dimensión más. Esto sería una *convolución 3D*.

Para ejemplificar visualmente este nuevo concepto, se va a presentar un segundo esquema de clasificación que lo emplee. Este consistirá esencialmente en la CNN 2D, solo que reemplazando las capas convolucionales 2D por versiones 3D. La arquitectura correspondiente se refleja en la Figura 4.5.

En la línea de lo que se comentaba, la salida de cada capa convolucional tiene ahora una dimensión más. En la CNN 2D se generaban múltiples mapas de características con cada capa convolucional, uno por cada filtro 2D de la capa. Ahora, cada filtro 3D es capaz de dar lugar a múltiples mapas bidimensionales de características

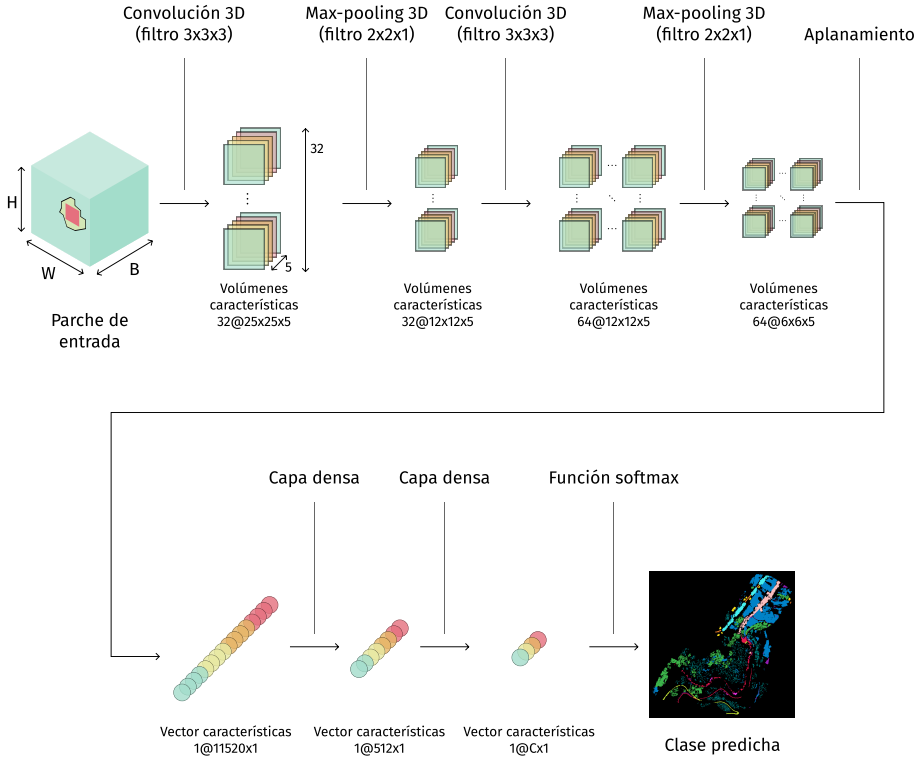


Figura 4.5: Esquema de la arquitectura de la CNN 3D descrita. Las dimensiones son las correspondientes a una entrada con dimensiones $H \times W \times B = 25 \times 25 \times 5$. Nótese que cada elemento de la salida de una capa convolucional se denomina *volumen de características*. El conjunto de volúmenes de características resultantes de una capa es la entrada de la siguiente capa.

en lugar de a uno único, al considerar subconjuntos de las bandas de entrada. Los mapas correspondientes a un único filtro 3D se agrupan bajo el nombre de *volumen de características*, generándose con cada capa tantos volúmenes como filtros le hayan sido designados.

El interés de emplear filtros 3D, en lugar de filtros 2D, es que permiten prestar más atención a la información espectral de la información a procesar. De nuevo, en el caso de filtros 2D, todas las bandas de entrada son consideradas a la vez para generar una única banda de salida. En cambio, en los filtros 3D es posible dar más importancia a peculiaridades de determinadas bandas de entrada, al ser procesados en subconjuntos. Dada la gran complejidad espectral de las HSI, esto ha llevado a plantear el uso de este tipo de convoluciones para tratar de extraer información más discriminativa, como en [16].

También se dispone el Cuadro 4.2 para detallar las características de cada capa de la CNN 3D. De nuevo, se emplea la función de activación ELU. Cabe destacar que el número de filtros se ha reducido para mantener relativamente equiparables el número de características obtenidas para la segunda subred, entre la CNN 2D y esta CNN 3D. De este modo, en caso de que la CNN 3D presente un rendimiento claramente mejor que la CNN 2D, se podrá determinar que ello es gracias a prestar más atención a las características espectrales de la información, en lugar de por incrementar simplemente el total de características extraídas.

Cuadro 4.2: Detalles de la arquitectura de la CNN 3D descrita. Las dimensiones de salida son las correspondientes a una entrada con dimensiones $H \times W \times B = 25 \times 25 \times 5$. Antes de cada convolución se aplica un relleno en las dimensiones H, W y B de la información, para preservar el tamaño de las mismas.

#	Tipo	Dim. salida	Activación	Tam. filtro	Desplazamiento
1	Conv. 3D	$25 \times 25 \times 5 \times 32$	ELU	$3 \times 3 \times 3$	$1 \times 1 \times 1$
2	Max-pool. 3D	$12 \times 12 \times 5 \times 32$	–	$2 \times 2 \times 1$	$2 \times 2 \times 1$
3	Conv. 3D	$12 \times 12 \times 5 \times 64$	ELU	$3 \times 3 \times 3$	$1 \times 1 \times 1$
4	Max-pool. 3D	$6 \times 6 \times 5 \times 64$	–	$2 \times 2 \times 1$	$2 \times 2 \times 1$
5	Aplanamiento	11520×1	–	–	–
6	Densa	512×1	ELU	–	–
7	Densa	$C \times 1$	Softmax	–	–

Esquema C: CNN 2D + RNN

Dentro de la clasificación HSI, otro de los paradigmas que se ha popularizado con el uso generalizado del *deep learning* es la aplicación de una Red Neuronal Recurrente –RNN– para la extracción de características espectrales. En este tipo de red se caracteriza por presentar bucles en su arquitectura, de modo que la salida de la red en un *paso de tiempo* i depende de cuál fuese la salida en el anterior paso de tiempo $i-1$ [23]. Esto otorga una cierta capacidad de "memoria" a la red, al poder actuar en base a información que le haya sido presentada con anterioridad [43].

Para incorporar una RNN a un esquema de clasificación HSI, es necesario tener en cuenta que la información de entrada esperada debe ser una simple secuencia de valores. Consecuentemente, ya no es posible suministrar directamente un parche de la imagen a tratar, como se hacía con una CNN, dado que esta información presenta tres dimensiones: alto, ancho y profundidad. Por ello, es habitual limitarse a suministrar a la RNN un determinado píxel–vector; por ejemplo, el del píxel central del parche, o la media espectral de los píxeles contenidos en él.

Para entender la utilidad de suministrar un píxel–vector a una RNN, es necesario tener presentes dos factores.

El primero de ellos es la complejidad espectral de la HSI, que puede llegar a abarcar decenas o cientos de bandas, y que por lo tanto puede contener una información muy rica. Por ello, es posible tomar un píxel–vector y particionarlo en subconjuntos de bandas, tomando cada uno de estos como un paso de tiempo a analizar para extraer características identificativas.

El segundo de ellos es que, en un proceso de entrenamiento de una red, las muestras se habitúan suministrar de forma conjunta en lotes, a pesar de que las redes se diseñen y se representen habitualmente –como en este documento– a nivel de muestra. Gracias a ello, una RNN puede emplear las características extraídas de diferentes píxel–vectores para identificar semejanzas espectrales entre muestras HSI que sean suministradas en diferentes instantes de tiempo. Así, puede tratar de emplear estos parecidos para realizar mejores predicciones en base decisiones anteriores.

Tomando en cuenta esta nueva posibilidad de análisis, se describirá ahora una mejora de la CNN 2D presentada anteriormente, tomando como base el esquema descrito en [24].

Como ya se explicó, la entrada del esquema de la CNN 2D es un parche de la imagen a clasificar. En ella, se procesaba este parche a través de múltiples capas convolucionales y de *pooling* para extraer características tanto espectrales como espaciales, con las que realizar la clasificación del parche, tal y como se refleja en la Figura 4.4 en la página 18. Esta será ahora la predicción y_1.

Adicionalmente, se va a incorporar en paralelo una RNN que se encargue de extraer información espectral del píxel central del parche, al cual se trata de asignar una categoría. La arquitectura correspondiente se refleja en la Figura 4.6. Como se puede ver en ella, el píxel–vector central se introduce en un módulo Long Short–Term Memory (LSTM) [44], que es el que proporciona el comportamiento recurrente a esta rama RNN. Este tipo de módulo recurrente ofrece determinadas ventajas frente a la RNN propuesta originalmente [23], al mitigar tanto el desvanecimiento como la explosión del gradiente durante el proceso de retropropagación.

Una vez el módulo recurrente se ha encargado de extraer la información espectral pertinente, todas estas características son introducidas, de forma análoga a la CNN, en una subred conformada por capas densas que se encarga de realizar una clasificación en base a ellas. Esta será la predicción y_2. El Cuadro 4.3 detalla las propiedades de cada capa de esta rama con el comportamiento de una RNN.

Cuadro 4.3: Detalles de la arquitectura de la rama RNN descrita para mejorar la CNN 2D. Las dimensiones de salida son las correspondientes a una entrada con dimensiones $H \times W \times B = 1 \times 1 \times 5$. Nótese que cada banda del píxel–vector de entrada se toma como un paso de tiempo, y el módulo LSTM produce un vector de características de 128 elementos por cada uno. Cabe destacar que el módulo LSTM ya incorpora funciones de activación dentro de sí mismo.

#	Tipo	Dim. salida	Activación	Tam. filtro	Desplazamiento
1	LSTM	128×5	–	–	–
2	Aplanamiento	640×1	–	–	–
3	Densa	256×1	ELU	–	–
4	Densa	$C \times 1$	Softmax	–	–

Con todo ello, la mejora de la CNN 2D no consiste en emplear estas dos predicciones por separado, sino que se fusionarán las características extraídas por ambas ramas de forma separada, para realizar una clasificación que se apoye en lo que

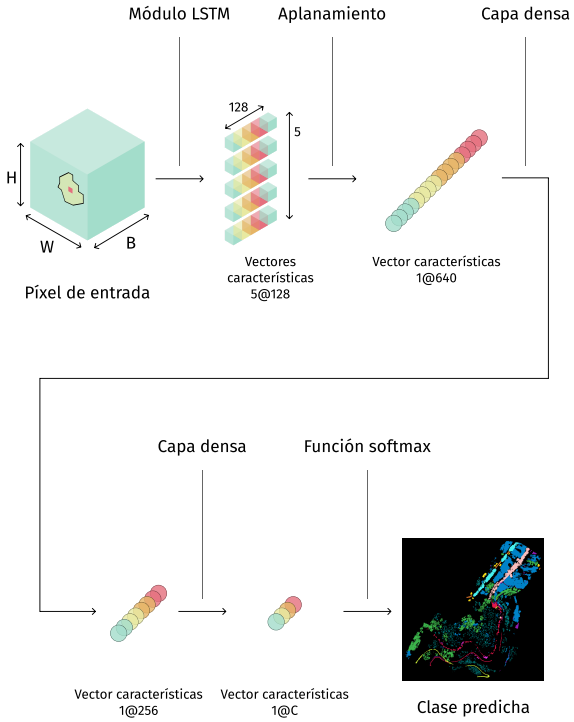


Figura 4.6: Esquema de la arquitectura de la rama RNN propuesta para mejorar la CNN 2D. Las dimensiones son las correspondientes a una entrada con dimensiones $H \times W \times B = 1 \times 1 \times 5$ –un píxel–vector con 5 bandas–.

cada rama sepa hacer mejor. Esta será la predicción y_3 –que debería ser la más fiable–. Concretamente, se combinarán con una concatenación los dos vectores de características generados por cada rama antes de la capa densa correspondiente a la clasificación con softmax. De este modo, se estará tratando de mejorar las capacidades de la CNN 2D al prestar mayor atención a la información espectral, al igual que se hizo con la CNN 3D descrita anteriormente.

Entonces, la arquitectura global de la CNN 2D + RNN descrita se puede ver reflejada en la Figura 4.7. Eso sí, cabe destacar que en la CNN 2D se generaba un vector con 512 características antes del paso que daba lugar a la clasificación, tal y como refleja el Cuadro 4.1 en la página 19. Ahora, se ha reducido el número de neuronas de la capa pertinente para que la rama de la CNN 2D produzca un vector de 256 características, de modo que al concatenarlo con la información de la rama RNN se acabe teniendo un vector de 512 características para la clasificación final. Al contribuir cada rama con 256 características, se pretende equilibrar la aportación de cada una, de modo que su importancia en la clasificación sea semejante.

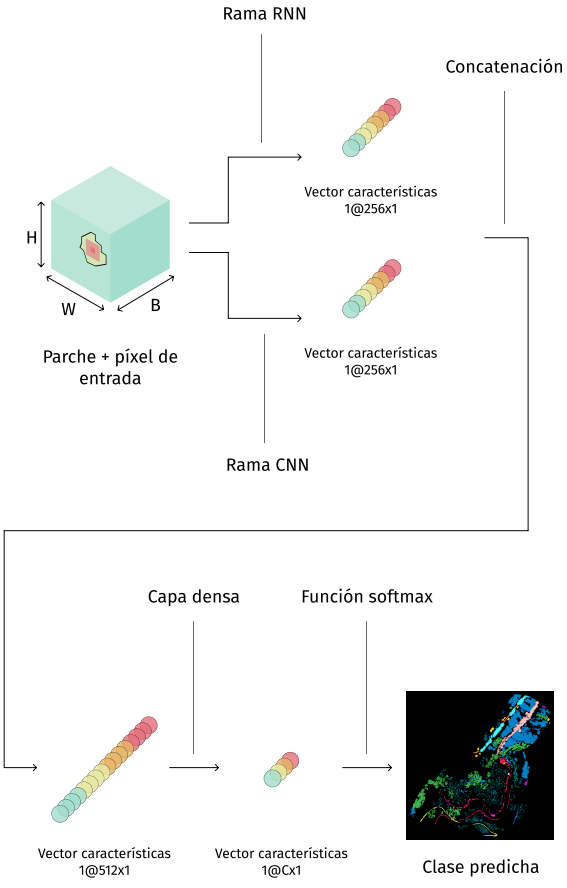


Figura 4.7: Esquema de la arquitectura de la CNN 2D + RNN descrita. Las dimensiones son las correspondientes a una entrada con dimensiones $H \times W \times B = 25 \times 25 \times 5$. Nótese cómo la clasificación final se realiza apoyándose en las características extraídas por ambas ramas.

Además, en la línea de potenciar este equilibrio de peso entre ambas ramas, la función de error $\mathcal{L}$ de este esquema tendrá en consideración todas las predicciones

definidas anteriormente, y_1, y_2, e y_3, definiéndose tal que

$$\mathcal{L} = \mathcal{L}^{y_1} + \mathcal{L}^{y_2} + \mathcal{L}^{y_3}, \tag{4.4}$$

en lugar de ser simplemente $\mathcal{L} = \mathcal{L}^{y_3}$. Es decir, se computará el error cometido por cada rama al predecir de forma individual la categoría de la información de entrada, y también se computará el error empleando las ramas de forma conjunta. Al final, lo que se pretende conseguir con esta definición es que los gradientes no tiendan a favorecer la evolución de tan solo una de las ramas durante el proceso de retropropagación en el entrenamiento.

Esquema D: CNN 2D residual

En los últimos años de la clasificación HSI, otro paradigma del *deep learning* que ha gozado también de cierta atención es el uso de técnicas de aprendizaje residual. El interés de estas es que permiten construir CNNs muy profundas que puedan extraer características todavía más complejas de las imágenes, y por lo tanto realizar un mejor trabajo.

Para entender la necesidad de este paradigma, es necesario tener en cuenta que las redes neuronales no se pueden hacer todo lo profundas que se desee, simplemente apilando unas capas tras otras. Esto suele resultar en problemas como la degradación de las características extraídas en el proceso de inferencia [19], o un deterioro del gradiente en el proceso de retropropagación [20].

Para tratar de solventar estos problemas, en [19] se propuso alterar la estructura tradicional de la CNN, de modo que existan "atajos" de capas de bajo nivel a capas de alto nivel. En base a esta idea, se puede definir un *bloque residual*, que habitúa ser el bloque básico de construcción de una CNN residual –ResNet–. La Figura 4.8 representa la arquitectura de uno de estos bloques.

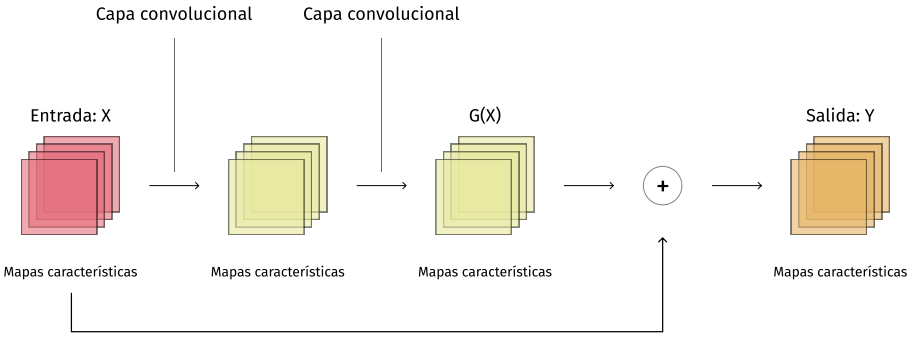


Figura 4.8: Esquema de la arquitectura de un bloque residual. Nótese cómo la información de entrada X se propaga también hacia la salida sin haber sufrido alteraciones.

Como se puede observar, la salida Y de este bloque no es exclusivamente el resultado $G(X)$ de aplicar a la entrada X las convoluciones pertinentes, sino que también se preserva en cierto modo la información de entrada sin alterar. Concretamente, la salida de un bloque residual se puede definir como

$$Y = G(X) + X. \tag{4.5}$$

De este modo, un bloque residual consigue propagar en cierta medida su información de entrada original a capas que se encuentren tras él, lo cual permitirá reducir problemas a la hora de hacer una CNN profunda, como la degradación de las características, o el deterioro del gradiente. Gracias a ello, se hace viable construir CNNs con decenas o incluso centenas de capas [19], y que puedan extraer características más discriminativas de la información para dar lugar a un mejor desempeño.

Entonces, tomando en cuenta este nuevo paradigma, se presentará ahora una CNN 2D residual basada en el esquema de [21], también con el objetivo de comprobar si efectivamente las técnicas de aprendizaje residual permitirán obtener unos esquemas de clasificación más precisos que los descritos hasta el momento.

Para ello, es necesario definir en primer lugar un componente mayor que un simple bloque residual, y que será sobre el que se base la CNN 2D residual. Este componente se denominará *etapa residual*, y consiste esencialmente en múltiples bloques residuales conectados unos tras otros. Dentro de cada etapa, las capas convolucionales contenidas emplearán el mismo número de filtros 2D, de modo que el número de mapas de características generados tras cada convolución se mantendrá constante dentro de cada etapa.

Además, teniendo en cuenta la definición dada en la Ecuación (4.5), es importante asegurarse de que X presente el mismo número de mapas de características que Y, para realizar la suma entre ellos. Dado que al primer bloque residual de una etapa nada le asegura que X contenga el mismo número de bandas que las que el bloque generará en Y, se introduce además al inicio de cada etapa residual una capa convolucional adicional que se encargue de cambiar la dimensionalidad de la información de entrada dada, de modo que coincida con la esperada.

El esquema contará con un total de tres etapas residuales, cada una con un número mayor de filtros que la anterior, para extraer características cada vez más abstractas y discriminativas.

Con todo ello, la salida de la primera subred de la CNN no será exclusivamente las características que se obtengan tras la última etapa residual. En lugar de ello, se fusionarán las características extraídas tras cada etapa residual –tres en total–, con el objetivo de proporcionar a la etapa de clasificación características de baja, media y alta abstracción. La fusión consistirá en una simple suma de los mapas de características salientes en cada etapa –como la suma dentro de un bloque residual–. Eso sí, será necesario introducir dos capas convolucionales adicionales que, antes de realizar la fusión, se encarguen de que la dimensionalidad de la salida de las dos primeras etapas coincida con la de la salida de la última etapa.

Para terminar esta primera subred, los mapas de características resultantes de la fusión serán procesados por una capa de *average–pooling* global. A diferencia de las capas de *pooling* presentadas anteriormente, la etiqueta de *global* indica que la operación reducirá todo lo posible la dimensionalidad espacial de cada mapa de características. Concretamente, el *pooling* generará un único valor por cada mapa, que consistirá en la media aritmética de los elementos que lo componen.

En último lugar, la segunda subred encargada de la clasificación constará de una única capa densa que tome el resultado del *pooling* global para predecir la categoría de la información de entrada, en base a las características que la CNN profunda haya conseguido identificar.

Toda la arquitectura descrita anteriormente se refleja visualmente en la Figura 4.9, por una mayor claridad. A su vez, el Cuadro 4.4 detalla las características de cada capa de la red. El principal motivo por el cual estas capas convolucionales 2D emplean un menor número de filtros en comparación a redes anteriores, es para tratar de mantener asumibles los tiempos de ejecución del esquema, que ya se verán notablemente incrementados en comparación con los demás clasificadores por la gran profundidad de la red. Tampoco se ha planteado el uso de convoluciones 3D por este mismo motivo.

Precisamente por esta última motivación, se ha incorporado un paso de normalización por lotes [45] en los bloques residuales que componen cada etapa, tras las capas convolucionales, y antes de las funciones de activación asociadas. Esto debería ayudar a acelerar la convergencia de la red [46], y que por lo tanto se requieran menos iteraciones en el entrenamiento.

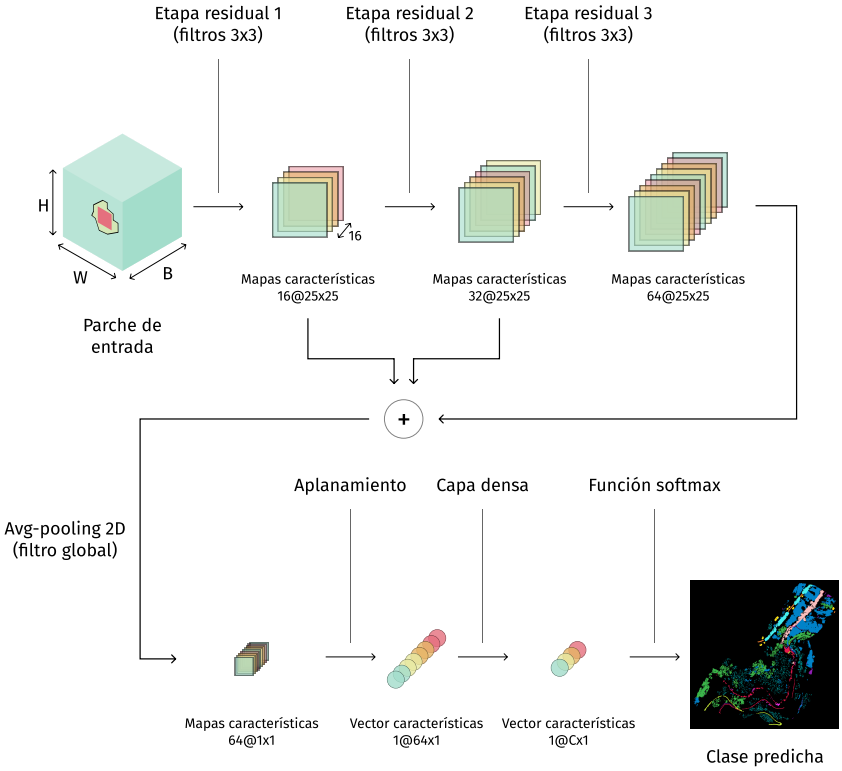


Figura 4.9: Esquema de la arquitectura de la CNN 2D residual descrita. Las dimensiones son las correspondientes a una entrada con dimensiones $H \times W \times B = 25 \times 25 \times B$. Nótese cómo las características empleadas en clasificación final se extraen de las tres etapas residuales, cada una centrada en extraer características más abstractas que la anterior.

Cuadro 4.4: Detalles de la arquitectura de la CNN 2D residual descrita. Las dimensiones de salida son las correspondientes a una entrada con dimensiones $H \times W \times B = 25 \times 25 \times B$. Antes de cada convolución se aplica un relleno en las dimensiones H y W de la información, para preservar el tamaño de las mismas. Cabe destacar que no se incluyen las dos capas convolucionales que adaptarían la salida de las dos primeras etapas residuales a la dimensionalidad de la tercera etapa para la fusión; sus filtros serían de tamaño 1×1, y les seguiría también la función ELU.

Etapa	**#**	**Tipo**	**Dim. salida**	**Activación**	**Tam. filtro**	**Desplazamiento**
1	1	Conv. 2D	$25 \times 25 \times 16$	ELU	1×1	1×1
	2–7	Conv. 2D	$25 \times 25 \times 16$	ELU	3×3	1×1
2	8	Conv. 2D	$25 \times 25 \times 32$	ELU	1×1	1×1
	9–14	Conv. 2D	$25 \times 25 \times 32$	ELU	3×3	1×1
3	15	Conv. 2D	$25 \times 25 \times 64$	ELU	1×1	1×1
	16–21	Conv. 2D	$25 \times 25 \times 64$	ELU	3×3	1×1
–	22	Avg-pool. 2D	$1 \times 1 \times 64$	–	25×25	25×25
–	23	Aplanamiento	64×1	–	–	–
–	24	Densa	$C \times 1$	Softmax	–	–

5 | Materiales y métricas experimentales

A lo largo de este capítulo se presentarán los recursos que serán necesarios para la posterior ejecución de la fase experimental del trabajo.

Por una parte, se introducirán las imágenes con las que probar los esquemas de clasificación presentados anteriormente, y se concretarán todas las métricas a utilizar para determinar sus desempeños. Por otra parte, también se detallará el entorno de ejecución que se empleará para los experimentos, tanto en cuanto a componentes hardware, como por dependencias de software.

5.1. Conjuntos de datos experimentales

Los esquemas de clasificación descritos anteriormente serán evaluados usando un conjunto de datos de imágenes de cuencas de ríos gallegos. Este material fue creado con el objetivo de monitorizar la interacción de las masas de vegetación nativa con estructuras artificiales y especies invasoras tales como los eucaliptos.

Las imágenes fueron tomadas en 2018 con una cámara multiespectral MicaSense RedEdge-MX [47], montada en un vehículo aéreo no tripulado, y volando a 120 m de altitud. Los sensores de esta cámara permiten capturar, cada segundo, cinco bandas espectrales[1] correspondientes a las longitudes de onda de 475 nm (azul), 560 nm (verde), 668 nm (rojo), 717 nm (*red edge*[2]), y 842 nm (infrarrojo cercano), con una resolución espacial de 8.2 cm/píxel.

Serán tres las imágenes a emplear de este conjunto de datos, y cubrirán diferentes localizaciones a lo largo del río Oitavén, situado en la provincia gallega de Pontevedra:

I1. **Río Oitavén:** región del río localizada en 42°22′15.48″ N 8°25′47.07″ W. Sus dimensiones son de 6689 × 6722 píxeles (*alto* × *ancho*), cubriendo un total de 758,30 × 759,41 m.

I2. **Riachuelo Ermidas:** cruce de este riachuelo con el río, en 42°22′48.43″ N 8°24′53.36″ W. Sus dimensiones son de 11924 × 18972 píxeles, cubriendo un total de 1373,60 × 2177,42 m.

I3. **Embalse de Eiras:** embalse que se interpone en el curso del río en 42°20′45.26 ″ N 8°30′10.81″ W, para suministrar agua a la ciudad de Vigo. Sus dimensiones son de 5176 × 18221 píxeles, cubriendo un total de 643,40 × 2256,55 m.

[1] Aunque pueda parecer en primera instancia que esta cantidad de bandas sea demasiado reducida, al poder llegar a contar las HSI con decenas o incluso cientos de ellas, estas imágenes multiespectrales permitirán obtener observaciones perfectamente válidas dado que: (1) los algoritmos empleados para el tratamiento de imágenes multi e hiperespectrales suelen ser los mismos, (2) y de hecho en muchos casos se reduce la dimensionalidad espectral de las imágenes hiperespectrales a unas pocas bandas, con Principal Component Analysis (PCA) [48] habitualmente, para hacerlas tratables ante ciertos procedimientos [16], [21], [24], [25].

[2] Este punto intermedio entre el rojo e infrarrojos habitúa ser de gran interés en imágenes con foco en capturar vegetación, puesto que enriquece notablemente la información recogida sobre ella [49].

Dadas las grandes dimensiones que presentan, se presupone un elevado coste para clasificarlas, así que serán unas candidatas apropiadas para determinar el impacto de incorporar los superpíxeles a esta tarea. Existen otros conjuntos de datos extremadamente populares y extendidos en el campo de la clasificación HSI, como [50], pero por desgracia presentan unas dimensiones muy reducidas, siendo en muchas ocasiones menores a 1000×1000 píxeles, y por ello no se han incorporado a la fase experimental de este trabajo.

En la Figura 5.1 se pueden observar las imágenes en color compuesto correspondientes a las tres imágenes de ríos, junto con sus datos de referencia. Estos últimos fueron construidos a lo largo de múltiples años, con la colaboración de expertos en silvicultura, y el grupo de HSI del CiTIUS [51]. El Cuadro 5.1 recoge las once categorías identificables en esta información de referencia, indicando además el número de muestras que se podrán tomar de cada categoría en cada imagen. Para segmentarlas se ha empleado SLIC, indicando una media de 800 píxeles/superpíxel, y un parámetro de compacidad de 40, tomando como referencia [29]. El umbral de parada se sitúa en un valor de 200, y se permiten un máximo de 10 iteraciones de forma complementaria.

Cuadro 5.1: Categorías identificables en las tres imágenes multiespectrales de cuencas de ríos gallegos. Se indica además el número de muestras identificables en cada imagen, tanto sin haberlas segmentado, como tras haberlo hecho empleando SLIC.

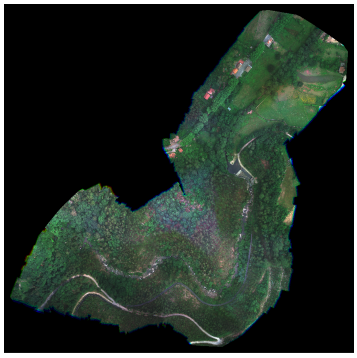
		Oitavén		**Ermidas**		**Eiras**	
# y color	**Categoría**	segmentos	píxeles	segmentos	píxeles	segmentos	píxeles
1.	Agua	649	322 603	351	163 930	1410	734 617
2.	Roble	3328	1 543 811	1695	804 040	4946	2 030 824
3.	Tejados	187	86 814	235	138 678	18	8232
4.	Prado	4413	2 604 511	5521	3 423 506	2243	744 978
5.	Asfalto	83	44 069	1520	737 409	150	85 209
6.	Tierra	331	112 644	342	123 416	350	96 935
7.	Piedra	670	382 437	775	174 088	1388	144 800
8.	Cemento	208	52 911	65	32 866	74	27 061
9.	Vegetación de ribera	321	137 516	0	0	135	46 927
10.	Eucalipto	830	499 623	3899	1 135 997	31	8451
11.	Pino	3927	1 076 619	582	184 547	292	95 132
	Muestras totales:	14 947	6 863 558	14 985	6 918 477	11 037	4 023 166

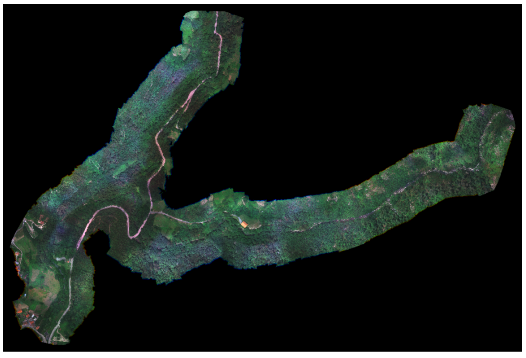
5.2. Métricas experimentales

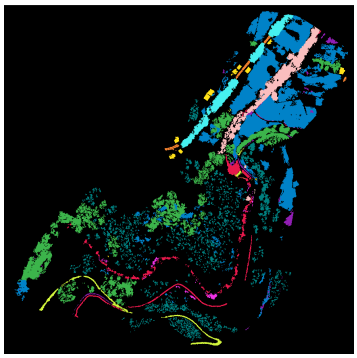
Durante la fase experimental de este trabajo, el desempeño de los esquemas de clasificación descritos se valorará tanto en términos de precisión, como en términos de rendimiento computacional.

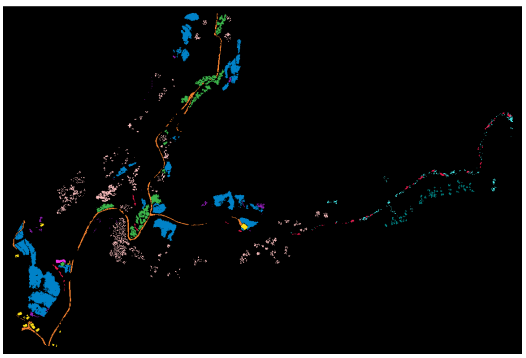
5.2.1. Métricas de precisión

Tras entrenar un clasificador sobre una imagen, siempre se probará su capacidad de generalización suministrándole muestras de la misma a las que no haya sido expuesto. A raíz de ello, se generará un mapa de clasificación que indicará la categoría predicha por el clasificador para cada píxel involucrado en las muestras de prueba –tanto realizando predicciones a nivel de píxel, como a nivel de superpíxel–. Estas predicciones serán comparadas con la información de referencia disponible, con el objeto de determinar en qué medida se aproximan.



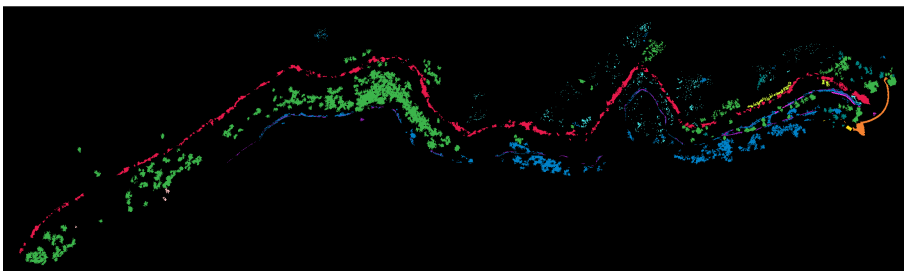






(a) Río Oitavén

(b) Riachuelo Ermidas



(c) Embalse de Eiras

Figura 5.1: Imágenes en color compuesto de las imágenes multiespectrales de cuencas de ríos gallegos, junto con la correspondiente información de referencia de cada una.

Para ello, se emplearán las siguientes medidas estándar en el ámbito de la clasificación HSI [52]:

- **Overall Accuracy (OA):** porcentaje total de píxeles correctamente clasificados.
- **Average Accuracy (AA):** media de los porcentajes de píxeles correctamente clasificados dentro de cada categoría.
- **Coeficiente kappa de Cohen:** porcentaje de acierto total en píxeles corregido tras tener en cuenta cuántos aciertos se pueden esperar por mero azar.

5.2.2. Métricas de rendimiento

El entrenamiento y prueba de un esquema sobre una imagen también será evaluado en cuanto a tiempo de ejecución requerido, y eficiencia implicada. Concretamente:

- Si el experimento va a trabajar con superpíxeles, se registrará el tiempo requerido para segmentar la imagen con SLIC.
- Se registrará el tiempo requerido para preprocesar la imagen de cara al entrenamiento y prueba. Esto abarca desde la lectura de la imagen, hasta la identificación y registro de las muestras contenidas en ella; se incluye el tiempo de segmentación en caso de generarlas en base a superpíxeles.
- Se registrará el tiempo requerido para entrenar el esquema.
- Se registrará el tiempo requerido para probar el esquema.
- Y se determinarán las aceleraciones –o *speedups*– experimentados en el entrenamiento y prueba a raíz de emplear muestras en base a superpíxeles, en vez de a nivel de píxel. Esta métrica indica cuántas veces un tiempo optimizado t_o es más veloz que un tiempo base t_b, tal que $speedup = t_b/t_o$.

5.3. Entorno de pruebas

5.3.1. Configuración de hardware

El computador sobre el cual se lanzarán todos los experimentos dispondrá de dos procesadores Intel Xeon Gold 5220 [53], 192 GB de memoria RAM, y la tarjeta gráfica NVIDIA Tesla V100S con 32 GB de VRAM [54]. Esta última permitirá acelerar el entrenamiento de los esquemas basados en *deep learning*, entrenándolos y probándolos utilizando aritmética de coma flotante en simple precisión.

5.3.2. Configuración de software

En cuanto al entorno de software asociado, se empleará el sistema operativo CentOS 8.2.2004 [55], junto con Python 3.8.5 [56], GNU Compiler Collection 9.3.0 [57], y CUDA 11.2.67 [58] con soporte para cuDNN [59]. Esta última característica también permitirá acelerar todavía más la ejecución de los esquemas de *deep learning* sobre la GPU.

El lenguaje Python se empleará para prácticamente la totalidad de los códigos desarrollados durante este trabajo. Los paquetes más destacables de los que dependen serían PyTorch 1.8.0 [60], NumPy 1.19.2 [61], y scikit–learn 0.24.1 [62]. Mientras que PyTorch proporcionará la base sobre la cual elaborar los esquemas de

deep learning, scikit–learn proporcionará directamente la implementación de SVM a probar.

De forma complementaria, el lenguaje C++ [63] se utilizará para desarrollar códigos auxiliares que se comuniquen con los códigos principales, para acelerar el preprocesamiento de las imágenes experimentales. Cabe destacar, sobre todo, que la implementación de SLIC presentada en el Apartado 3.2 en la página 6 se ha preparado sobre C++ en lugar de Python, para tratar de minimizar el impacto de incorporar esta etapa al flujo de clasificación HSI. Por este mismo motivo, se ha adaptado además a un modelo de ejecución multihilo a través de la interfaz de programación OpenMP [64].

6 | Pruebas

Este capítulo presentará y desarrollará el plan experimental que permitirá comprobar el impacto de la segmentación en superpíxeles sobre los diferentes esquemas basados en *deep learning* presentados anteriormente.

En primer lugar, será necesario realizar una serie de aclaraciones sobre cómo se guiará el preprocesamiento de las imágenes, y el proceso de entrenamiento de los diferentes clasificadores a evaluar. Tras ello, se concretarán los experimentos en cuestión a realizar, y se terminará el capítulo recopilando todos los resultados experimentales que se puedan extraer de los mismos.

6.1. Políticas de realización de los experimentos

6.1.1. De preprocesamiento de las imágenes

Para tratar de minimizar todo lo posible el impacto de la incorporación de la segmentación antes del flujo de clasificación HSI, todas las imágenes serán segmentadas empleado 16 hilos colaboradores en SLIC. Esto es posible gracias a la programación multihilo realizada sobre la interfaz OpenMP.

Los parámetros a emplear en la segmentación son los concretados en el Apartado 5.1 en la página 29: una media de 800 píxeles/superpíxel, un parámetro de compacidad de 40, un valor de 200 para el umbral de parada, y un máximo de 10 iteraciones. En base a este tamaño medio de superpíxeles, se ha decidido optar por un parámetro de vecindad $N = 25$, al igual que en [29]. Este valor influirá no solo a la hora de tomar muestras de los superpíxeles, sino también al trabajar a nivel de píxel, acorde a las explicaciones del Apartado 3.3 en la página 10.

Cabe destacar por otra parte que, en el caso de la CNN 2D + RNN, se suministrará a la rama recurrente el píxel–vector central del parche, tomando cada una de las cinco bandas existentes como un paso de tiempo.

Además, como es habitual, todas las muestras que se extraigan de las imágenes experimentales serán normalizadas para facilitar a los clasificadores su procesamiento. Concretamente, serán normalizadas al rango $[0, 1]$.

6.1.2. De entrenamiento de SVM

Para tratar de obtener el mejor resultado posible al entrenar un clasificador SVM, es crucial realizar una buena selección de los diferentes parámetros que rigen su comportamiento [3]:

- Un parámetro de gran importancia se suele denominar C, y determina la medida en que penalizar el error cometido con muestras que no se consigan clasificar correctamente. Se probarán los valores $C = \{128, 1024, 8196\}$.
- Como se explicó en el Apartado 4.1 en la página 13, es habitual emplear kernels para proyectar las muestras a tratar a un espacio de mayor dimensionalidad

para mejorar la separabilidad entre las diferentes categorías. En este caso, se escogerá el kernel de función de base radial; se realizaron una serie de pruebas comparando este kernel con el polinomial y el sigmoidal, y ha sido el que mejores resultados ha reportado siempre.

- Un último parámetro de importancia asociado a este kernel que se ha escogido se trata de γ, ya que influye en su comportamiento. Se probarán los valores $\gamma = \{1, 10, 100, {}^1/_B, {}^1/_{B \cdot \sigma_x^2}\}$, siendo B el número de bandas de la imagen, y σ_x^2 la varianza de las muestras de entrenamiento.

De entre todas las posibles configuraciones de parámetros, la óptima se determinará siguiendo un proceso de validación cruzada. En este, se proponen tantos clasificadores como combinaciones de parámetros existan. Por otra parte, las muestras de entrenamiento se dividen en k partes, empleando $k-1$ partes para el entrenamiento de cada clasificador candidato, y la restante para determinar su precisión provisional. Cada clasificador candidato es entrenado un total de k veces, probando todas las combinaciones posibles de partes de entrenamiento. Al final, se escoge como combinación de parámetros óptima la de aquel clasificador candidato que reporte la mejor media de sus k precisiones provisionales.

En este caso, se tomará un valor de $k = 5$. Además cabe destacar que, para controlar este proceso de validación cruzada, la resolución de cada configuración de parámetros estará limitada a 5×10^5 iteraciones –encontrar un clasificador SVM consiste en resolver un problema de optimización matemática–. La intención de ello es evitar que una "mala" combinación de parámetros que requiera demasiado tiempo para converger no perjudique demasiado el entrenamiento de SVM.

La implementación de `scikit--learn` para SVM se ejecuta exclusivamente sobre CPU, por lo que se lanzarán además un total de 16 procesos en paralelo para acelerar el proceso de entrenamiento del clasificador. En cambio, la etapa de prueba deberá realizarse con un único proceso, así que estos tiempos de procesamiento podrán verse notablemente impactados por ello.

6.1.3. De entrenamiento de los esquemas de *deep learning*

Continuando ahora con los esquemas de *deep learning* descritos, es importante tener presente que el proceso de aprendizaje de una red neuronal consiste, esencialmente, en un proceso iterativo que se repite hasta alcanzar un número determinado de iteraciones –épocas–, o una precisión objetivo. En cada época se suministran todas las muestras de entrenamiento a la red, en lotes de un tamaño predefinido –128 en este caso–, y los parámetros de la red se van ajustando con un proceso de retropropagación [65] en función del error cometido por la red entre la salida para cada lote, y la salida esperada para el mismo.

Un parámetro crucial para este tipo de entrenamiento es el ratio de aprendizaje, comúnmente representado por α. Este controla la medida en que se actualizan los parámetros de la red durante la retropropagación. Un valor muy alto acelerará el proceso de entrenamiento, pero se corre el riesgo de que la red no sea capaz de estabilizarse en un óptimo local, o que directamente se desvíe del mismo. Por el contrario, un valor muy bajo corre el riesgo de que el entrenamiento sea demasiado lento.

Siguiendo las propuestas sobre las que se basan los esquemas de *deep learning* anteriormente presentados [21], [24], [29], se escogerá un ratio $\alpha = 0{,}001$, pero se incorporará además un mecanismo que reducirá α en un orden de magnitud cada vez que se alcance una meseta en el error de validación cometido. Este error se determina, tras cada época, a partir del acierto de la red sobre el conjunto de validación. De este modo, se tratará de incentivar un rápido aprendizaje de las

redes al comienzo, pero se permitirá reducir el factor de aprendizaje al acercarse a mínimos locales para facilitar la convergencia hacia ellos, con el objetivo de alcanzar unos mejores resultados.

A lo largo de los años, se han desarrollado una variedad de algoritmos de optimización a emplear para entrenar redes neuronales, y que habitúan seguir la filosofía del descenso del gradiente. El ejemplo más notable de ello es el descenso del gradiente estocástico [66]. Sin embargo, en los últimos años Adaptive Moment Estimation (Adam) [67] ha resultado ser una opción muy popular dentro del campo del *deep learning*, sobre todo para entrenar redes complejas y profundas –en definitiva, con muchos parámetros–, dado que adapta el ratio de aprendizaje vigente a los diferentes parámetros que componen la red para tratar de alcanzar un buen balance. Un parámetro que se actualice constantemente se verá menos afectado por el ratio de aprendizaje, y un parámetro que se altere de forma ocasional se verá más alterado. Gracias a sus características, se ha comprobado cómo Adam resulta ser una de las mejores opciones a la hora de escoger un algoritmo basado en el descenso del gradiente [67], y por ello se empleará durante el entrenamiento de todos los esquemas de *deep learning* aquí descritos, con parámetros $\beta_1 = 0{,}9$ y $\beta_2 = 0{,}999$.

Por supuesto, es necesario concretar en algún momento de qué valores parten los parámetros –pesos y sesgos– a ajustar en las diferentes redes. Un primer enfoque a esta consideración habituaba ser la simple inicialización aleatoria de los valores de los parámetros, pero este comenzó a dificultar el entrenamiento de redes que ya comenzasen a contar con múltiples capas [68]. Es por ello que surgieron nuevos métodos de inicialización como el propuesto en [68], comúnmente denominado como Inicialización de Xavier –o de Glorot–, y que será al que se recurra en estos experimentos.

Cabe destacar que este anterior método se empleará para la inicialización de los parámetros de capas densas y de capas convolucionales. El único otro elemento incluido en los esquemas de clasificación descritos, y que requiere de la inicialización de sus parámetros, es el módulo LSTM. Este no es compartible con la Inicialización de Xavier, así que se empleará el método descrito en [69], tal y como se hace en la publicación de inspiración para la rama RNN [24].

Una vez determinadas todas estas opciones, es necesario concretar en último lugar el número de épocas durante las que serán entrenadas las diferentes redes. No se han tomado los valores especificados en sus respectivas publicaciones, dado que estas simplemente se han tomado como inspiración para elaborar los esquemas aquí descritos, así que sus configuraciones no tienen por qué ser necesariamente extrapolables a estos experimentos. Tras una serie de pruebas, se han determinado las cantidades de épocas reflejadas en el Cuadro 6.1 para cada red. Se ha tratado de escoger un valor que permita a la red acercase a un mínimo local una vez ha consumido cerca del 50 % de épocas disponibles, y se dejan el resto de épocas para tratar de converger a dicho mínimo. Para esto último, se ha monitorizado el progreso de las redes durante las pruebas mediante los errores de entrenamiento y de validación cometidos a cada paso.

Cuadro 6.1: Número de épocas de entrenamiento designadas para cada clasificador de *deep learning* a probar.

	CNN 2D	**CNN 3D**	**CNN 2D + RNN**	**CNN 2D residual**
# de épocas	100	100	160	200

6.2. Descripción de los experimentos

Para evaluar con detenimiento el impacto de la incorporación de la etapa de segmentación a los esquemas de clasificación descritos anteriormente, se llevará a cabo una serie de experimentos tal que:

1. Los cinco clasificadores se entrenarán y probarán con cada una de las tres imágenes presentadas, para determinar sus desempeños finales en cuanto a precisión y consumo de recursos computacionales.

2. Cada clasificador se entrenará un total de seis veces sobre cada imagen, variando las condiciones de partida:

   a) Por una parte, se puede hacer una distinción entre entrenar habiendo aplicado una segmentación a la imagen, y sin haberlo hecho; es decir, entre emplear muestras a nivel de píxel, o muestras a nivel de superpíxel.

   b) Por otra parte, se probará a suministrar tres diferentes cantidades de muestras de entrenamiento a los clasificadores, para comprobar en qué medida las observaciones se pueden ver afectadas por si se puede suministrar poca o mucha información de aprendizaje a los esquemas.

   Concretamente, en el entrenamiento con segmentación se suministrarán al clasificador el 15 %, 35 % y 55 % de las muestras disponibles. Cada uno de estos tres entrenamientos tendrá su versión análoga sin segmentación, y se suministrarán tantas muestras a nivel de píxel como sean necesarias para igualar el número absoluto de muestras de entrenamiento en base a superpíxeles, de modo que las condiciones de partida sean comparables, tal y como se sugirió en el Apartado 3.3 en la página 10. El subconjunto de muestras de entrenamiento se escogerá aleatoriamente de entre todas las muestras disponibles.

   Las muestras restantes se dividirán a su vez en otros dos subconjuntos, también de forma aleatoria. Un primer subconjunto de validación tomará el 5 % de las muestras en base a superpíxeles, y el mismo número absoluto en base a píxeles, para monitorizar el progreso de los esquemas de *deep learning* durante su entrenamiento y tratar de identificar el fenómeno de sobreajuste –*overfitting*–, así como el fenómeno opuesto de *underfitting*. En último lugar, todas las muestras restantes se incorporarán al subconjunto de prueba sobre el que se evaluará la capacidad de generalización de los esquemas una vez hayan sido entrenados.

Dada la aleatoriedad inherente a estos experimentos, tanto por la elección de los subconjuntos de muestras, como por la inicialización aleatoria de los parámetros –pesos y sesgos– de las redes neuronales, se repetirán todas las pruebas un total de 10 veces, bajo las mismas condiciones experimentales, y se reportarán los valores medios de las métricas. Así se tratará de minimizar el impacto de dicha aleatoriedad durante el análisis experimental, para poder extraer unas conclusiones más sólidas.

6.3. Resultados experimentales

A continuación, se reflejan en los Cuadros 6.2 a 6.4 los resultados obtenidos durante la fase experimental sobre las imágenes *Oitavén*, *Ermidas*, y *Eiras* respectivamente.

Cuadro 6.2: Resultados experimentales de los cinco clasificadores a evaluar, sobre la imagen *Río Oitavén*. Los porcentajes de muestras usadas en el entrenamiento se calculan en base a la cantidad de muestras disponibles al trabajar con superpíxeles. Las mejores precisiones dentro de cada porcentaje y modo de procesar la imagen –en base a píxeles o superpíxeles– se resaltan en **negrita**.

% muestras entrenamiento		**SVM**		**CNN 2D**		**CNN 3D**		**CNN 2D + RNN**		**CNN 2D residual**	
		píxeles	superpíxeles	píxeles	superpíxeles	píxeles	superpíxeles	píxeles	superpíxeles	píxeles	superpíxeles
15 %	OA (%)	72.87	82.77	81.75	82.89	81.27	82.79	82.64	83.45	**88.61**	**88.89**
	AA (%)	60.86	69.33	69.60	67.11	69.25	66.57	71.34	69.93	**81.72**	**82.20**
	k (%)	63.97	74.83	76.16	74.90	75.53	74.82	77.31	75.74	**85.31**	**83.98**
	tiempo entrenamiento (s)	6.72	6.85	57.96	56.59	63.28	62.74	102.89	98.15	213.52	202.99
	speedup entrenamiento	0.98x		1.02x		1.01x		1.05x		1.05x	
	tiempo prueba (s)	535.37	1.92	467.37	1.13	527.26	1.20	486.61	1.17	719.17	1.60
	speedup prueba	278.99x		413.96x		439.41x		417.01x		450.48x	
35 %	OA (%)	74.78	85.49	84.37	85.75	84.78	85.69	85.49	86.28	**93.26**	**94.65**
	AA (%)	64.54	74.78	74.18	73.41	74.88	72.53	76.46	74.14	**89.12**	**89.18**
	k (%)	66.55	78.87	79.58	79.12	80.14	79.07	81.06	79.84	**91.26**	**92.26**
	tiempo entrenamiento (s)	21.95	23.15	77.12	73.19	86.97	87.63	143.80	149.80	356.08	354.47
	speedup entrenamiento	0.95x		1.05x		0.99x		0.96x		1.00x	
	tiempo prueba (s)	1084.55	2.21	465.69	0.92	535.75	0.98	483.55	1.00	713.29	1.35
	speedup prueba	490.77x		506.27x		547.97x		481.47x		529.60x	
55 %	OA (%)	75.75	87.03	85.60	87.58	86.46	86.84	86.92	87.54	**95.59**	**95.49**
	AA (%)	66.62	78.61	75.41	75.23	78.27	79.13	78.81	77.75	**92.56**	**91.71**
	k (%)	67.94	81.11	81.17	81.75	82.35	80.80	82.93	81.76	**94.27**	**93.45**
	tiempo entrenamiento (s)	49.51	56.42	98.67	95.85	111.54	116.13	183.94	185.02	515.04	504.35
	speedup entrenamiento	0.88x		1.03x		0.96x		0.99x		1.02x	
	tiempo prueba (s)	1572.19	2.18	461.66	0.74	523.69	0.80	489.72	0.79	707.15	1.03
	speedup prueba	720.77x		620.67x		668.81x		616.33x		689.00x	

Cuadro 6.3: Resultados experimentales de los cinco clasificadores a evaluar, sobre la imagen *Riachuelo Ermidas*. Los porcentajes de muestras usadas en el entrenamiento se calculan en base a la cantidad de muestras disponibles al trabajar con superpíxeles. Las mejores precisiones dentro de cada porcentaje y modo de procesar la imagen –en base a píxeles o superpíxeles– se resaltan en **negrita**.

% muestras entrenamiento		**SVM**		**CNN 2D**		**CNN 3D**		**CNN 2D + RNN**		**CNN 2D residual**	
		píxeles	superpíxeles	píxeles	superpíxeles	píxeles	superpíxeles	píxeles	superpíxeles	píxeles	superpíxeles
15 %	OA (%)	84.51	92.81	91.56	94.22	90.86	93.68	92.81	95.10	**96.45**	**97.53**
	AA (%)	63.75	69.36	73.14	73.38	71.00	69.55	75.94	74.52	**82.73**	**83.86**
	k (%)	77.42	84.75	87.90	87.47	86.89	86.36	89.70	89.39	**94.93**	**94.68**
	tiempo entrenamiento (s)	5.15	5.06	59.28	58.33	69.46	65.70	107.69	101.48	224.49	207.07
	speedup entrenamiento	1.02x		1.02x		1.06x		1.06x		1.08x	
	tiempo prueba (s)	382.56	1.84	469.37	1.19	537.99	1.27	492.79	1.20	724.32	1.61
	speedup prueba	207.40x		393.79x		424.77x		412.18x		449.99x	
35 %	OA (%)	86.26	94.59	94.76	96.43	94.50	95.90	95.37	96.57	**97.82**	**97.91**
	AA (%)	68.92	74.50	80.39	79.17	79.84	78.61	81.82	80.14	**86.85**	**85.95**
	k (%)	80.12	88.49	92.51	92.28	92.13	91.17	93.38	92.58	**98.89**	**95.59**
	tiempo entrenamiento (s)	11.83	14.34	82.70	76.56	92.75	90.27	148.61	145.18	361.42	355.45
	speedup entrenamiento	0.83x		1.08x		1.03x		1.02x		1.02x	
	tiempo prueba (s)	676.71	1.85	467.28	0.98	541.25	1.01	494.04	1.00	716.50	1.31
	speedup prueba	365.82x		474.94x		536.16x		495.79x		548.24x	
55 %	OA (%)	87.00	95.16	95.82	97.01	95.65	96.82	96.17	97.27	**98.26**	**98.45**
	AA (%)	70.80	77.00	82.39	81.53	82.69	81.60	83.45	82.14	**87.63**	**87.06**
	k (%)	81.23	89.74	94.02	93.57	93.79	93.17	94.53	94.07	**97.51**	**96.67**
	tiempo entrenamiento (s)	24.43	27.73	101.80	98.47	119.28	117.97	190.73	185.41	523.03	511.50
	speedup entrenamiento	0.88x		1.03x		1.01x		1.03x		1.02x	
	tiempo prueba (s)	1014.13	1.84	470.47	0.76	534.38	0.79	492.80	0.78	719.11	1.02
	speedup prueba	550.46x		621.13x		676.92x		630.99x		706.80x	

Cuadro 6.4: Resultados experimentales de los cinco clasificadores a evaluar, sobre la imagen *Embalse Eiras*. Los porcentajes de muestras usadas en el entrenamiento se calculan en base a la cantidad de muestras disponibles al trabajar con superpíxeles. Las mejores precisiones dentro de cada porcentaje y modo de procesar la imagen –en base a píxeles o superpíxeles– se resaltan en **negrita**.

% muestras entrenamiento		**SVM**		**CNN 2D**		**CNN 3D**		**CNN 2D + RNN**		**CNN 2D residual**	
		píxeles	superpíxeles	píxeles	superpíxeles	píxeles	superpíxeles	píxeles	superpíxeles	píxeles	superpíxeles
15 %	OA (%)	86.35	91.35	91.57	90.96	91.23	91.59	91.85	92.18	**94.44**	**95.93**
	AA (%)	55.85	51.99	67.96	60.88	65.29	56.77	68.72	64.28	**81.52**	**79.60**
	k (%)	79.31	84.16	87.37	83.67	86.87	84.77	87.82	85.81	**91.70**	**92.76**
	tiempo entrenamiento (s)	4.30	4.73	60.43	56.04	61.78	59.37	105.42	96.51	186.83	173.21
	speedup entrenamiento	0.91x		1.08x		1.04x		1.09x		1.08x	
	tiempo prueba (s)	154.12	1.53	273.91	1.03	314.94	1.05	287.74	1.05	423.26	1.33
	speedup prueba	101.00x		266.97x		299.86x		274.85x		318.71x	
35 %	OA (%)	87.39	92.90	92.96	93.53	92.90	92.99	93.37	93.75	**95.74**	**97.36**
	AA (%)	59.95	61.23	71.60	71.08	72.74	63.95	74.68	74.27	**87.22**	**90.06**
	k (%)	80.91	87.13	89.47	88.27	89.38	87.36	90.08	88.74	**93.65**	**95.31**
	tiempo entrenamiento (s)	8.30	12.22	74.89	67.29	80.74	79.71	133.98	126.08	292.68	289.54
	speedup entrenamiento	0.68x		1.11x		1.01x		1.06x		1.01x	
	tiempo prueba (s)	287.71	1.60	272.90	0.85	313.09	0.91	286.14	0.88	416.77	1.13
	speedup prueba	180.27x		319.45x		345.63x		326.17x		368.79x	
55 %	OA (%)	88.00	92.88	94.19	94.21	94.08	93.82	94.64	95.07	**97.56**	**98.08**
	AA (%)	62.89	62.64	77.74	75.57	76.96	70.50	79.93	80.80	**93.01**	**94.64**
	k (%)	81.89	87.24	91.33	89.54	91.16	88.84	92.00	91.08	**96.38**	**96.64**
	tiempo entrenamiento (s)	13.86	24.48	82.80	80.11	99.45	98.18	156.95	154.45	402.48	398.98
	speedup entrenamiento	0.57x		1.03x		1.01x		1.02x		1.01x	
	tiempo prueba (s)	391.44	1.48	274.72	0.69	315.11	0.74	287.84	0.72	419.71	0.90
	speedup prueba	265.15x		396.94x		428.08x		400.19x		464.40x	

6.3.1. Resultados en cuanto a métricas de precisión

A nivel global, puede observarse en todos los casos una clara tendencia a que se incrementen las precisiones a medida que se desciende en cada cuadro. Es decir, al incrementar el número de muestras empleadas durante el entrenamiento de los clasificadores, estos reportan mejores resultados al ser probados. Esto es perfectamente coherente y esperable. Al suministrar una información más variada al clasificador durante su entrenamiento, este debería ser capaz de extraer un conocimiento más profundo de los datos a los que se enfrenta, para poder realizar mejor su tarea de clasificación más adelante.

Poniendo ahora un foco más preciso sobre los cuadros, cada uno puede ser separado en cinco columnas, una por cada clasificador. Al compararlas entre ellas, se pueden analizar las capacidades que unos clasificadores consiguen alcanzar frente a otros:

- A lo largo de las tres imágenes, SVM resulta ser el clasificador que proporciona las precisiones más bajas, estando, por norma general, a una considerable distancia de la CNN 2D y la 3D. Esto se encuentra en la línea de lo esperable, dado que SVM se trata de una herramienta de aprendizaje automático tradicional.

- La CNN 2D y la 3D resultan ser a su vez los esquemas de *deep learning* cuyas precisiones son más bajas, aunque siguen siendo de todos modos considerablemente superiores a SVM. Ambas redes tienden a reportar precisiones muy semejantes en todos los experimentos, así que no es posible determinar que ninguna sea mejor que la otra. Eso sí, es posible concluir que el empleo de convoluciones 3D, con el objetivo de prestar más atención a las características espectrales de las imágenes, no ha resultado ser una técnica de utilidad con estas imágenes, al no reportar unas precisiones superiores a la CNN 2D.

- Continuando con la CNN 2D + RNN, esta sí ha sido capaz de reportar en casi todas las situaciones mejores resultados que las CNNs 2D y 3D. No son diferencias muy pronunciadas, estando en general por debajo del 2 % de incremento, pero siguen siendo mejoras en todo caso. El objetivo de este esquema de clasificación era prestar más atención a la información espectral de las imágenes gracias a la incorporación del módulo LSTM, y esta mejora verifica la utilidad esperada para este componente.

- En último lugar, la CNN 2D residual es la vencedora indiscutible en cuanto a precisiones. Es algo que se cumple independientemente de la imagen y de la cantidad de muestras de entrenamiento. Además, se distancia en muchos casos de forma notable de los demás esquemas; es algo destacable sobre todo en las imágenes de *Oitavén* y de *Eiras*. De este modo, se comprueba cómo la red ha podido sacar partido de la gran profundidad que presenta para extraer características más complejas y completas de la información a tratar, para realizar mejor su tarea de clasificación.

Las diferencias de precisión entre los múltiples clasificadores van acorde a lo que otros autores del campo de la clasificación HSI ya han comprobado en diversas comparativas de esquemas, trabajando a nivel de píxel [5]-[7]. Como es coherente, los métodos basados en *deep learning* consiguen alcanzar mejores resultados que un método tradicional como SVM. Dentro de los primeros, los esquemas más simples, como las CNNs 2D y 3D, se encuentran un escalón por debajo de clasificadores que incorporan nuevos paradigmas para mejorar la calidad de las características extraídas. Por una parte, un módulo recurrente consigue mejorar ligeramente las precisiones al tomar más partido de la información espectral de las HSI. Por otra

parte, una ResNet como la aquí descrita dará lugar a unas clasificaciones muy precisas, al poder tomar partido de su gran profundidad y complejidad para extraer información más elaborada de cara a la clasificación.

En tercer lugar, es posible profundizar todavía más en la información de los cuadros si se comparan los valores de un clasificador concreto entre sí, dado que se presentan tanto los de sus experimentos con segmentación, como los de sus experimentos sin ella. De entre los cinco clasificadores, el único en el que se pueden apreciar diferencias significativas entre estos dos casos es SVM, ya que en los demás las precisiones no se ven realmente afectadas si se entrena con muestras en base a píxeles o en base a superpíxeles.

En cuanto a los clasificadores basados en *deep learning*, como incorporar la segmentación no da lugar a ninguna variación notable entre las precisiones que consiguen alcanzar, se seguiría sosteniendo la jerarquía que se comentaba anteriormente al ordenar los clasificadores en base a precisiones.

Haciendo más énfasis en el caso de SVM, aunque lo que sucede pueda parecer extraño en primera instancia, tiene sentido si se recuerda el modo en que se generan las muestras en base a superpíxeles, descrito en el Apartado 3.3 en la página 10. SVM es un clasificador que toma un píxel–vector como muestra. En el caso de generar muestras en base a píxeles, cada muestra de entrenamiento contendrá la información de un único píxel de la imagen. En cambio, al generar muestras en base a superpíxeles, el píxel–vector que caracteriza una muestra se habrá compuesto como la media espectral de múltiples píxeles que se puedan encontrar en torno al píxel central del segmento, y por lo tanto contendrá una información más rica. Esto es algo que no tiene tanta relevancia para los esquemas basados en redes neuronales, dado que todos toman un parche de la imagen como entrada, y esto implica que cada muestra se componga, sí o sí, de varios píxeles tanto si se ha segmentado la imagen, como si no se ha hecho.

En resumen, los resultados observados indican que emplear una segmentación SLIC, tal y como se ha planteado en este trabajo:

- Permite enriquecer las muestras de clasificadores basados en píxel–vectores –como SVM, y probablemente otros como una CNN 1D [15]–. Esto les permite realizar unas clasificaciones más precisas.
- Y en cuanto a los clasificadores que trabajan con parches, aunque la segmentación no haya mejorado sus precisiones, tampoco se han empeorado. Por lo tanto, en cuanto a fiabilidad de predicciones se refiere, se puede emplear de forma indiferente una imagen sin alterar, o habiéndola segmentado.

6.3.2. Resultados en cuanto a métricas de rendimiento

Siguiendo una estructura análoga a la del Apartado 6.3.1, al partir de una visión global de los cuadros es posible observar cómo emplear un mayor número de muestras de entrenamiento incrementa, de manera directamente proporcional, el tiempo de entrenamiento de los clasificadores. Es algo coherente, ya que se incrementa la cantidad de información a procesar durante esta etapa.

Pasando ahora a analizar y comparar las columnas de los cinco clasificadores entre sí:

- SVM siempre requiere el menor tiempo de entrenamiento de entre todos, incluso incluyendo todo el proceso de validación cruzada, el cual implica generar 75 clasificadores candidatos en cada experimento[1]. Es algo coherente, puesto

[1]Existen 15 posibles combinaciones de todos los parámetros a probar, y se generan $k = 5$ divisiones diferentes de las muestras de entrenamiento para la validación cruzada, dando lugar a un total de $15 \times 5 = 75$ resoluciones.

que un gran inconveniente de los esquemas basados en *deep learning* son los grandes costes de ejecución asociados a ellos [8], agravados por el hecho de que el entrenamiento de una red neuronal sea también un proceso iterativo.

Por otra parte, es conveniente aclarar que la etapa de prueba de SVM no resulta tan rápida, siendo incluso a veces más lenta que la etapa de prueba de la CNN 2D, como consecuencia de que se lleve a cabo con un único proceso sobre la CPU. De todos modos, no es algo muy relevante dado que podría ser solventado con relativa facilidad.

- La CNN 2D resulta ser la más veloz de los esquemas de *deep learning*. También es algo perfectamente esperable, al ser la red más pequeña y con operaciones más simples. Por ejemplo la CNN 3D tiene la misma cantidad de capas, pero el hecho de realizar convoluciones 3D en vez de 2D –así como extraer un 25 % más de características– acaban implicando unos tiempos de ejecución mayores.

- La CNN 2D + RNN debe analizarse con cuidado. En primera instancia, puede parecer que es más costosa que la CNN 2D y la 3D, dado su mayor tiempo de entrenamiento. En cambio, el tiempo de prueba para esta red sí es mayor al que se obtiene con la CNN 2D, pero no con la CNN 3D. Lo que se está observando es que la incorporación de la rama espectral a la CNN 2D implica, como no podía ser de otro modo, tiempos de ejecución superiores a esta, pero no tanto como para alcanzar el sobrecoste de la CNN 3D.

  El entrenamiento de las redes es un proceso iterativo, pero la prueba no. Si se recuerda el Cuadro 6.1 en la página 36, la CNN 2D + RNN se entrena durante un 60 % de más épocas que la CNN 3D, justificando así que la primera tarde más en entrenarse a pesar de que el coste computacional de todas sus capas sea menor. Y esto último se corrobora por el hecho de que la CNN 2D + RNN sea más veloz en la etapa de prueba, que no es iterativa.

- En último lugar, y como es esperable, la CNN 2D residual supera en gran medida a todos los demás clasificadores en cuanto a tiempos de entrenamiento y de prueba. La gran profundidad que presenta se traduce en un gran número de operaciones convolucionales, y ello conlleva unos tiempos de ejecución muy superiores a los demás esquemas. Por supuesto, esta profundidad tiene la gran baza de conseguir las mejores precisiones de todas las redes consideradas, en caso de que se esté dispuesto a pagar el precio.

Finalmente, se va a proceder a analizar en qué medida se ven afectados los tiempos de cada clasificador al pasar de trabajar a nivel de píxel, a emplear muestras en base a superpíxeles:

- La etapa de entrenamiento no se ve afectada, por norma general, ante la inclusión de la segmentación. Como se concretó en el Apartado 6.2 en la página 37, el número absoluto de muestras de entrenamiento se mantiene constante en cada pareja de experimentos –uno con segmentación, y otro sin ella–, de modo que las condiciones de partida del clasificador sean equiparables, así que todo esto es esperable.

  Como mucho, puede destacarse una cierta ralentización del entrenamiento en el caso de SVM cuando se trabaja en base a superpíxeles, que se deberá a la necesidad de computar la media espectral de una gran variedad de píxeles para generar cada muestra. Por supuesto, este sobrecoste se traduce en la mayor riqueza de la información que caracteriza las muestras, lo cual permite a SVM incrementar las precisiones alcanzadas. En caso de tomar un único píxel–vector de cada superpíxel –por ejemplo, se podría coger el píxel central

del segmento–, seguramente no se notaría una ralentización del entrenamiento, pero las precisiones dejarían de ser superiores.

- Con todo esto, lo verdaderamente interesante se produce en la etapa de prueba de los esquemas. El emplear superpíxeles conlleva un cambio de escala en los tiempos de clasificación de los subconjuntos de muestras de prueba; concretamente, implica pasar de cientos de segundos, a valores de 1 o 2 segundos.

  Como se planteó anteriormente en el Apartado 3.3 en la página 10, siendo $S \times S = S^2$ el tamaño medio de píxeles/superpíxel, sería esperable observar una aceleración de esta etapa de prueba en el orden de S^2, ya que se pasará a realizar una sola predicción a nivel de superpíxel por cada S^2 predicciones a nivel de píxel. Esto es precisamente lo que se puede comprobar en este caso. Todos los clasificadores son capaces de experimentar aceleraciones en el orden de cientos, gracias a realizar predicciones en base a superpíxeles que contienen cientos de píxeles.

  Como última nota, cabe indicar que lo esperable en los tiempos de prueba es que se reduzcan a medida que el número de muestras de entrenamiento se incrementa. Esto es algo que se observa claramente en el caso de entrenar los esquemas de clasificación en base a superpíxeles, pero no en el caso de no segmentar las imágenes. Esto es una consecuencia de utilizar el mismo número de muestras de entrenamiento tanto si se trabaja con píxeles, como si se hace con superpíxeles, dentro de cada pareja de experimentos. Si se recuerda el Cuadro 5.1 en la página 30, el número de muestras totales en base a píxeles es tan elevado, que al final en todos los experimentos sin segmentación se acaban clasificando más del 99 % de muestras en la etapa de prueba.

  A pesar de todo ello, en el caso de emplear un 15 % de muestras en base a superpíxeles, el 80 % de las muestras se procesan durante la prueba del clasificador[2]. Este valor no se encuentra lejos del 99 % de muestras a procesar durante la prueba en el experimento a nivel de píxel correspondiente, y aún así se siguen observando aceleraciones en el orden de cientos para la etapa de prueba, así que las observaciones realizadas anteriormente se siguen sosteniendo a pesar de este matiz.

Como se explicó en el Apartado 5.2.2 en la página 32, también se tendrán en cuenta en las métricas de rendimiento los tiempos requeridos para segmentar y preprocesar las imágenes experimentales. Estos son cruciales para este trabajo, dado que determinarán el coste a pagar en caso de querer incorporar la etapa de segmentación al flujo de la clasificación HSI.

Entonces, el Cuadro 6.5 refleja estos costes, desglosados por cada imagen. El esquema y la cantidad de muestras a emplear en el experimento no influyen en estas dos métricas. Por lo tanto, los valores que se reflejan se tratan de unas medias de todos los experimentos con cada imagen.

Sorprendentemente, la incorporación de la etapa de segmentación antes del flujo de clasificación HSI no solo ha resultado mantener el tiempo de preprocesamiento de las imágenes, sino que incluso ha reducido el coste de esta fase, al ser todas las aceleraciones –*speedups*– mayores que 1.

A pesar de que el tiempo de segmentación supone hasta una decena de segundos, es importante tener presente que el total de muestras resultantes en la imagen se llega a reducir en múltiples órdenes de magnitud, tal y como reflejaba el Cuadro 5.1 en la página 30. Es decir, se ahorra tanto coste computacional al tener que identificar y registrar menos muestras en las imágenes, que se compensa el sobre-

[2] Cabe recordar que un 5 % de las muestras se reservan para el conjunto de validación.

Cuadro 6.5: Coste de incorporar la segmentación SLIC a cada una de las tres imágenes experimentales. El tiempo de preprocesamiento representa el coste total de leer la imagen, y de identificar y registrar las muestras disponibles en ella, además de segmentarla en caso de trabajar a nivel de superpíxel.

	Oitavén		**Ermidas**		**Eiras**	
	píxeles	superpíxeles	píxeles	superpíxeles	píxeles	superpíxeles
tiempo segmentación (s)	0.00	2.28	0.00	10.08	0.00	5.06
tiempo preprocesamiento (s)	28.10	10.43	44.80	37.70	23.50	18.37
speedup preprocesamiento	2.69x		1.19x		1.28x	

coste de incorporar la segmentación con SLIC, e incluso se reduce el coste total de preprocesar la imagen de cara a la posterior clasificación HSI.

7 | Discusión experimental

A modo de síntesis, se recogerán aquí los resultados experimentales más destacables que se han reflejado y comentado en el capítulo anterior, así como las posibles implicaciones que ello pueda tener en la evolución del campo de la clasificación HSI.

Los experimentos llevados a cabo han mostrado cómo los superpíxeles permiten reducir el número de primitivas a procesar en una imagen, gracias a capturar la redundancia de esta mediante el agrupamiento de píxeles similares. Esto permite alcanzar una importante aceleración en el proceso de clasificar una imagen con cualquier esquema de clasificación HSI, al realizarse una predicción por superpíxel, tal y como se sugirió en el Apartado 3.3 en la página 10, en lugar de tratar independientemente cada uno de sus miembros.

Además, tal y como se explicó también en el Apartado 3.1 en la página 5, los superpíxeles no deberían conducir a una pérdida de la información contenida en la imagen, sino que solo tratan de reducir la cantidad de información redundante a tratar. Esto también se ha podido verificar al comprobar que entrenar y probar los esquemas de clasificación HSI de *deep learning* sobre una imagen segmentada no incurre en una degradación de las precisiones de clasificación; es decir, que las muestras extraídas de las imágenes segmentadas son igual de ricas en información que las muestras obtenidas en base a píxeles.

Por ejemplo, en los experimentos de este trabajo se ha comprobado cómo los superpíxeles permiten pasar de clasificar una imagen con una CNN 2D en 1–1,5 minutos, a hacerlo en cuestión de 1 segundo. Es más, tampoco es necesario limitarse a una sencilla red neuronal para clasificación HSI, sino que también pasa a ser perfectamente asumible emplear complejos esquemas, como una CNN residual. Por ejemplo, la tratada en este trabajo ha pasado de estar clasificando las imágenes durante 3–8,5 minutos, a tardar menos de 2 segundos con cada una.

Como aspecto destacado, se puede señalar además que el coste de incorporar la etapa de segmentación puede llegar a ser nulo como tal, e incluso acelerar el preprocesamiento de las imágenes de cara al flujo de la clasificación HSI, al reducirse en múltiples órdenes de magnitud la cantidad de muestras a tratar. En este caso, las imágenes son segmentadas empleando una implementación personalizada de SLIC y paralelizada con OpenMP. Empleando un total de 16 hilos para esta etapa, se consigue que el coste de incorporarla sea tan bajo que al final el tiempo de preprocesamiento total de las imágenes resulta ser menor que antes, al reducirse de forma tan significativa el número total de muestras a considerar.

En la literatura de la clasificación HSI se han propuesto esquemas de *deep learning* más elaborados que CNNs puras, como incluir módulos recurrentes, o construir CNNs muy profundas gracias al paradigma del aprendizaje residual. Las mayores capacidades de estas redes a la hora de analizar la información a clasificar les permite alcanzar mejores precisiones. Sin embargo, es importante destacar que todas las comparativas de la literatura disponibles hasta el momento, que ponen unos tipos de redes neuronales frente a otras, han tratado siempre las imágenes a nivel de píxel. Por lo tanto, ahora se ha comprobado además cómo esta jerarquía de los esquemas en base a precisiones se sostiene también a la hora de trabajar con las imágenes en

base a superpíxeles, como consecuencia de que incorporar la segmentación no altere realmente las precisiones que se consiguen alcanzar.

En este trabajo se ha aplicado una segmentación por superpíxeles a dos CNN puras, una con filtros 2D –CNN 2D– y otra con filtros 3D –CNN 3D–, y a otras dos CNNs más avanzadas, incluyendo una un módulo recurrente –CNN 2D + RNN–, y fundamentándose otra en el aprendizaje residual –CNN 2D residual–. Los mejores resultados han sido obtenidos por la CNN 2D residual, independientemente de la imagen y de la cantidad de muestras de entrenamiento. El motivo posiblemente esté relacionado con su capacidad para extraer características más complejas y completas de la información a tratar, dada su gran profundidad en cuanto a cantidad de capas convolucionales. Esto último es factible gracias a la incorporación de "atajos" de capas de bajo nivel a capas de alto nivel, en la forma de bloques residuales, para solventar problemas como el deterioro del gradiente, o la degradación de las características extraídas [19].

Los siguientes mejores resultados en cuanto a precisión han sido obtenidos por la CNN 2D + RNN, que supera ligeramente a los resultados de las CNNs 2D y 3D. No son diferencias muy pronunciadas, estando generalmente por debajo del 2 % de incremento, pero siguen siendo superiores en todo caso. Este tipo de red es capaz de tomar partido de su módulo recurrente LSTM para identificar semejanzas espectrales entre muestras HSI que sean suministradas en diferentes instantes de tiempo, para tratar de emplear estos parecidos para realizar mejores predicciones en base decisiones anteriores [24].

Finalmente, la CNN 2D y la 3D obtienen las precisiones más bajas de los clasificadores basados en *deep learning*, pero siguen siendo de todos modos considerablemente superiores a SVM. Cabe destacar que se esperaba que la CNN 3D reportarse mejores precisiones, al tener capacidad para tratar información espacial como una CNN 2D, pero poder prestar a la vez más atención a las características espectrales de la imagen, al tratar sus bandas en subconjuntos, en lugar de hacerlo al mismo tiempo [18]. Sin embargo, ambas redes han reportado precisiones muy semejantes en todos los experimentos, así que no es posible determinar que ninguna sea mejor que la otra.

En cualquier caso, se debe destacar de nuevo que la incorporación de la segmentación a los esquemas de *deep learning* no ha supuesto ninguna penalización con respecto a las precisiones obtenidas en una clasificación tradicional realizada a nivel de píxeles.

La introducción de superpíxeles a esquemas de clasificación basados en *deep learning* tiene más implicaciones que simplemente reducir un coste computacional. Puede incluso conllevar que sea viable incorporar este tipo de clasificadores, incluso complejos –como las ResNets– a muchas aplicaciones para las que resultan actualmente prohibitivos, dados sus elevados costes de ejecución.

Por ejemplo, supóngase el caso de clasificar las secuencias de imágenes capturadas por un dron utilizando una red previamente entrenada. Si esta tarea ha de realizarla el propio dron en tiempo real, capturando, por ejemplo, una imagen cada minuto, y debiendo procesarla antes de capturar la siguiente, puede hacerse fácilmente inviable que el dron haga este procesamiento por sí mismo. En cambio, la incorporación de superpíxeles al esquema de clasificación puede hacer factible este proceso incluso con redes muy complejas, en búsqueda de las mejores precisiones posibles.

Finalmente, cabe destacar que los esquemas de *deep learning* de clasificación HSI con los que aquí se ha experimentado no se corresponden exactamente con los esquemas propuestos en sus respectivas publicaciones originales, sino que se han modificado ligeramente con objeto de integrarlos en un entorno consistente y válido para todos los clasificadores. De esta forma, se consigue que estos esquemas representen las ideas de *deep learning* más presentes en la actualidad en el campo de la

clasificación HSI, y que los resultados obtenidos puedan ser comparados en igualdad de condiciones. Además, este entorno unificado también facilitará la tarea de incorporar la clasificación a nivel de superpíxeles a los nuevos esquemas de *deep learning* que se propongan en el futuro, y compararlos con los previamente estudiados.

8 | Conclusiones y trabajo futuro

En este Trabajo de Fin de Grado, se ha estudiado la aplicación de segmentaciones en superpíxeles a diversos esquemas de clasificación de imágenes hiperespectrales basados en *deep learning*, con el objetivo de reducir el impacto computacional de estos costosos modelos. El procedimiento ha consistido en segmentar la imagen a tratar previamente a la clasificación, de modo que se trabaje en base a superpíxeles, en lugar de con píxeles individuales.

Dado que los superpíxeles agrupan conjuntos de píxeles semejantes, es posible realizar una sola predicción de categoría por superpíxel y asignársela a todos sus píxeles, en lugar de realizar una predicción por cada píxel de la imagen. De este modo, se reduce la redundancia de la información a procesar, con lo que se limita enormemente el coste de clasificar una imagen, al realizarse muchas menos predicciones. Por otra parte, las precisiones de clasificación no deberían verse perjudicadas, debido precisamente a que un superpíxel agrupa píxeles similares, que pertenecerán a la misma categoría tras la clasificación.

Con todo ello, las principales contribuciones de este trabajo son las siguientes:

1. Se ha comprobado experimentalmente que la incorporación de una segmentación en superpíxeles a los clasificadores permite reducir significativamente el coste de clasificar una imagen. Esto es de gran utilidad en los esquemas basados en *deep learning*, dado que las elevadas precisiones que estos clasificadores consiguen alcanzar son a cambio de unos consumos computacionales mucho más elevados que los métodos de aprendizaje automático tradicionales, tales como SVM.

2. Puesto que los superpíxeles reducen la redundancia en la imagen a procesar, no incurren en una pérdida de información durante la clasificación, lo cual se ha comprobado experimentalmente a través de la calidad de las precisiones que los clasificadores son capaces de alcanzar.

3. La etapa de segmentación en superpíxeles propuesta en este trabajo se basa en el algoritmo SLIC, que se ha optimizado y paralelizado con el objeto de limitar el sobrecoste computacional que pueda ocasionar frente al procesamiento de la imagen a nivel de píxel. El tiempo de ejecución de esta etapa se ve holgadamente compensado por la reducción del tiempo de clasificación de la imagen, dado que el número de elementos a procesar se reduce de forma muy significativa al trabajar con superpíxeles.

4. Este estudio ha considerado una gran variedad de esquemas basados en *deep learning* publicados en la bibliografía. Con ello, se cubren las técnicas y conceptos que hay detrás de los esquemas de clasificación HSI basados en *deep learning* que son populares actualmente en el campo. Consecuentemente, estas conclusiones son extrapolables a un gran número de clasificadores propuestos en la literatura por otros autores.

5. En concreto, se han considerado clasificadores que giran en torno al concepto de las CNNs, al ser el tipo de red más popular durante los últimos años en el ámbito de la clasificación HSI. Concretamente, se han empleando CNNs con filtros 2D y 3D, dado que procesan regiones con múltiples píxeles de la imagen –parches– para emplear tanto información espacial como espectral en la clasificación.

   Aparte de emplear CNNs puras, se han construido otros clasificadores más elaborados mediante la incorporación de otros paradigmas, como el uso de módulos recurrentes, o el uso del aprendizaje residual. Tal y como es de conocimiento común dentro del campo de la clasificación HSI, estos clasificadores más complejos permiten alcanzar clasificaciones de mejor calidad por norma general.

6. En lo mejor de nuestro conocimiento, este trabajo es una aportación original, en el sentido de que no se ha encontrado en la literatura ningún estudio similar que compare diferentes esquemas de clasificación HSI basados en *deep learning* aplicando técnicas de segmentación en superpíxeles.

Por consiguiente, se han podido cumplir plenamente los objetivos planteados para este Trabajo de Fin de Grado. Adicionalmente, puesto que los resultados de este trabajo son previsiblemente extrapolables a otros esquemas de *deep learning* de la literatura, se han abierto dos vías para:

- Acelerar muchos esquemas de clasificación HSI basados en *deep learning* publicados actualmente en la bibliografía, de cara a poder utilizarlos en aplicaciones de prácticamente tiempo real.
- Proponer esquemas más complejos basados en *deep learning* que pudieran ser directamente excesivamente costosos a nivel de píxel.

En cuanto a las posibles ampliaciones del trabajo realizado, también se han abierto diversas vías, tales como:

- Determinar hasta qué punto el variar el algoritmo de segmentación puede llegar a influir en el desempeño de los clasificadores al trabajar con superpíxeles. Por ejemplo, podría darse el caso de que algoritmos como CRS, ETPS, o SEEDS, generen segmentaciones más precisas que SLIC, incrementando la calidad de las clasificaciones[1], o que también generen segmentaciones de peor calidad. Por otra parte, muchos algoritmos de segmentación en superpíxeles permiten personalizar un factor de compacidad, al igual que SLIC, así que también sería de interés analizar el impacto de este valor.
- Para la fase experimental de este trabajo se han empleado imágenes de un conjunto de datos de cuencas de ríos gallegos. Sería de interés replicar los experimentos con otros conjuntos de datos de imágenes hiperespectrales de gran resolución, a modo de completitud.
- Podrían incorporarse al análisis otra variedad de herramientas del *deep learning* que se hayan empleando en el campo de la clasificación HSI, pero que no son tan populares actualmente, como las Redes Generativas Adversarias –Generative Adversarial Networks (GAN)– [70].

[1] Es relativamente inevitable que algunos de los superpíxeles resultantes tras una segmentación contengan unos pocos píxeles que debiesen pertenecer más bien a otro superpíxel, y que por lo tanto podrán ser categorizados erróneamente. Es algo que sucederá sobre todo en los límites entre diferentes objetos de la imagen, en función de cómo de bien consigan adaptarse los superpíxeles a ellos.

A | Manual técnico

A lo largo de este apéndice, se recogerán todas las instrucciones necesarias para preparar un entorno en GNU/Linux sobre el que poder ejecutar todo el código fuente involucrado en la fase experimental de este trabajo, de cara a facilitar la reproducibilidad de los resultados experimentales.

A.1. Adquisición del código experimental

Todo el código necesario para replicar los experimentos de este trabajo puede ser consultado en forma de un repositorio de GitHub (https://github.com/alvrogd/DeepSuperCNNs). A través de este mismo enlace, es posible descargar directamente todos los ficheros a la máquina local para comenzar a replicar el entorno experimental.

A.2. Instalación de prerrequisitos

Para facilitar lo máximo posible el replicar exactamente el entorno sobre el que se llevó a cabo la fase experimental, se ha preparado un contenedor de Docker (https://www.docker.com/) para gestionar todo el software involucrado. Gracias a esta herramienta, será extremadamente sencillo ejecutar el mismo software y en las mismas versiones sobre cualquier arquitectura `x86_64` moderna, para uniformizar la preparación del entorno sobre cualquier máquina, y evitar anomalías por nuevas versiones del software.

Entonces, es imprescindible instalar en primer lugar Docker Engine para poder generar el contenedor que replique todo el entorno de software. Para ello, se recomienda encarecidamente consultar las últimas instrucciones disponibles en la documentación de Docker (https://docs.docker.com/engine/), para adquirir la versión más reciente de esta plataforma.

De forma opcional, el código experimental es capaz de tomar partido de cualquier tarjeta gráfica con suporte para CUDA (https://developer.nvidia.com/cuda-zone) que se encuentre conectada al equipo. En el caso de los contenedores de Docker, es necesario instalar un componente adicional a Docker Engine para que un contenedor pueda acceder a una tarjeta gráfica del equipo. Se recomienda también consultar la última versión de la documentación de instalación de NVIDIA Container Toolkit (https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/install-guide.html#docker) para poder utilizar este componente complementario.

A.3. Construcción del contenedor de software

Una vez se ha verificado que la instalación de la plataforma Docker ha sido exitosa, se puede pasar a construir el contenedor que albergará todas las dependencias

de software del código experimental.

Dentro del directorio raíz del proyecto, se incluye un directorio `dockerfiles` en el que se disponen todos los ficheros involucrados para generar y ejecutar el contenedor de Docker. Para construirlo, simplemente es necesario ejecutar el script `build_container.sh`:

```
./build_container.sh
```

Él mismo se encargará de recoger y combinar todas las dependencias software especificadas en el fichero `Dockerfile`, respetando las versiones indicadas. Cuando termine su ejecución, se habrá construido satisfactoriamente el contenedor de software para ejecutar el código experimental.

El script no debería reportar ningún tipo de error durante la construcción del contenedor. En todo caso, conviene asegurarse de que la plataforma de Docker está funcionando correctamente en la máquina, que esta se basa en la arquitectura `x86_64`, y que dispone de suficiente espacio de almacenamiento. Si surge cualquier error extraño, se recomienda consultarlo en la web, o generar una incidencia en el repositorio del proyecto (https://github.com/alvrogd/DeepSuperCNNs), que será atendida en la mayor brevedad posible.

A.4. Ejecución del contenedor de software

Antes de lanzar el contenedor, es necesario realizar una pequeña modificación en el script `launch_container.sh`. La línea 8 del mismo es:

```
    -v /home/alvrogd/TFG/DeepSuperCNNs:/opt/DeepSuperCNNs \
```

Su propósito es indicar a Docker cómo hacer que el código experimental se encuentre accesible para el contenedor de software. Por ello, es necesario sustituir el fragmento `/home/alvrogd/TFG/DeepSuperCNNs` por la ruta absoluta hasta el directorio raíz del proyecto tras descargarlo.

Una vez hecho este cambio, el contenedor se puede lanzar a ejecución a través de ese mismo script:

```
./launch_container.sh bash
```

Se encargará de indicar a Docker que busque el contenedor previamente construido, y que lo lance a ejecución, perfectamente preparado para ejecutar cualquier código. Concretamente, se abrirá una terminal dentro del contenedor con la que se podrá interactuar desde la máquina local:

```
alvrogd@pop-os:~/TFG/DeepSuperCNNs$ ./dockerfiles/launch_container.sh bash
root@e0ca4e1f517b:/opt/DeepSuperCNNs#
```

En caso de haber indicado correctamente la ruta absoluta hasta el proyecto en la máquina local, este será accesible desde el contenedor de Docker a través del directorio `/opt/DeepSuperCNNs`. Es posible salir del contenedor mediante una sencilla instrucción `exit`.

A.5. Adquisición de los conjuntos de datos experimentales

De partida, no se incluye ninguna imagen multiespectral ni hiperespectral con los ficheros del proyecto. Por lo tanto, no será posible lanzar ningún experimento hasta haber adquirido alguna de ellas.

Por una parte, el Grupo de Inteligencia Computacional de la Universidad del País Vasco pone a disposición en su web (http://www.ehu.eus/ccwintco/index.php/Hyperspectral_Remote_Sensing_Scenes) una multitud de imágenes hiperespectrales extremadamente populares dentro de la comunidad HSI. Por otra parte, las imágenes multiespectrales de las cuencas de ríos gallegos que se han empleado en este trabajo no son públicamente accesibles. En caso de estar interesado en ellas, será necesario contactar al propietario del repositorio del proyecto (https://github.com/alvrogd/DeepSuperCNNs).

El código experimental recurre al fichero `datasets.py` para saber cómo acceder a las imágenes en el almacenamiento local, y cómo tratarlas. Dentro de este fichero, existe una estructura a partir de la línea 21 que especifica al código qué imágenes se encuentran disponibles. Esta estructura contiene una entrada por cada imagen, cuyo formato dependerá a su vez del formato de almacenamiento de la imagen:

- Una entrada correspondiente a una imagen hiperespectral tendrá una forma tal que:

```
"salinas": {
    "format": "mat",
    "image": [
        "Salinas_corrected.mat",
        "salinas_corrected",
    ],
    "gt": [
        "Salinas_gt.mat",
        "salinas_gt",
    ],
    "classes": [
        "Brocoli_green_weeds_1",
        "Brocoli_green_weeds_2",
        "Fallow",
        "Fallow_rough_plow",
        "Fallow_smooth",
        "Stubble",
        "Celery",
        "Grapes_untrained",
        "Soil_vinyard_develop",
        "Corn_senesced_green_weeds",
        "Lettuce_romaine_4wk",
        "Lettuce_romaine_5wk",
        "Lettuce_romaine_6wk",
        "Lettuce_romaine_7wk",
        "Vinyard_untrained",
        "Vinyard_vertical_trellis",
    ],
    "segmentation_params": {
        "segment_size": 50,
        "m": 25,
        "error_thr": 300,
        "max_iterations": 10,
    }
},
```

La primera línea especifica el nombre con el cual identificar la imagen. Será lo que se deba indicar al código para que utilice una imagen u otra.

La segunda línea de la entrada indica el formato de la imagen; en este caso, todas las imágenes hiperespectrales accesibles en la web especificada anteriormente se tratan de ficheros `.mat`.

La información requerida para cada imagen se divide en dos ficheros: los datos de la imagen en sí, y la información de referencia para cada píxel de la misma. El campo que comienza en la línea 3 contiene dos entradas que conciernen a la información de la imagen; la primera indica el nombre del fichero con los datos de la imagen, y la segunda indica un nombre dependiente de la imagen bajo el cual son accesibles los datos[1]. El campo de la información de referencia comienza en la línea 7, y sus dos entradas son análogas a las de la información de la imagen en sí.

Continuando, en la línea 11 se comienza un nuevo campo que indicará los nombres de todas y cada una de las categorías identificables en la información de referencia de la imagen. El orden de declaración debe respetar la numeración con la que se haya compuesto la información de referencia; en este caso, los píxeles marcados con la etiqueta 1 deben ser las muestras de `Brocoli_green_weeds_1`, los marcados con la etiqueta 2 deben ser las muestras de `Brocoli_green_weeds_2`…

En último lugar, el campo que comienza en la línea 29 permite personalizar los parámetros con los que ejecutar SLIC, en caso de querer segmentar la imagen de cara al flujo de clasificación HSI. La primera entrada indica el tamaño medio de superpíxel ($S \times S$), la segunda indica el factor de regularidad o compacidad (m), la tercera indica el umbral que determina cuándo converge el algoritmo, y la cuarta establece el máximo de iteraciones permitidas, como alternativa al umbral.

- Una entrada correspondiente a las imágenes multiespectrales experimentales presenta una forma muy similar a la anterior:

```
1  "z1": {
2      "format": "gtiff",
3      "image": [
4          "z1_blue.tif",
5          "z1_green.tif",
6          "z1_nir.tif",
7          "z1_red.tif",
8          "z1_red_edge.tif",
9      ],
10     "gt": [
11         "gtv3.raw",
12     ],
13     "classes": [
14         "auga",
15         "carballo",
16         "tellados",
17         "prado",
18         "asfalto",
19         "terra",
20         "pedra",
21         "cemento",
```

[1] Tras leer un fichero `.mat`, este contendrá múltiples entradas identificadas por un nombre, en un formato clave–valor. Los datos de la imagen se encuentran dentro de uno de estos campos, cuyo nombre varía de una imagen hiperespectral a otra. Para simplificar la ejecución del código, en `datasets.py` ya se han preconfigurado varias imágenes hiperespectrales, de modo que no sea necesario lidiar con este campo.

```
22            "vexetacion_ribeira",
23            "eucalipto",
24            "pineiros",
25        ],
26        "segmentation_params": {
27            "segment_size": 800,
28            "m": 40,
29            "error_thr": 200,
30            "max_iterations": 10,
31        }
32    },
```

Las únicas diferencias respecto a las imágenes hiperpespectrales serían relativas a cómo acceder a los ficheros de la imagen.

Ahora, estas imágenes multiespectrales ya no se almacenan en formato `.mat`, sino que se tratan de imágenes GeoTIFF. Cada banda de la imagen se encuentra en un fichero diferente, y por ello se especifican tantas entradas como bandas en el campo que comienza en la línea 3. En cuanto a la información de referencia, esta se almacena en un formato binario, así que simplemente es necesario indicar el nombre del fichero bajo el que se puede encontrar.

Para simplificar la ejecución del código experimental, el fichero `datasets.py` ya incluye de partida la configuración necesaria para leer y tratar:

- **Imágenes hiperespectrales:** Indian Pines, Pavia Centre, Pavia University, Salinas.
- **Imágenes multiespectrales:** Río Oitavén (*z1*), Riachuelo Ermidas (*z2*), Embalse Eiras (*embalse*).

Ahora bien, la web de la cual se pueden adquirir las imágenes hiperespectrales (http://www.ehu.eus/ccwintco/index.php/Hyperspectral_Remote_Sensing_Scenes) contiene algunos ejemplos más. En caso de querer emplearlos, será necesario añadir al fichero `datasets.py` una nueva entrada por cada nueva imagen a probar, configurándolas según el formato explicado anteriormente.

En base a la información especificada en `datasets.py`, el código tratará de leer las imágenes dentro del directorio `datasets` disponible en la raíz del proyecto. Dentro de este, será necesario crear un directorio por cada imagen que se haya configurado, cuyo nombre deberá corresponderse con el identificador dado para la imagen en `datasets.py`. Dentro del directorio, es necesario situar todos los ficheros de la imagen –tanto sus datos como la información de referencia– respetando los nombres especificados anteriormente.

B | Manual de usuario

A lo largo de este apéndice, se indicarán todas las instrucciones a seguir para poder replicar cualquier experimento de este trabajo.

Es necesario haber preparado y conseguido lanzar correctamente el contenedor de Docker que dará acceso al código experimental, junto con todas las dependencias de software asociadas, así como haber adquirido alguna imagen multi o hiperespectral que emplear. En caso de no haber completado estas tareas, se recomienda consultar el Apéndice A antes de proseguir.

B.1. Cómo repetir un experimento

De entre todos los ficheros de código del proyecto, el único con el que es necesario interactuar para repetir un experimento es `run_network.py`. Este script de Python se encarga él solo de:

1. Leer una cierta imagen de entrada, segmentándola en caso de habérselo pedido, y preparando todas las muestras que haya en ella para el flujo de clasificación HSI.
2. Entrenar un cierto clasificador sobre la anterior imagen.
3. Probar el desempeño del clasificador una vez entrenado.

Por supuesto, este script admite una gran variedad de opciones para personalizar el experimento a ejecutar. Estas serían:

- **Parámetros sobre la imagen:**
  1. `--dataset <valor>`: indica la imagen con la que se trabajará durante el experimento.

     Debe corresponderse con el identificador de alguna de las imágenes especificadas en `datasets.py`.
  2. `--segment_dataset | --no_segment_dataset:` si se especifica el primer valor, se segmenta la imagen empleando SLIC y los parámetros especificados para ella en su entrada en `datasets.py`; en caso contrario, se trabaja con la imagen a nivel de píxel.
  3. `--train_ratio <valor>`: número decimal en el rango $(0, 1)$ que indica el porcentaje de muestras totales de la imagen a dedicar al subconjunto de entrenamiento del clasificador; por ejemplo, el valor `0.15` se corresponde con un 15 % de las muestras totales.

     El ratio de prueba se determina en base a este, y al valor dado como ratio de validación.

4. `--val_ratio <valor>`: número decimal en el rango $(0, 1)$ que indica el porcentaje de muestras totales de la imagen a dedicar al subconjunto de validación del clasificador[1]; por ejemplo, el valor `0.05` se corresponde con un 5 % de las muestras totales.
   El ratio de prueba se determina en base a este, y al valor dado como ratio de entrenamiento.
5. `--use_patches | --no_use_patches:` si se especifica el primer valor, cada muestra consistirá en un parche sobre la imagen de entrada; en caso contrario, cada muestra consistirá en un único píxel–vector.
   Los esquemas basados en *deep learning* siempre toman como entrada un parche; esto sucede incluso con la CNN 2D + RNN, ya que ella misma extrae el píxel–vector central del parche que se utiliza en la rama RNN. Por otra parte, SVM trabaja siempre con píxel–vectores.
6. `--patch_size <valor>`: número entero que indica la vecindad N a la hora de generar las muestras.
   Por ejemplo, supóngase un valor de `25`. En el caso de tomar parches como muestras, cada uno abarcará 25×25 píxeles. En caso de tomar píxel–vectores como muestras, cada uno consistirá en la media espectral de una región de 25×25 píxeles.
7. `--apply_pca | --no_apply_pca:` si se especifica el primer valor, se reduce la dimensionalidad espectral de la imagen[2] –número de bandas– empleando el método PCA; en caso contrario, no se altera.
8. `--new_dimensionality <valor>`: número entero que indica la cantidad de bandas a conservar en caso de reducir la dimensionalidad espectral de la imagen.

- **Parámetros sobre el clasificador:**

1. `--network <valor>`: indica el clasificador con el que se trabajará durante el experimento; el valor indicado debe ser uno de los siguientes:
   - `SVC`: SVM de scikit–learn.
   - `Accion`: CNN 2D.
   - `Chen`: CNN 3D.
   - `Song`: CNN 2D residual.
   - `Xu`: CNN 2D + RNN.
2. `--joint | --no_joint:` debe especificarse el segundo valor con el clasificador CNN 2D + RNN, y el primer valor con los demás; es indiferente para SVM.
3. `--epochs <valor>`: número entero que indica la cantidad de épocas durante las que entrenar un clasificador basado en *deep learning*; es irrelevante para SVM.
4. `--batch_size <valor>`: número entero que indica la cantidad de muestras a suministrar en cada lote.
   En los clasificadores basados en *deep learning*, las muestras se suministran en lotes tanto en el entrenamiento como en la prueba. En el caso de SVM, tan solo se suministran en lotes durante la prueba[3].

[1] Este conjunto no se emplea con SVM, ya que el proceso de validación cruzada se realiza exclusivamente con los datos del subconjunto de entrenamiento.

[2] Es algo de especial utilidad con las imágenes hiperespectrales, dado que al contener cientos de bandas, algunas redes pueden llegar a requerir cantidades gigantes de memoria para ejecutarse.

[3] El procedimiento de hallar un clasificador SVM consiste en resolver un problema de optimización matemática. Por lo tanto, es necesario trabajar al mismo tiempo con toda la información de entrenamiento para tener presentes todas las variables y restricciones del problema.

5. `--num_workers <valor>`: número entero que indica la cantidad de procesos auxiliares que se permiten lanzar en paralelo; sus implicaciones dependen del clasificador:
   - Para SVM, determina el número de procesos a emplear para acelerar el proceso de entrenamiento.
   - Para los clasificadores basados en *deep learning*, determina cuántos procesos preparan y suministran muestras durante el entrenamiento y prueba a medida que se soliciten.
     Dada la rapidez de las tarjetas gráficas para entrenar redes neuronales, se recomienda establecer este valor cercano a `4` para mantener la GPU ocupada en todo momento. En caso de emplear la CPU para ejecutar las redes neuronales, PyTorch tratará de emplear múltiples hilos para acelerar la ejecución, así que el valor óptimo de procesos auxiliares dependerá en gran medida de las especificaciones concretas de la máquina; se recomienda probar diferentes valores hasta encontrar aquel que reporte el menor coste por época de entrenamiento.
6. `--device <valor>`: identificador que especifica el dispositivo a emplear para ejecutar los esquemas basados en *deep learning*:
   - Para emplear la CPU, se debe indicar `cpu` como valor.
   - En caso de disponer de una tarjeta gráfica con soporte para CUDA y querer emplearla, se debe indicar el número entero que la identifique dentro del sistema. Este se puede hallar fácilmente con el comando `nvidia-smi`. Por ejemplo, el siguiente sistema dispondría de dos GPUs NVIDIA Tesla V100S, identificadas por los números `0` y `1`:

```
root@d1fef36062a7:/opt/DeepSuperCNNs# nvidia-smi
Tue Jun 22 10:29:01 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 455.23.05    Driver Version: 455.23.05    CUDA Version: 11.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla V100S-PCI...  On   | 00000000:3B:00.0 Off |                    0 |
| N/A   26C    P0    23W / 250W |      0MiB / 32510MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   1  Tesla V100S-PCI...  On   | 00000000:D8:00.0 Off |                    0 |
| N/A   28C    P0    25W / 250W |      0MiB / 32510MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
```

   Por defecto, se emplea la primera tarjeta gráfica conectada al equipo o, en caso de no haber ninguna, la CPU.

Estos serían todos los argumentos que el script principal podría recibir e interpretar.

Para proceder a ejecutar el código experimental, se debe cargar en primer lugar el entorno de Python con todos sus paquetes, mediante el comando:

```
source /opt/conda/bin/activate
```

Y ahora ya se podría lanzar a ejecución el script `run_network.py` con todas las opciones deseadas. Por ejemplo:

```
python run_network.py \
    --dataset "z1" \
```

```
--segment_dataset \
--train_ratio 0.15 \
--val_ratio 0.05 \
--use_patches \
--patch_size 25 \
--network "Accion" \
--joint \
--epochs 100 \
--batch_size 128 \
--num_workers 4
```

A medida que progrese el experimento, se irán mostrando por pantalla tanto su avance, como las diferentes métricas experimentales que se vayan recogiendo.

En último lugar, cabe destacar que la implementación de SLIC paralelizada en OpenMP escoge por sí misma el número óptimo de hilos a emplear en función de las especificaciones de la máquina sobre la que se ejecute. Aún así, si se desea emplear un número concreto de hilos, es posible limitar el número de colaboradores si, antes de lanzar el script principal, se configura la siguiente variable de entorno:

```
export OMP_NUM_THREADS=16
```

Por ejemplo, el anterior comando limitaría el uso de hilos a un máximo de `16`. De todos modos, es necesario tener presente que esta opción afecta a absolutamente todo código de OpenMP que se ejecute de aquí en adelante, por lo que podría llegar a limitar también el número de hilos utilizados por PyTorch. Esto puede ser de relevancia, por ejemplo, en caso de querer entrenar las redes neuronales con la CPU.

B.2. Problemas de ejecución comunes

A continuación, se recogen algunos problemas comunes que pueden surgir durante la ejecución del código experimental:

- Con SVM es necesario especificar el argumento `--no_use_patches` dado que es un clasificador que trabaja a nivel de píxel–vector. Los esquemas basados en *deep learning* trabajan con parches, por lo que debe especificarse el argumento `--use_patches`.

  Si no se cumplen estas condiciones, el código reportará una excepción en tiempo de ejecución sobre que la dimensionalidad de las muestras extraídas de la imagen no es la esperada:

  ```
  Traceback (most recent call last):
    File "/opt/DeepSuperCNNs/run_network.py", line 139, in <module>
      networks.train(network, data_loader, dataset, hyperparams)
    File "/opt/DeepSuperCNNs/networks.py", line 1311, in train
      output = network(samples)
    File "/opt/conda/lib/python3.8/site-packages/torch/nn/modules/ (...)
      result = self.forward(*input, **kwargs)
    File "/opt/DeepSuperCNNs/networks.py", line 129, in forward
      tensor = tensor.permute(0, 3, 1, 2)
  RuntimeError: number of dims don't match in permute
  ```

- El único clasificador de *deep learning* que funciona con el argumento `--no_joint` es la CNN 2D + RNN. Para las demás redes neuronales se debe especificar el argumento `--joint`.

Si no se cumplen estas condiciones, el código podrá reportar una excepción en tiempo de ejecución:

```
Traceback (most recent call last):
  File "/opt/DeepSuperCNNs/run_network.py", line 139, in <module>
    networks.train(network, data_loader, dataset, hyperparams)
  File "/opt/DeepSuperCNNs/networks.py", line 1316, in train
    output_spectral, output_spatial, output_joint = network(samples)
ValueError: too many values to unpack (expected 3)
```

- La imagen a emplear durante el experimento debe contener una entrada correctamente especificada en el fichero `datasets.py`, tal y como se explica en el Apéndice A.5 en la página 53.

  En caso de que no existe una entrada para ella, el código reportará una excepción en tiempo de ejecución sobre que no existe tal imagen:

```
Traceback (most recent call last):
  File "/opt/DeepSuperCNNs/run_network.py", line 90, in <module>
    dataset = datasets.Dataset(
  File "/opt/DeepSuperCNNs/datasets.py", line 335, in __init__
    self.image = self.load_image(name)
  File "/opt/DeepSuperCNNs/datasets.py", line 452, in load_image
    if AVAIL_DATASETS[name]["format"] == "mat":
KeyError: 'z3'
```

- Los nombres de los ficheros que componen la imagen a emplear durante el experimento deben corresponderse con lo especificado en el fichero `datasets.py`.

  En caso de que un fichero no pueda ser encontrado, el código reportará una excepción en tiempo de ejecución sobre que no se puede leer algún fichero de la imagen:

```
Traceback (most recent call last):
  File "/opt/DeepSuperCNNs/run_network.py", line 90, in <module>
    dataset = datasets.Dataset(
  File "/opt/DeepSuperCNNs/datasets.py", line 338, in __init__
    self.gt    = self.load_gt(name)
  File "/opt/DeepSuperCNNs/datasets.py", line 527, in load_gt
    with open(f"datasets/{name}/{AVAIL_DATASETS ... ", "rb") as input:
FileNotFoundError: [Errno 2] No such file or directory: 'datasets/z1/gtv3.raw'
```

- El número de trabajadores –parámetro `num_workers`– tiene un impacto directo en la cantidad de memoria RAM requerida, ya que cada proceso irá preparando en su memoria muestras que ir suministrando más adelante a la CPU o GPU, a medida que se necesiten.

  Por lo tanto, conviene mantener este parámetro dentro de unos valores comprensibles para evitar alcanzar situaciones de escasez de RAM en el equipo, que pueden degradar el rendimiento del experimento, o incluso causar que se congele o cierre.

- El anterior parámetro también puede dar lugar a otro problema, causado en este caso por el contenedor de Docker. Los diferentes trabajadores utilizan la memoria RAM mapeada en `/dev/shm` para comunicación, y Docker asigna un tamaño por defecto de 64 MB. Este puede ser insuficiente, sobre todo al emplear más de 4 trabajadores. Por lo tanto, en caso de que algún trabajador reporte durante la ejecución un error tal que:

```
RuntimeError: DataLoader worker (pid 15689) is killed by signal: Bus error. \
    It is possible that dataloader's workers are out of shared memory. \
    Please try to raise your shared memory limit.
```

Será necesario incrementar el tamaño de dicha memoria. Para ello, es posible indicar a Docker el tamaño deseado en MB mediante la incorporación del siguiente argumento al script `launch_container.sh`:

```
--shm-size=1024m \
```

- Algunos de los clasificadores basados en *deep learning* adaptan sus parámetros a la imagen a procesar. Si se emplea una imagen hiperespectral con cientos de bandas, la cantidad de parámetros que pueden resultar en algunos clasificadores puede ser tal que se agota al completo la memoria disponible, reportándose un error como el del punto anterior.

  Ante estos casos, será necesario reducir la dimensionalidad espectral de las imágenes antes de proceder a la clasificación, para hacerlas tratables.

En caso de que surja cualquier otro tipo de error durante la ejecución de un experimento, se recomienda generar una incidencia en el repositorio del proyecto (https://github.com/alvrogd/DeepSuperCNNs), que será atendida en la mayor brevedad posible.

Bibliografía

[1] L. Zhu, J. Suomalainen, J. Liu, J. Hyyppä, H. Kaartinen y H. Haggren, «A Review: Remote Sensing Sensors,» en *Multi-purposeful Application of Geospatial Data*, R. B. Rustamov, S. Hasanova y M. H. Zeynalova, eds., Rijeka: IntechOpen, 2018, cap. 2.

[2] D. Chutia, D. K. Bhattacharyya, K. K. Sarma, R. Kalita y S. Sudhakar, «Hyperspectral Remote Sensing Classifications: A Perspective Survey,» *Transactions in GIS*, vol. 20, n.º 4, págs. 463-490, 2016.

[3] F. Melgani y L. Bruzzone, «Classification of hyperspectral remote sensing images with support vector machines,» *IEEE Transactions on Geoscience and Remote Sensing*, vol. 42, n.º 8, págs. 1778-1790, 2004.

[4] L. Zhang, L. Zhang y B. Du, «Deep Learning for Remote Sensing Data: A Technical Tutorial on the State of the Art,» *IEEE Geoscience and Remote Sensing Magazine*, vol. 4, n.º 2, págs. 22-40, 2016.

[5] M. Paoletti, J. Haut, J. Plaza y A. Plaza, «Deep learning classifiers for hyperspectral imaging: A review,» *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 158, págs. 279-317, 2019, ISSN: 0924-2716.

[6] S. Li, W. Song, L. Fang, Y. Chen, P. Ghamisi y J. A. Benediktsson, «Deep Learning for Hyperspectral Image Classification: An Overview,» *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, n.º 9, págs. 6690-6709, 2019.

[7] N. Audebert, B. Le Saux y S. Lefevre, «Deep Learning for Classification of Hyperspectral Data: A Comparative Review,» *IEEE Geoscience and Remote Sensing Magazine*, vol. 7, n.º 2, págs. 159-173, 2019.

[8] S. Liu, R. S. W. Chu, X. Wang y W. Luk, «Optimizing CNN-Based Hyperspectral Image Classification on FPGAs,» en *Applied Reconfigurable Computing*, C. Hochberger, B. Nelson, A. Koch, R. Woods y P. Diniz, eds., Cham: Springer International Publishing, 2019, págs. 17-31, ISBN: 978-3-030-17227-5.

[9] C. Shi y C.-M. Pun, «Superpixel-based 3D deep neural networks for hyperspectral image classification,» *Pattern Recognition*, vol. 74, págs. 600-616, 2018, ISSN: 0031-3203.

[10] Ren y Malik, «Learning a classification model for segmentation,» en *Proceedings Ninth IEEE International Conference on Computer Vision*, 2003, 10-17 vol.1.

[11] C. Shi y C.-M. Pun, «Multiscale Superpixel-Based Hyperspectral Image Classification Using Recurrent Neural Networks With Stacked Autoencoders,» *IEEE Transactions on Multimedia*, vol. 22, n.º 2, págs. 487-501, 2020.

[12] Y. Bengio, «Learning Deep Architectures for AI,» *Found. Trends Mach. Learn.*, vol. 2, n.º 1, págs. 1-127, ene. de 2009, ISSN: 1935-8237.

[13] Y. Lecun, L. Bottou, Y. Bengio y P. Haffner, «Gradient-based learning applied to document recognition,» *Proceedings of the IEEE*, vol. 86, n.º 11, págs. 2278-2324, 1998.

[14] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai y T. Chen, «Recent advances in convolutional neural networks,» *Pattern Recognition*, vol. 77, págs. 354-377, 2018, ISSN: 0031-3203.

[15] W. Hu, Y. Huang, L. Wei, F. Zhang y H. Li, «Deep Convolutional Neural Networks for Hyperspectral Image Classification,» *Journal of Sensors*, vol. 2015, pág. 258 619, jul. de 2015.

[16] Y. Chen, H. Jiang, C. Li, X. Jia y P. Ghamisi, «Deep Feature Extraction and Classification of Hyperspectral Images Based on Convolutional Neural Networks,» *IEEE Transactions on Geoscience and Remote Sensing*, vol. 54, n.º 10, págs. 6232-6251, 2016, cited By 1031.

[17] A. Ben Hamida, A. Benoit, P. Lambert y C. Ben Amar, «3-D Deep Learning Approach for Remote Sensing Image Classification,» *IEEE Transactions on Geoscience and Remote Sensing*, vol. 56, n.º 8, págs. 4420-4434, 2018.

[18] Y. Li, H. Zhang y Q. Shen, «Spectral–Spatial Classification of Hyperspectral Imagery with 3D Convolutional Neural Network,» *Remote Sensing*, vol. 9, n.º 1, 2017, ISSN: 2072-4292.

[19] K. He, X. Zhang, S. Ren y J. Sun, «Deep Residual Learning for Image Recognition,» en *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, págs. 770-778.

[20] R. K. Srivastava, K. Greff y J. Schmidhuber, «Training Very Deep Networks,» en *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, ép. NIPS'15, Montreal, Canada: MIT Press, 2015, págs. 2377-2385.

[21] W. Song, S. Li, L. Fang y T. Lu, «Hyperspectral Image Classification With Deep Feature Fusion Network,» *IEEE Transactions on Geoscience and Remote Sensing*, vol. 56, n.º 6, págs. 3173-3184, 2018.

[22] M. E. Paoletti, J. M. Haut, R. Fernandez-Beltran, J. Plaza, A. J. Plaza y F. Pla, «Deep Pyramidal Residual Networks for Spectral–Spatial Hyperspectral Image Classification,» *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, n.º 2, págs. 740-754, 2019.

[23] R. J. Williams y D. Zipser, «A Learning Algorithm for Continually Running Fully Recurrent Neural Networks,» *Neural Computation*, vol. 1, n.º 2, págs. 270-280, 1989.

[24] Y. Xu, L. Zhang, B. Du y F. Zhang, «Spectral–Spatial Unified Networks for Hyperspectral Image Classification,» *IEEE Transactions on Geoscience and Remote Sensing*, vol. 56, n.º 10, págs. 5893-5909, 2018.

[25] X. Mei, E. Pan, Y. Ma, X. Dai, J. Huang, F. Fan, Q. Du, H. Zheng y J. Ma, «Spectral-Spatial Attention Networks for Hyperspectral Image Classification,» *Remote Sensing*, vol. 11, n.º 8, 2019, ISSN: 2072-4292.

[26] D. Stutz, A. Hermans y B. Leibe, «Superpixels: An evaluation of the state-of-the-art,» *Computer Vision and Image Understanding*, vol. 166, págs. 1-27, 2018, ISSN: 1077-3142.

[27] P. G. Bascoy, A. S. Garea, D. B. Heras, F. Argüello y Á. Ordoñez, «Texture-based analysis of hydrographical basins with multispectral imagery,» en *SPIE Remote Sensing 2019, Remote Sensing for Agriculture, Ecosystems, and Hydrology XXI*, 2019, ISBN: 978-1-5106-3001-7.

[28] S. R. Blanco, D. B. Heras y F. Argüello, «Texture Extraction Techniques for the Classification of Vegetation Species in Hyperspectral Imagery: Bag of Words Approach Based on Superpixels,» *Remote Sensing*, vol. 12, n.º 16, 2020, ISSN: 2072-4292.

[29] Á. Acción, F. Argüello y D. B. Heras, «Dual-Window Superpixel Data Augmentation for Hyperspectral Image Classification,» *Applied Sciences*, vol. 10, n.º 24, 2020, ISSN: 2076-3417.

[30] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua y S. Süsstrunk, «SLIC Superpixels Compared to State-of-the-Art Superpixel Methods,» *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, n.º 11, págs. 2274-2282, 2012.

[31] C. Conrad, M. Mertz y R. Mester, «Contour-Relaxed Superpixels,» en *Energy Minimization Methods in Computer Vision and Pattern Recognition*, A. Heyden, F. Kahl, C. Olsson, M. Oskarsson y X.-C. Tai, eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, págs. 280-293, ISBN: 978-3-642-40395-8.

[32] J. Yao, M. Boben, S. Fidler y R. Urtasun, «Real-Time Coarse-to-Fine Topologically Preserving Segmentation,» en *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, jun. de 2015.

[33] M. Van den Bergh, X. Boix, G. Roig, B. de Capitani y L. Van Gool, «SEEDS: Superpixels Extracted via Energy-Driven Sampling,» en *Computer Vision – ECCV 2012*, A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato y C. Schmid, eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, págs. 13-26, ISBN: 978-3-642-33786-4.

[34] D. Clausi, «K-means Iterative Fisher (KIF) unsupervised clustering algorithm applied to image texture segmentation,» *Pattern Recognition*, vol. 35, n.º 9, págs. 1959-1972, 2002, ISSN: 0031-3203.

[35] S. Abe, *Support vector machines for pattern classification*. Springer London Ltd., 2012.

[36] C. Cortes y V. Vapnik, «Support-vector networks,» *Machine Learning*, vol. 20, n.º 3, págs. 273-297, sep. de 1995.

[37] J. G. Díaz, *Apuntes de Modelos y Técnicas de Optimización*, mar. de 2021.

[38] C.-W. Hsu y C.-J. Lin, «A comparison of methods for multiclass support vector machines,» *IEEE Transactions on Neural Networks*, vol. 13, n.º 2, págs. 415-425, 2002.

[39] V. Powell, *Image Kernels explained visually*, En línea; consultado el 11 de junio de 2021, 2021. dirección: https://setosa.io/ev/image-kernels/.

[40] S. U. Computer Science Deparment, *CS231n Convolutional Neural Networks for Visual Recognition*, Consultado el 11 de junio de 2021. dirección: https://cs231n.github.io/convolutional-networks/.

[41] V. Nair y G. E. Hinton, «Rectified Linear Units Improve Restricted Boltzmann Machines,» en *ICML*, 2010.

[42] D.-A. Clevert, T. Unterthiner y S. Hochreiter, *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)*, 2016. arXiv: 1511.07289 [cs.LG].

[43] Y. Zhao, X. Jin y X. Hu, «Recurrent convolutional neural network for speech processing,» en *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017, págs. 5300-5304.

[44] S. Hochreiter y J. Schmidhuber, «Long Short-Term Memory,» *Neural Computation*, vol. 9, n.º 8, págs. 1735-1780, 1997.

[45] S. Ioffe y C. Szegedy, «Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,» en *Proceedings of the 32nd International Conference on Machine Learning*, F. Bach y D. Blei, eds., ép. Proceedings of Machine Learning Research, vol. 37, Lille, France: PMLR, jul. de 2015, págs. 448-456.

[46] K. He, X. Zhang, S. Ren y J. Sun, «Deep Residual Learning for Image Recognition,» en *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, jun. de 2016.

[47] AgEagle Sensor Systems Inc., d/b/a MicaSense, *RedEdge–MX | MicaSense*, En línea; consultado el 14 de junio de 2021, 2021. dirección: https://micasense.com/rededge-mx/.

[48] S. Wold, K. Esbensen y P. Geladi, «Principal component analysis,» *Chemometrics and Intelligent Laboratory Systems*, vol. 2, n.º 1, págs. 37-52, 1987, Proceedings of the Multivariate Statistical Workshop for Geologists and Geochemists, ISSN: 0169-7439.

[49] RapidEye AG, *The RapidEye Red Edge Band*, En línea; consultado el 14 de junio de 2021. dirección: https://www.geoimage.com.au/CaseStudies/Red_Edge_White_Paper.pdf.

[50] Grupo de Inteligencia Computacional de la Universidad del País Vasco (UPV/EHU, *Hyperspectral Remote Sensing Scenes – Grupo de Inteligencia Computacional (GIC)*, En línea; consultado el 24 de junio de 2021. dirección: http://www.ehu.eus/ccwintco/index.php/Hyperspectral_Remote_Sensing_Scenes.

[51] Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS), *Inicio | Centro Singular de Investigación en Tecnoloxías Intelixentes – CiTIUS*, En línea; consultado el 14 de junio de 2021, 2021. dirección: https://citius.usc.es/.

[52] R. G. Congalton, «A review of assessing the accuracy of classifications of remotely sensed data,» *Remote Sensing of Environment*, vol. 37, n.º 1, págs. 35-46, 1991, ISSN: 0034-4257.

[53] Colaboradores de WikiChip, *Xeon Gold 5220 – Intel – WikiChip*, En línea; consultado el 23 de junio de 2021. dirección: https://en.wikichip.org/wiki/intel/xeon_gold/5220.

[54] NVIDIA Corporation, *NVIDIA Tesla V100 GPU Accelerator*, En línea; consultado el 23 de junio de 2021. dirección: https://images.nvidia.com/content/technologies/volta/pdf/tesla-volta-v100-datasheet-letter-fnl-web.pdf.

[55] The CentOS Project, *The CentOS Project*, En línea; consultado el 23 de junio de 2021. dirección: https://www.centos.org/.

[56] Python Software Foundation, *Welcome to Python.org*, En línea; consultado el 23 de junio de 2021. dirección: https://www.python.org/.

[57] Free Software Foundation, Inc., *GCC, the GNU Compiler Collection – GNU Project – Free Software Foundation (FSF)*, En línea; consultado el 23 de junio de 2021. dirección: https://gcc.gnu.org/.

[58] NVIDIA Corporation, *CUDA Zone | NVIDIA Developer*, En línea; consultado el 23 de junio de 2021. dirección: https://developer.nvidia.com/cuda-zone.

[59] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro y E. Shelhamer, *cuDNN: Efficient Primitives for Deep Learning*, 2014. arXiv: 1410.0759 [cs.NE].

[60] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai y S. Chintala, «PyTorch: An Imperative Style, High-Performance Deep Learning Library,» en *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox y R. Garnett, eds., Curran Associates, Inc., 2019, págs. 8024-8035.

[61] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke y T. E. Oliphant, «Array programming with NumPy,» *Nature*, vol. 585, n.º 7825, págs. 357-362, sep. de 2020.

[62] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot y E. Duchesnay, «Scikit-learn: Machine Learning in Python,» *Journal of Machine Learning Research*, vol. 12, págs. 2825-2830, 2011.

[63] Standard C++ Foundation, *Standard C++*, En línea; consultado el 23 de junio de 2021. dirección: https://isocpp.org/.

[64] OpenMP Architecture Review Board (ARB), *Home – OpenMP*, En línea; consultado el 23 de junio de 2021. dirección: https://www.openmp.org/.

[65] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard y L. D. Jackel, «Backpropagation Applied to Handwritten Zip Code Recognition,» *Neural Computation*, vol. 1, n.º 4, págs. 541-551, 1989.

[66] Y. A. LeCun, L. Bottou, G. B. Orr y K.-R. Müller, «Efficient BackProp,» en *Neural Networks: Tricks of the Trade: Second Edition*, G. Montavon, G. B. Orr y K.-R. Müller, eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, págs. 9-48, ISBN: 978-3-642-35289-8.

[67] D. P. Kingma y J. Ba, *Adam: A Method for Stochastic Optimization*, 2017. arXiv: 1412.6980 [cs.LG].

[68] X. Glorot e Y. Bengio, «Understanding the difficulty of training deep feedforward neural networks,» en *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, Y. W. Teh y M. Titterington, eds., ép. Proceedings of Machine Learning Research, vol. 9, Chia Laguna Resort, Sardinia, Italy: PMLR, mayo de 2010, págs. 249-256.

[69] A. M. Saxe, J. L. McClelland y S. Ganguli, *Exact solutions to the nonlinear dynamics of learning in deep linear neural networks*, 2014. arXiv: 1312.6120 [cs.NE].

[70] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville e Y. Bengio, *Generative Adversarial Networks*, 2014. arXiv: 1406.2661 [stat.ML].

Índice de figuras

Índice de cuadros