ESCOLA TÉCNICA SUPERIOR
DE ENXEÑARÍA

Trabajo de Fin de Máster

Clasificación de imágenes de teledetección con técnicas de aprendizaje profundo en situaciones de escasez de datos

Autor:
Álvaro Goldar Dieste

Tutores:
Francisco Santiago Argüello Pedreira
Dora Blanco Heras

Máster Interuniversitario en Computación de Altas Prestacións

Escola Técnica Superior de Enxeñaría

Departamento de Electrónica e Computación

Universidade de Santiago de Compostela

Santiago de Compostela, 16 de junio de 2022

Trabajo de Fin de Máster presentado para la obtención del Máster Interuniversitario en Computación de Altas Prestacións

Índice general

Siglas

AA Average Accuracy

ACGAN Auxiliary Classifier Generative Adversarial Networks

Adam Adaptive Moment Estimation

BAGAN Balancing Generative Adversarial Networks

CGAN Conditional Generative Adversarial Networks

CNN Convolutional Neural Network

CPU Central Processing Unit

CRS Countour Relaxed Superpixels

CUDA Compute Unified Device Architecture

cuDNN Compute Unified Device Architecture Deep Neural Network

ELU Exponential Linear Unit

ETPS Extended Topology Preserving Segmentation

GAN Generative Adversarial Networks

GCC GNU Compiler Collection

GPU Graphics Processing Unit

k Coeficiente kappa de Cohen

LeakyReLU Leaky Rectified Linear Unit

OA Overall Accuracy

PCA Principal Component Analysis

PReLU Parametric Rectified Linear Unit

RAM Random Access Memory

ReLU Rectified Linear Unit

ResGAN Residual Balancing Generative Adversarial Networks

ResNet Residual Network

SEEDS Superpixels Extracted via Energy–Driven Sampling

SLIC Simple Linear Iterative Clustering

SVM Support Vector Machine

VRAM Video Random Access Memory

WP WaterPixels

Resumen

Dentro del problema de clasificar imágenes multi e hiperespectrales, las técnicas basadas en aprendizaje profundo son las opciones más empleadas. El motivo es que son capaces de alcanzar mayores precisiones de clasificación que los métodos de aprendizaje automático tradicionales empleados anteriormente. Concretamente, las Redes Neuronales Convolucionales (CNN) han resultado ser la arquitectura más popular en la actualidad para resolver estos problemas de clasificación multiclase.

No obstante, aunque las arquitecturas neuronales son herramientas muy efectivas, también requieren una gran cantidad de datos de aprendizaje para extraer todo su potencial. Además, los conjuntos de datos que se emplean en teledetección tienden a presentar dos grandes problemas: la escasez de datos anotados y el desbalance entre clases, de modo que ciertas clases de elementos tienen muchas más muestras que otras. Esta problemática limita la capacidad de los métodos de aprendizaje automático, y lo hace especialmente en las arquitecturas neuronales. Considerando lo común que es el proceso de clasificación en teledetección, resulta de gran interés encontrar vías para minimizar el impacto de las limitaciones de los conjuntos de datos y evitar perjudicar así el desempeño de los clasificadores basados en aprendizaje profundo que se emplean en la actualidad en este dominio.

En este trabajo se propone un esquema de clasificación de imágenes de teledetección fundamentado en aprendizaje profundo, cuyo diseño ha sido guiado por el objetivo de minimizar el impacto de la escasez de muestras y desbalance de clases en los conjuntos de datos, para maximizar así la calidad de las clasificaciones realizadas. Para ello, se adopta como clasificador del esquema una arquitectura convolucional profunda –CNN residual–, al ser la base de los mejores clasificadores de la actualidad. El entrenamiento de esta red se apoya con técnicas de aumentado de datos. Por una parte, se aplican técnicas de aumentado tradicionales a las muestras de entrenamiento reales. Por otra parte, se integra la CNN residual en un esquema Generative Adversarial Networks (GAN), dentro del cual existe una red neuronal auxiliar que genere nuevas muestras sintéticas realistas que suministrar al clasificador durante el entrenamiento. Para mejorar la calidad de la información de entrenamiento, se aplica una etapa de segmentación en superpíxeles a la imagen como preprocesamiento, extrayéndose a posteriori una muestra real por cada segmento. Para maximizar la calidad de las clasificaciones generadas, se evalúa el comportamiento de las arquitecturas neuronales bajo una multitud de configuraciones diferentes, para identificar y preservar los criterios de diseño que les permiten alcanzar un mejor desempeño. También se aplican diversas optimizaciones computacionales para reducir todo lo posible el tiempo de entrenamiento del esquema desarrollado.

Desde el punto de vista investigador, el interés de este trabajo radica en la combinación de diferentes técnicas que se habitúan emplear por separado en la literatura, ahora bajo un único esquema de clasificación. Además, se adoptará un diseño de GAN todavía no evaluado en el campo de la teledetección, que presenta mejoras en su estructura para alcanzar mejores desempeños en problemas que trabajen con conjuntos de datos limitados.

Palabras clave: teledetección, clasificación de imágenes, aprendizaje profundo, segmentación en superpíxeles, redes neuronales convolucionales, redes residuales, aumentado de datos, GAN, BAGAN

Resumo

Dentro do problema de clasificar imaxes multi e hiperespectrais, as técnicas baseadas en aprendizaxe profunda son as opcións máis empregadas. O motivo é que son capaces de alcanzar maiores precisións de clasificación que os métodos de aprendizaxe automática tradicionais empregados anteriormente. Concretamente, as Redes Neuronais Convolucionais (CNN) resultaron ser a arquitectura máis popular na actualidade para resolver estes problemas de clasificación multiclase.

Con todo, aínda que as arquitecturas neuronais son ferramentas moi efectivas, tamén requiren unha gran cantidade de datos de aprendizaxe para extraer todo o seu potencial. Ademais, os conxuntos de datos que se empregan na teledetección tenden a presentar dous grandes problemas: a escaseza de datos anotados e o desbalance entre clases, de modo que certas clases de elementos teñen moitas máis mostras que outras. Esta problemática limita a capacidade dos métodos de aprendizaxe automática, e faio especialmente nas arquitecturas neuronais. Considerando o común que é o proceso de clasificación na teledetección, resulta de gran interese atopar vías para minimizar o impacto das limitacións dos conxuntos de datos e evitar prexudicar así o desempeño dos clasificadores baseados en aprendizaxe profunda que se empregan na actualidade neste dominio.

Neste traballo proponse un esquema de clasificación de imaxes de teledetección fundamentado en aprendizaxe profunda, cuxo deseño foi guiado polo obxectivo de minimizar o impacto da escaseza de mostras e desbalance de clases nos conxuntos de datos, para maximizar así a calidade das clasificacións realizadas. Para iso, adóptase como clasificador do esquema unha arquitectura convolucional profunda –CNN residual–, ao ser a base dos mellores clasificadores da actualidade. O adestramento desta rede apóiase con técnicas de aumentado de datos. Por unha banda, aplícanse técnicas de aumentado tradicionais ás mostras de adestramento reais. Por outra banda, intégrase a CNN residual nun esquema Generative Adversarial Networks (GAN), dentro do cal existe unha rede neuronal auxiliar que xere novas mostras sintéticas realistas que facilitar ao clasificador durante o adestramento. Para mellorar a calidade da información de adestramento, aplícase unha etapa de segmentación en superpíxeles á imaxe como preprocesamiento, extraéndose a posteriori unha mostra real por cada segmento. Para maximizar a calidade das clasificacións xeradas, avalíase o comportamento das arquitecturas neuronais baixo unha multitude de configuracións diferentes, para identificar e preservar os criterios de deseño que lles permiten alcanzar un mellor desempeño. Tamén se aplican diversas optimizacións computacionais para reducir todo o posible o tempo de adestramento do esquema desenvolvido.

Desde o punto de vista investigador, o interese deste traballo radica na combinación de diferentes técnicas que se habitúan empregar por separado na literatura, agora baixo un único esquema de clasificación. Ademais, adoptarase un deseño de GAN aínda non avaliado no campo da teledetección, que presenta melloras na súa estrutura para alcanzar mellores rendementos en problemas que traballen con conxuntos de datos limitados.

Palabras chave: teledetección, clasificación de imaxes, aprendizaxe profunda, segmentación en superpíxeles, redes neuronais convolucionais, redes residuais, aumentado de datos, GAN, BAGAN

Abstract

Within the problem of classifying multi and hyperspectral images, deep learning-based techniques are the most widely used ones. The reason is that they can achieve higher classification accuracies than the traditional machine learning methods previously employed. Specifically, Convolutional Neural Networks (CNNs) have proven to be the most popular architecture nowadays for solving these multi-class classification problems.

However, although neural architectures are very effective tools, they also require a large amount of learning data to extract their full potential. In addition, datasets used in remote sensing tend to have two major problems: sparsity of annotated data and imbalance between classes, so that certain classes of objects have many more samples than others. This problem limits the capacity of machine learning methods, especially in neural architectures. Considering how common the classification process is within remote sensing, it is of great interest to find ways to minimize the impact of dataset limitations and thus avoid impairing the performance of deep learning-based classifiers currently employed in this domain.

In this work, we propose a classification scheme for remote sensing images based on deep learning, whose design has been guided by the objective of minimizing the impact of sample sparsity and class imbalance in the datasets, to maximize the quality of the classifications performed. For this purpose, a deep convolutional architecture –residual CNN– is adopted as the classifier of the scheme, as it is the basis of the best classifiers nowadays. The training of this network is supported by data augmentation techniques. On the one hand, traditional augmentation techniques are applied to real training samples. On the other hand, the residual CNN is integrated into a Generative Adversarial Networks (GAN) scheme, within which there is an auxiliary neural network that generates new realistic synthetic samples to supply the classifier during training. To improve the quality of the training information, a superpixel segmentation stage is applied to the image as preprocessing, extracting a real sample for each segment afterwards. To maximize the quality of the generated classifications, the behavior of the neural architectures is evaluated under a multitude of different configurations, to identify and preserve the design criteria that allows achieving the best performance. Various computational optimizations are also applied to reduce as much as possible the training time of the developed scheme.

From the research point of view, the interest of this work lies in the combination of different techniques that are usually used separately in the literature, now present in a single classification scheme. In addition, a GAN design not yet evaluated in the remote sensing field will be adopted, which presents improvements in its structure to achieve better performances in problems working with limited datasets.

Keywords: remote sensing, image classification, deep learning, superpixel segmentation, convolutional neural networks, residual networks, data augmentation, GAN, BAGAN

1 | Introducción y contexto científico

En el campo científico de la teledetección, las cámaras de imagen multi e hiperespectral capturan la interacción de la luz con los objetos de una escena a un gran nivel de detalle, dando lugar 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, posibilitando por ejemplo aplicaciones de monitorización de la superficie terrestre. Entre ellas se pueden destacar la detección de cambios en regiones de cultivo [2], monitorización de ecosistemas (por ejemplo, la invasión de especies vegetales exóticas) [3], o la evolución de construcciones humanas [4].

Para llevar a cabo aplicaciones prácticas como estas es necesario realizar una multitud de tareas. Un proceso muy común se trata de la clasificación de imágenes, y por ello recibe mucha atención de la comunidad científica. Clasificar una imagen consiste en asignar a cada píxel de la misma una categoría que representa fielmente lo que se puede encontrar en él [5]. Este proceso 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.

Desafortunadamente, los conjuntos de datos que se emplean en teledetección tienden a presentar dos grandes problemas: la escasez de datos anotados que ejemplifiquen las categorías, y el desbalance entre estas clases, de modo que ciertas clases de elementos tienen muchas más muestras que otras [6]. El primer problema limita la capacidad de los métodos de aprendizaje automático que se emplean para la clasificación, al dificultarles adquirir un conocimiento completo del problema a tratar, mientras que el segundo factor dificulta aún más el modelado de las clases más escasas, impidiéndolo al completo en ocasiones.

Hace unos años, los métodos de aprendizaje automático tradicionales, como Support Vector Machine (SVM) [7] o Random Forests [8], eran las opciones más populares para clasificar imágenes en teledetección. A lo largo de la última década se ha incrementado enormemente la aplicación de redes neuronales profundas a una multitud de problemas del mundo real, incluida la visión por computador. El motivo es que han probado ser unas herramientas efectivas para resolver problemas cada vez más complejos, ante los cuales las técnicas de aprendizaje automático tradicional presentan ciertas limitaciones [9]. Por supuesto, esta tendencia ha alcanzado también el campo de la teledetección en los últimos años [10], con la consecuencia de que los métodos por excelencia en la actualidad para la clasificación en teledetección sean arquitecturas neuronales, y no métodos tradicionales, al poder alcanzar mayores precisiones [11]-[13].

De entre todas las arquitecturas neuronales que se han puesto a prueba en la visión por computador, la más popular de todas es la Red Neuronal Convolucional –Convolutional Neural Network (CNN) [14]–. Este tipo de red neuronal se basa en aplicar convoluciones a la información de entrada para extraer características de ella; aplicar una convolución a una imagen consiste, fundamentalmente, en emplear un filtro que realza ciertas características. Una CNN aplica numerosas convoluciones de

forma sucesiva: unas para identificar diferentes características, y otras para combinar gradualmente características simples –como bordes de objetos– en características más complejas –como formas–, que ya permitan discriminar unas categorías de otras con precisión. Esta arquitectura ha permitido alcanzar grandes avances no solo en la visión por computador, sino también en otros campos como el procesamiento auditivo, y por supuesto ha terminado siendo la arquitectura por excelencia para la clasificación en teledetección, al demostrar una gran capacidad para modelar la naturaleza no lineal de las imágenes multi e hiperespectrales.

Dentro de la clasificación en teledetección se pueden seguir dos enfoques muy diferentes para determinar la categoría de un píxel dado. El primero consiste en hacer que el clasificador analice exclusivamente las características espectrales del píxel de interés [15]. Por otra parte, el segundo enfoque introduce información espacial en la predicción, al hacer que el clasificador analice las características tanto del píxel objetivo como de los píxeles vecinos a este, para extraer una información de contexto con la que alcanzar un mejor desempeño. Por poner un ejemplo concreto, la distinción de especies vegetales puede ser muy complicada si tan solo se analiza la variación de un tono de verde a otro –información espectral–, mientras que si se analiza un conjunto de píxeles –información espectral y espacial– se pueden llegar a determinar, por ejemplo, texturas de copas de árboles que permitan distinguir claramente unas especies vegetales de otras. Mientras que métodos tradicionales como SVM requieren el apoyo de otras técnicas para extraer información espectral y espacial de las muestras, las CNN son una arquitectura que puede extraer automáticamente ambos tipos de información, simplificando el diseño de un clasificador para teledetección. Esta característica, sumada a la gran capacidad de las arquitecturas neuronales para modelar la naturaleza no lineal de las imágenes de teledetección, hacen que las CNN que extraen información espectral y espacial [16]-[18] sean las que consiguen realizar clasificaciones más precisas [11]-[13], siendo la base de los mejores clasificadores desarrollados hoy en día en el campo.

En cualquier caso, aunque las redes neuronales son en general herramientas muy efectivas, también requieren una gran cantidad de datos de aprendizaje para extraer todo su potencial, agravando así los problemas previamente mencionados acerca de los conjuntos de datos de teledetección [13]. Esta problemática, junto con lo común que es el proceso de clasificación, hacen que sea de gran interés encontrar vías para minimizar el impacto de dichas limitaciones, y evitar perjudicar así el desempeño de clasificadores basados en aprendizaje profundo para este dominio. Este problema se puede afrontar desde diversos puntos de vista, que se recogen a continuación:

1. **Extracción de características profundas.**

   Puesto que la información de aprendizaje es limitada, cuanto más efectivo sea el clasificador al extraer de las muestras disponibles características representativas de las categorías a modelar, mejor podrá realizar la tarea de clasificación.

   Considerando el caso concreto de diseñar una CNN la intuición dice que, cuanto más profunda sea la red –cuantas más convoluciones concatene–, podrá extraer de la información características más elaboradas para discernir mejor una categoría dada de otras. Es decir, una posible vía para combatir las limitaciones de los conjuntos de datos de teledetección es emplear arquitecturas convolucionales muy profundas como clasificadores para explotar lo máximo posible la escasa información de aprendizaje disponible. Precisamente por este motivo, las grandes arquitecturas convolucionales son la base de muchos trabajos presentes en la literatura de la teledetección para la tarea de clasificación [19], [20].

2. **Aumentado de datos.**

En lugar de aumentar la responsabilidad del clasificador, otra alternativa consiste en generar sintéticamente nuevas muestras para aumentar la riqueza de los datos de aprendizaje, y mejorar en consecuencia la generalización del clasificador.

Aunque el modo más común para realizar un aumentado de datos es aplicando transformaciones a las muestras ya existentes, también se ha experimentado recientemente con arquitecturas neuronales conocidas como Generative Adversarial Networks (GAN) [21] que permiten crear muestras completamente nuevas a partir de una estimación de la distribución de los datos de referencia [22]. Este tipo de arquitectura neuronal contiene dos redes, pudiendo ser una el clasificador convolucional tradicional, y la otra la red generadora de datos nuevos que se suministran a la primera para hacer así el aumentado.

Este último tipo de técnica de aumentado neuronal tiene un gran potencial frente a las técnicas tradicionales, y por ello ha sido propuesta como alternativa [23]. En cualquier caso, el entrenamiento de las GAN tiende a ser problemático en situaciones de escasez de datos [24], por lo que un requerimiento para su utilización es disponer de conjuntos de datos de considerable tamaño. Esto limita considerablemente la aplicación de este tipo de arquitectura neuronal en la teledetección. Al mejor saber de los autores, no es habitual complementar las arquitecturas GAN con técnicas de aumentado tradicionales para introducir una salvaguarda en caso de que falle el aumentado de datos basado en aprendizaje profundo. Para empeorar más la situación, las GAN son todavía más problemáticas ante conjuntos de datos desbalanceados, de modo que tienden a generar siempre la misma muestra sintética para una clase minoritaria o incluso no capturan en absoluto su distribución, generando así puro ruido [25].

3. **Extracción de muestras de mayor calidad.**

   Al clasificar una imagen incluyendo información espectral y espacial, el procedimiento habitual consiste en dividir la imagen en pequeños fragmentos llamados *parches*. A continuación, cada parche se procesa individualmente como si fuera una imagen independiente, y una vez que todos ellos han sido categorizados, la imagen completa también lo habrá sido. El tamaño óptimo de los parches dependerá del conjunto de datos que se esté procesando, utilizándose tamaños mayores cuanto mayor sea la resolución espacial del sensor con el que se ha capturado, pero siempre intentando que contengan mayoritariamente píxeles de una única clase. La técnica más común para generar parches es utilizar una ventana deslizante, extrayendo un parche centrado sobre cada píxel a procesar en la imagen [10]. Sin embargo, con este sencillo enfoque muchos parches contendrán inevitablemente píxeles de categorías diferentes a la del píxel objetivo, especialmente en los bordes de los objetos de la escena, introduciendo en cierto modo "impurezas" en la información que se pretende que represente fielmente a la categoría asociada.

   Dentro de la visión por computador, una técnica habitual para reducir el consumo de recursos computacionales consiste en segmentar las imágenes que se van a procesar. La segmentación en superpíxeles es un paso de preprocesamiento que agrupa píxeles similares en superpíxeles, conformando regiones homogéneas y contiguas con un tamaño relativamente constante, lo cual permite tratar todos los píxeles de un superpíxel como un único elemento en posteriores procesamientos [26], [27]. En el caso concreto de la clasificación en teledetección, la segmentación permite reducir el coste de clasificar toda una imagen, ya que en lugar de extraer un parche para cada píxel a clasificar utilizando una ventana deslizante, se puede extraer un parche para cada superpíxel y asignar la categoría predicha a todo píxel que pertenezca a él. La precisión de la

clasificación no será peor, ya que los superpíxeles agrupan píxeles similares, y sí se acelerará en varios órdenes de magnitud [3], [28]-[30]. En resumen, se procesa menos información redundante.

Aunque la intención habitual en la literatura es incorporar la segmentación en superpíxeles para reducir el consumo computacional, también puede emplearse para aumentar la calidad de las muestras de entrenamiento si se enfoca desde otro punto de vista. Si el tamaño del parche se asemeja al tamaño de los superpíxeles, se consigue que los parches contengan principalmente píxeles de la categoría que pretenden representar. Aunque se extraerán en total menos muestras que trabajando con una ventana deslizante, no hay realmente ningún perjuicio. Las muestras obtenidas en base a una ventana deslizante proporcionarán mucha información redundante, al haber de parche a parche una traslación de un solo píxel, y la información contenida será más ruidosa si llegan a contener más píxeles de las categorías restantes que de la categoría objetivo. En comparación, las muestras en base a superpíxeles reportarán una información de aprendizaje igual de rica, e incluso más filtrada para mejorar la calidad de la misma. En definitiva, incorporar una segmentación en superpíxeles puede llegar a mejorar la calidad de las muestras extraídas de un conjunto de datos de teledetección, impactando de forma positiva en la calidad de los clasificadores entrenados.

Con todo ello, mediante la realización de este Trabajo de Fin de Máster se pretende desarrollar un esquema de clasificación basado en aprendizaje profundo para imágenes de teledetección, que trata de minimizar el impacto de las restricciones de escasez y desbalance de clases en los conjuntos de datos para no limitar la capacidad del clasificador neuronal. Para ello, se combinarán diferentes técnicas que se habitúan emplear por separado en la literatura, además de incorporar un diseño alternativo de arquitectura GAN que aún no ha sido traído al campo de la teledetección [25]. Desde el punto de vista investigador, estas últimas serían las contribuciones más destacables que resultarían del cumplimiento de los objetivos propuestos para este Trabajo de Fin de Máster. Más concretamente:

1. Para explotar al máximo la información espectral y espacial de las escasas muestras de aprendizaje, se utilizará como clasificador del esquema una arquitectura convolucional muy profunda.

2. Para tratar de incrementar la riqueza de la información de entrenamiento, y por ende mejorar la generalización del clasificador, se aplicarán técnicas de aumentado a las muestras extraídas para crear otras nuevas:

   a) El clasificador se integrará en una arquitectura GAN para que una red neuronal auxiliar asista en su aprendizaje contribuyendo con la generación de nuevas muestras. Para tratar de resolver el problema de aprender a modelar categorías minoritarias, se adoptará la arquitectura propuesta en [25], cuya eficacia aún no ha sido evaluada en la teledetección, pero parece realmente prometedora acorde a su publicación original.

   b) Como salvaguarda a que la GAN no pueda contribuir en algunos casos de forma significativa al aumentado con una variedad de nuevas muestras, también se aplicarán técnicas de aumentado tradicional a los datos de entrenamiento. Esta limitación podría darse en casos en los que no se disponga de suficientes muestras de referencia con las que estimar adecuadamente la distribución de algunas categorías.

3. Para facilitar todavía más al clasificador adquirir un buen conocimiento del problema durante su aprendizaje, se incorporará como etapa de preprocesamiento una segmentación en superpíxeles que guiará la posterior extracción

de muestras, con la intención de mejorar la calidad de la información de entrenamiento extraída del conjunto de datos.

4. Para maximizar la calidad de clasificación del esquema resultante de combinar todas estas técnicas, se evaluará el comportamiento de las arquitecturas neuronales bajo una multitud de configuraciones diferentes, para identificar y preservar los criterios de diseño que les permiten alcanzar un mejor desempeño.

5. Precisamente para facilitar la prueba y comparación de tantas configuraciones de las arquitecturas neuronales como sea posible, conviene tratar de reducir todo lo posible el tiempo de ejecución del esquema desarrollado. Para ello, se aplicarán diferentes optimizaciones en cuanto consumo de recursos computacionales para acelerar el aprendizaje de las arquitecturas neuronales.

2 | Esquema de clasificación propuesto

Este capítulo presentará el esquema de clasificación de imágenes de teledetección diseñado en este trabajo. Se considerará su aplicación a imágenes multi e hiperespectrales, con el objetivo de que aprenda a asignar a cada muestra de las imágenes la categoría o clase más apropiada.

El esquema que se propone se trata de un clasificador supervisado basado en la segmentación de la imagen en superpíxeles, junto con una arquitectura de aprendizaje profundo de tipo Balancing Generative Adversarial Networks (BAGAN). Posteriormente, en el Capítulo 4 se estudiarán diferentes estrategias para optimizar el uso de recursos computacionales de estas arquitecturas neuronales, así como para maximizar el desempeño en la tarea de clasificación.

La Figura 2.1 muestra una visión de alto nivel del esquema de clasificación propuesto. En concreto, los principales puntos que se han tenido en cuenta para el diseño del mismo son los siguientes:

1. **La primera etapa del esquema consiste en la segmentación de la imagen en superpíxeles**.

   El objetivo de esta etapa es guiar el sucesivo proceso de extracción de información de la imagen, generando una muestra por cada segmento resultante. La segmentación en superpíxeles permitirá mejorar la calidad de la información de entrenamiento suministrada a las arquitecturas neuronales, a la vez que se reduce el coste computacional de la clasificación total de la imagen.

   Se escogerá el algoritmo de segmentación en superpíxeles WaterPixels (WP) [31], puesto que presenta ventajas en cuanto a la calidad de los segmentos generados y un reducido tiempo de computación.

2. **La arquitectura neuronal diseñada es de tipo BAGAN, y consta a su vez de otras dos subarquitecturas:**

   a) El autoencoder es la primera red neuronal a entrenar en este esquema. Se le suministrarán las muestras de entrenamiento reales con el objetivo de que componga un entendimiento inicial de la información contenida en la imagen. Una vez finalizado el entrenamiento del autoencoder, su conocimiento será transferido a la arquitectura GAN que se encuentra a continuación.

   b) La arquitectura GAN consta de dos redes neuronales, un discriminador D y un generador G, que se entrenarán de forma adversaria. El objetivo de G es aprender a generar muestras sintéticas que puedan hacerse pasar por muestras reales de la imagen. Por otra parte, D recibe durante su entrenamiento tanto muestras reales como estas muestras sintéticas de G, con el objetivo de aprender a identificar cuáles son falsas, además de aprender a determinar la categoría a la que pertenecen. El entrenamiento competitivo radica en que a G le interesa ser capaz de engañar a D,

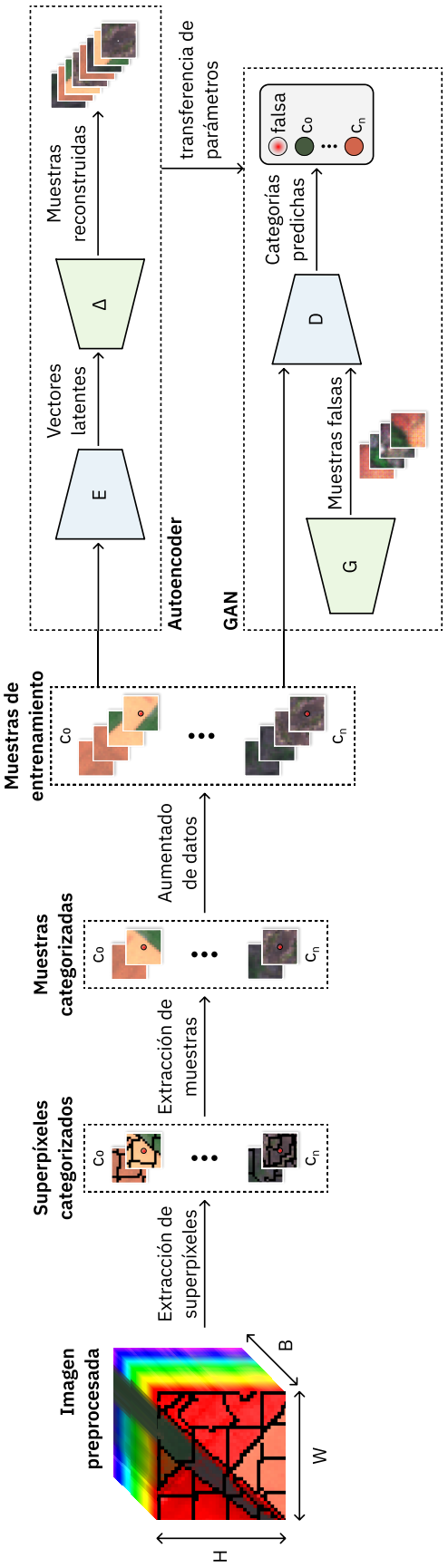


Figura 2.1: Representación de alto nivel del esquema de clasificación de imágenes para teledetección desarrollado en este trabajo. Nótese cómo se combinan técnicas para extraer muestras de entrenamiento de mayor calidad, para extraer características profundas de las mismas, y de aumentado de datos de entrenamiento, todo ello con el objetivo de maximizar el desempeño de la clasificación final de las imágenes de teledetección.

viéndose incentivada a crear muestras lo más realistas posible. Consecuentemente, el hecho de suministrárselas a D junto con las muestras reales termina dando lugar un aumentado de datos, puesto que D podrá aprender a realizar la clasificación aprovechando tanto información real como información sintética.

Cabe resaltar que la red D adoptará una arquitectura CNN residual –Residual Network (ResNet)– para explotar todo lo posible la información espectral y espacial de las muestras que se le suministren, y aprender así a realizar lo mejor posible la clasificación.

3. **Para asistir más al proceso de entrenamiento del clasificador, se aplicará una técnica de aumentado de datos tradicional a las muestras reales disponibles.**

   En concreto, la técnica se basará en el uso de rotaciones y volteos para generar múltiples muestras de entrenamiento por cada muestra real de la imagen, dando lugar así a un conjunto de muestras de entrenamiento mucho mayor que el de partida.

Una vez concluido el entrenamiento de la arquitectura GAN, se desactivará en D la funcionalidad de discriminar si una muestra es falsa o no, de modo que solo determine la categoría que más se adecúe a las muestras analizadas. Esta red será el clasificador de imágenes de teledetección que resulta del entrenamiento de todo el esquema diseñado.

A lo largo de este capítulo de la memoria se introducirán gradualmente las diferentes técnicas que se han combinado en este esquema, para comprender el funcionamiento y contribución de cada una. En primer lugar, se explicará qué es una segmentación en superpíxeles y cómo combinarla con la tarea de clasificar una imagen. A continuación, se repasarán las arquitecturas neuronales que se habitúan emplear en la actualidad para esta tarea. Y finalmente se explicarán las técnicas neuronales que se comienzan a utilizar en la actualidad para realizar aumentado de datos.

2.1. Segmentación de una imagen en superpíxeles

Al capturar una escena del mundo real para almacenarla como una imagen digital se genera una estructura matricial que la contenga. La imagen –matriz– tiene un alto que denotaremos H y un ancho W, y cada píxel –elemento de la matriz– se encuentra caracterizado por un cierto valor en cada canal del espectro –o banda de la imagen–, siendo B el total de bandas.

Para que un programa informático trabaje con una imagen, el enfoque más natural es que lea la información tal cual se almacena, accediendo a cada elemento de esta matriz. Sin embargo, también es interesante preguntarse por qué trabajar con las imágenes a nivel de píxel, en lugar de ir un paso más allá [26]. 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 concreto 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, cada vez más algoritmos de visión por computador resultan 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 teledetección, por la gran

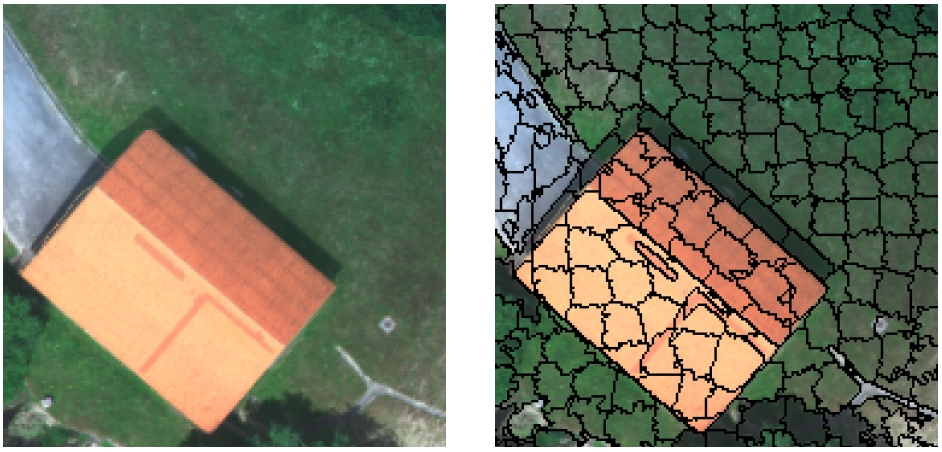
complejidad espectral que pueden llegar a tener las imágenes, con decenas o cientos de bandas.

A raíz de estas consideraciones surge el concepto de *superpíxel*. Este pretende ser una estructura más natural sobre la que basar el procesamiento de una imagen –sobre la que *acceder* a la información del mundo real que se captura–.

2.1.1. Qué es un superpíxel

Un superpíxel agrupa píxeles contiguos con características similares, conformando una región homogénea. Esta no tiene por qué tener una forma concreta, sino que puede adaptarse en tamaño y forma a los objetos de la escena a segmentar. Por ejemplo, un superpíxel de un tejado de una casa tratará de limitarse a dicho objeto para agrupar píxeles naranjas, dejando que los píxeles del suelo se agrupen en otros superpíxeles. Esto constituye una diferencia fundamental frente a métodos de segmentación más triviales como, por ejemplo, aquellos que simplemente particionan 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 las distintas estructuras podrán ser muy grandes. Por ejemplo, el tejado de una casa podría ser mucho más pequeño que un gran jardín. En realidad, los superpíxeles consideran las similitudes espectrales de los píxeles junto con 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 hará que los superpíxeles mantengan un tamaño relativamente semejante a lo largo de la imagen, aparte de dejar una cierta libertad a adaptarse a los objetos de la escena. Por claridad, la Figura 2.2 refleja esta propiedad de los superpíxeles sobre una imagen de prueba que ha sido segmentada en superpíxeles.



(a) Región de la imagen de entrada.

(b) Región con la segmentación en superpíxeles resultante superpuesta.

Figura 2.2: Resultado de segmentar una imagen en superpíxeles. Nótese cómo los superpíxeles presentan un tamaño semejante, pero también tratan de adaptarse a los objetos presentes en la escena.

La agrupación de píxeles similares para conformar elementos de mayor tamaño tiene dos consecuencias de gran interés:

1. Se reduce en gran medida el número total de elementos a tratar cuando

se procesa la imagen completa, reduciendo la carga computacional de los posteriores algoritmos que trabajen con la imagen segmentada.

2. Además, puesto que cada superpíxel agrupa píxeles similares, procesar cada uno de estos segmentos en lugar de cada píxel individual reduce en cierto sentido la información redundante que se introduce en posteriores algoritmos cuando se procesa la imagen completa.

En definitiva, un mapa de segmentación en superpíxeles puede reemplazar la estructura matricial original de la imagen, con la ventaja de presentar un menor número de elementos que todavía siguen representando la variedad de información que el sensor ha capturado del mundo real.

2.1.2. Superpíxeles en la clasificación en teledetección

Con todo ello, es importante concretar de qué forma un clasificador para teledetección se ve afectado por la incorporación de una segmentación en superpíxeles. Al final se está reduciendo considerablemente el número de primitivas disponibles –superpíxeles en lugar de píxeles–, lo cual se traduce en muchas menos muestras de la imagen de las que dispone el clasificador.

En un problema de clasificación supervisada, una muestra se compone de (1) una cierta información que la caracteriza, y de (2) la categoría a la que pertenece. La forma habitual de proceder al trabajar con un clasificador para teledetección a nivel de píxel es la que sigue[1]:

- **Para generar la información que caracteriza la muestra:** se toma una subregión de la imagen de $N \times N$ píxeles centrada sobre el píxel objetivo, siendo N un valor de vecindad dado.
- **Para determinar la categoría de la muestra:** se toma directamente la categoría que haya sido asignada al píxel objetivo en la información de referencia.

Por otra parte, trabajar en base a superpíxeles abre un nuevo abanico de posibilidades, al tener que seleccionar 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 [30], 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 la variabilidad que pueda haber entre los diferentes miembros del superpíxel, así como información de contorno, se incluirán los píxeles vecinos que se encuentren en un determinado rango de búsqueda de tamaño $N \times N$ píxeles situado sobre el píxel central. Esta subregión conformará el parche de entrada al clasificador. Por claridad, todo este procedimiento se ilustra en la Figura 2.3.

- **Para determinar la categoría de la muestra:** se tendrán en cuenta las categorías a las que pertenecen todos los píxeles del superpíxel.

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

[1] Cabe recordar que en el Capítulo 1 se enunció el mejor desempeño de las arquitecturas neuronales que trabajan con parches en lugar de píxeles individuales, siendo la base de los clasificadores para teledetección a día de hoy.

Todo este procedimiento es generalizable a un problema de aprendizaje no supervisado. Tan solo es necesario obviar la generación de la categoría del superpíxel, al ser esta información inexistente en las muestras de este paradigma.

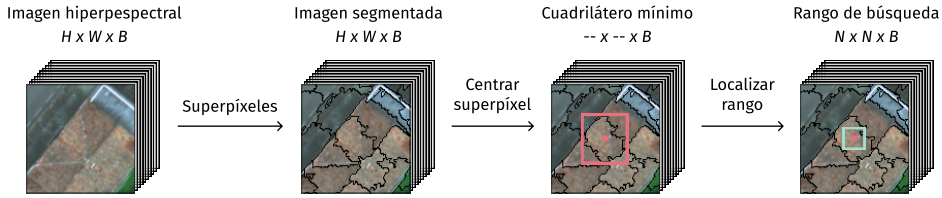


Figura 2.3: Proceso de posicionamiento del rango de búsqueda sobre un superpíxel dado. Nótese que la marca roja central representa la posición del píxel central calculado para el segmento.

Una vez concretado cómo generar muestras a partir de los superpíxeles, es posible analizar el impacto que ello pueda tener tanto en términos de precisión del clasificador como en aceleración de su ejecución. Todo clasificador de aprendizaje supervisado pasa, a grandes rasgos, por dos etapas en su ciclo de vida: una fase de entrenamiento –o aprendizaje–, y una fase de prueba –o evaluación–.

- **Impacto en la fase de entrenamiento.** En esta etapa, el clasificador extrae toda la información de utilidad que es capaz de los datos suministrados, para aprender lo mejor posible a realizar su tarea de clasificación.

  Tanto si se emplea la imagen a nivel de píxel como a nivel de superpíxel, lo razonable para poder realizar una comparación en términos equitativos es que el clasificador reciba el mismo número de muestras durante el entrenamiento. Esto equiparará la riqueza de la información suministrada en ambos casos, de modo que el clasificador tenga las mismas oportunidades de adquirir un conocimiento completo del problema enfrentado.

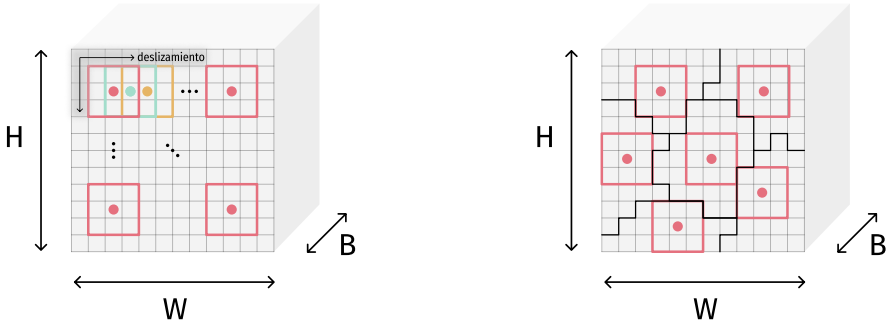
  Por una parte, el uso de superpíxeles no debería tener un impacto en el tiempo de entrenamiento del clasificador frente al caso de usar píxeles, al suministrarle el mismo número de muestras durante esta etapa. Por otra parte, sí es posible que se pueda llegar a apreciar cierto impacto positivo en la precisión obtenida por el clasificador al suministrarle una información de aprendizaje de mayor calidad. Cabe recordar la explicación del Capítulo 1, sobre cómo el uso de superpíxeles en lugar de una ventana deslizante para extraer muestras de la imagen posibilita la generación de información de mayor calidad, al reducir considerablemente las probabilidades de que una muestra de una categoría concreta contenga píxeles pertenecientes a otras categorías.

- **Impacto en la fase de prueba.** En esta etapa 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 la clasificación se podría realizar prediciendo individualmente la categoría de cada píxel de prueba, resulta mucho más interesante realizarla a nivel de superpíxel. Como los superpíxeles son regiones homogéneas –contienen píxeles similares–, es normal preguntarse hasta qué punto merece el esfuerzo predecir la categoría de cada píxel por separado, si se presupone que sus vecinos son similares y, consecuentemente, deberían pertenecer a la misma categoría. Es decir, podría predecirse una única categoría para cada superpíxel, y asignársela a todos los píxeles que contenga el segmento. Por claridad, en la Figura 2.4

se representan ambos enfoques sobre el procedimiento de clasificación de una imagen. En el caso de procesar la imagen a nivel de píxel –Figura 2.4(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 2.4(b)– simplemente es necesario realizar una predicción por superpíxel.

Por una parte, como el número de predicciones a realizar para clasificar la imagen usando superpíxeles es considerablemente menor, debería reducirse enormemente el coste de clasificar la totalidad de la imagen. Siendo $S \times S = S^2$ el tamaño medio de píxeles/superpíxel, 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 originales a nivel de píxel. Además, las precisiones de clasificación finales no deberían reducirse en absoluto dado que los superpíxeles no degradan la riqueza de la información procesada, sino que simplemente reducen su redundancia. Es más, incluso se podría llegar a observar cierta mejora de precisión en el clasificador por el hecho de suministrarle muestras de mayor calidad durante su aprendizaje, como se comentó previamente.



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

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

Figura 2.4: 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.

2.1.3. Algoritmo WaterPixels para la segmentación en superpíxeles

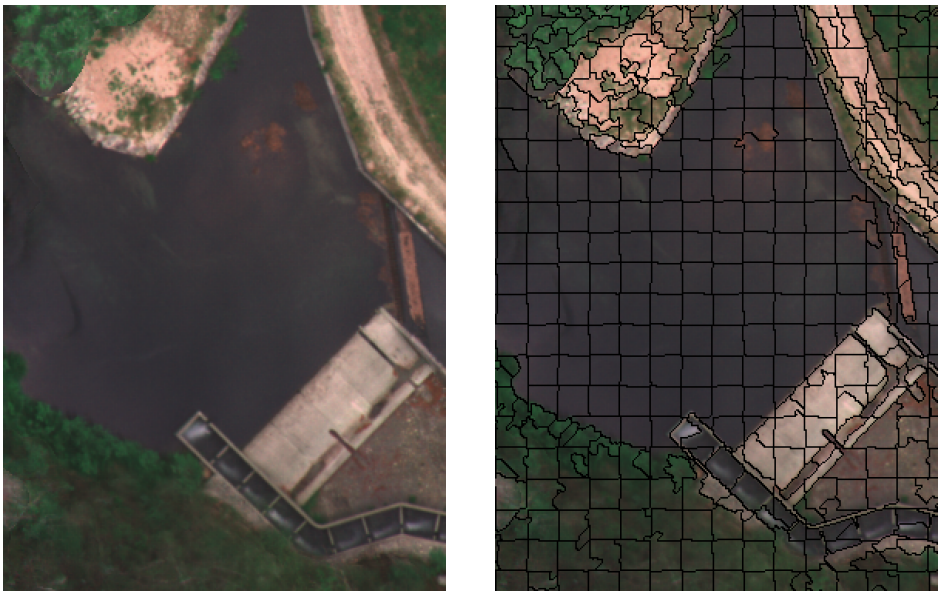
En la bibliografía se han aplicado una gran variedad de técnicas basadas en superpíxeles a numerosos problemas de visión por computador [27], [29], [30], [32], existiendo también muchas propuestas de algoritmos de segmentación en superpíxeles, como Countour Relaxed Superpixels (CRS) [33], Extended Topology Preserving Segmentation (ETPS) [34], o Superpixels Extracted via Energy–Driven Sampling (SEEDS) [35], entre otras. A grandes rasgos, estos algoritmos pueden ser categorizados en dos grupos [27]:

G1. Algoritmos basados en el descenso del gradiente.

G2. Algoritmos basados en grafos.

Los algoritmos de superpíxeles basados en grafos tienden a tener un coste computacional mayor, y tampoco suelen ofrecer control explícito sobre el tamaño de los segmentos –que incide en el total de superpíxeles generados– o su regularidad —libertad a adaptarse a objetos de la escena, haciendo que los superpíxeles sean más o menos cuadrados– [36]. Teniendo en cuenta estas consideraciones, para la teledetección resulta de interés centrar la atención en los algoritmos basados en el descenso del gradiente. El menor coste computacional de esta aproximación es una ventaja añadida dada la gran complejidad espectral de imágenes con decenas o cientos de bandas, mientras que la personalización de la segmentación permitirá adaptar adecuadamente los resultados a las características espaciales de la imagen con la que se trabaje.

De entre este último tipo de algoritmos, WP [31] resulta ser una técnica de segmentación en superpíxeles notablemente popular. A pesar de su sencillez, este algoritmo es capaz de conseguir segmentaciones de buena calidad y con costes computacionales reducidos [27], [36], además de permitir personalizar fácilmente el tamaño y regularidad deseados para los superpíxeles. Es por todo ello que se escogerá este algoritmo para incorporarlo como etapa de preprocesamiento en superpíxeles al esquema de clasificación desarrollado. La Figura 2.5 muestra el resultado de generar una segmentación con WP sobre una región de una imagen de prueba.



(a) Región de la imagen de entrada.

(b) Región con la segmentación resultante superpuesta.

Figura 2.5: Resultado de generar una segmentación con WP sobre una región de una imagen de prueba. Se ha especificado un tamaño medio de 400 píxeles/superpíxel ($S \times S = 400$) –lo que equivale a superpíxeles de 20 píxeles de lado aproximadamente ($S = \sqrt{400} = 20$)–, indicando que se fusionen superpíxeles de menos de 100 píxeles para conformar segmentos mayores, y empleando un factor de compacidad de 0,5 puntos.

2.2. Arquitecturas neuronales para clasificar imágenes

Dentro de los diferentes paradigmas del aprendizaje automático, el aprendizaje supervisado resulta ser el más extendido para la clasificación en teledetección. El motivo es que es capaz de proporcionar, en general, un mejor desempeño que los modelos basados en aprendizaje no supervisado [11].

En el aprendizaje supervisado se suministran a un clasificador muestras categorizadas de antemano para que aprenda a realizar su tarea. El objetivo del clasificador es extraer características representativas de cada muestra, deduciendo qué características se presentan en cada categoría –o clase– y cuáles no. De este modo, el clasificador compone un conocimiento general sobre qué trazas disciernen a una clase de las demás, para poder aplicarlo durante la etapa de prueba. De modo más formal, 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 ajustarse 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 es puesto a prueba ante nuevas muestras que no ha visto durante el entrenamiento, con el objetivo de explotar 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.

2.2.1. La Red Neuronal Convolucional

Las redes neuronales pertenecen a un subcampo del aprendizaje automático, comúnmente conocido como *aprendizaje profundo*, en el cual se trata de replicar el funcionamiento del cerebro humano para realizar una gran variedad de tareas complejas [9]. Generalmente, estas estructuras que son las redes neuronales se componen de un conjunto 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. Denotando W_l como 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 principal al entrenar un clasificador neuronal 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 la salida deseada ante una información de entrada dada. Por ejemplo, en el caso de la clasificación de imágenes debería ser capaz de indicar la categoría a la que pertenece un píxel de una imagen. Para ello, la red debería aprender a extraer e identificar características de los diferentes elementos de la imagen con objeto de poder discriminar entre las diferentes categorías.

Dentro del campo de la visión por computador, una de las arquitecturas basadas en aprendizaje profundo más populares durante los últimos años es la Red Neuronal Convolucional –CNN– [14]. Para facilitar su entendimiento, es conveniente introducir en primer lugar el concepto de convolución de una imagen.

En esencia, la operación de convolución consiste en aplicar un filtro sobre una información espacial –o posiblemente espectral y espacial– de entrada, con el objetivo

de identificar características de los elementos presentes en ella; por ejemplo, podría aplicarse un filtro de delineado para resaltar bordes de los objetos presentes. Las convoluciones son una herramienta muy extendida y utilizada en la visión por computador y en el aprendizaje automático asociado.

A modo de ejemplificación, en la Figura 2.6 se puede observar el resultado de aplicar una convolución a una imagen. Para generar un píxel de la imagen de salida, es 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 denomina *convolución 2D*.

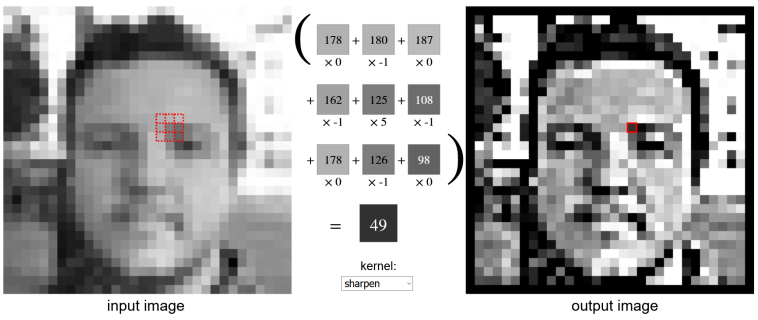


Figura 2.6: Aplicación, a una imagen en escala de grises, de una matriz de convolución correspondiente a un efecto de nitidez [37].

Retomando la CNN, el bloque básico de construcción de este tipo de red es la capa convolucional. Esta toma una determinada información de entrada x, que bien podría ser una imagen o tan solo 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 es

$$y = \{W_i * x + b_i\}_{i=1,2,\ldots,N}, \tag{2.1}$$

siendo $*$ la operación de convolución. En un caso como el de la Figura 2.6, al aplicar N diferentes filtros sobre una imagen en escala de grises –una única banda– se obtienen 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 espacial, en lugar de la capa densa tradicional de las redes neuronales[2], aporta una serie de ventajas para nada despreciables:

1. Por simplicidad, supóngase el caso más sencillo de una capa que genere una única banda de salida. Si se emplease 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 filtro que se comparte a lo largo de toda la información, y que conllevará 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 $i+1$.

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 ayudar a evitar la posibilidad de que surjan problemas como el sobreajuste –u *overfitting*–.

2. Por otra parte, para procesar una información de entrada espacial con la capa densa es necesario aplanarla, concatenando los valores de los diferentes píxeles en cada banda, unos detrás de otros. En contraposición, una capa de convolución permite conservar la estructura espacial original, permitiendo a una capa convolucional el analizar y extraer información espacial de la entrada, además de la información espectral. Cabe recordar que en el Capítulo 1 se comentó que los clasificadores que consideran información espectral y espacial son los que consiguen mejores resultados.

Otro bloque de gran importancia en una CNN es la capa de reducción de muestreo –también llamada *pooling*–. Esta reduce la dimensionalidad espacial de una información de entrada dada. 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 2.7 se representa la aplicación de una operación de *max–pooling*. Como bien su nombre indica, la salida de cada región es el elemento de mayor 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.

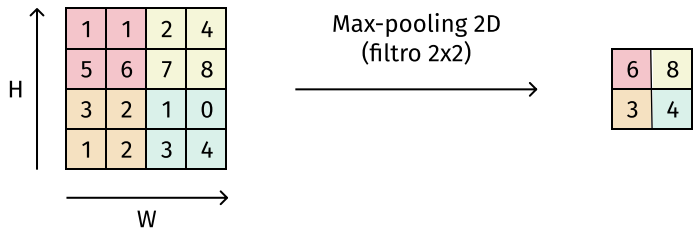


Figura 2.7: Representación de la aplicación de una operación de *max–pooling* a una región de entrada con una única banda [38].

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

1. Generalizar las características extraídas por las capas convolucionales.

   Si las capas convolucionales procesasen directamente los mapas de características generados por capas convolucionales anteriores, terminarían asociando las características extraídas a posiciones espaciales concretas. En cambio, al aplicar la reducción de dimensionalidad al mapa generado por una capa convolucional se mezclan las características presentes en cada región del mapa, eliminando en cierto modo sus vinculaciones espaciales, al informar tan solo de que ciertas características están presentes en puntos indefinidos dentro sus regiones correspondientes. Cuantas más capas de reducción haya presentes a lo largo de una arquitectura convolucional, más se abstraerán los mapas de características de las posiciones espaciales, evitando generar dependencias espaciales en el conocimiento del clasificador, que limitarían su capacidad final de generalización.

2. Además, la reducción de dimensionalidad permite reducir considerablemente la carga computacional de las siguientes capas de la red, al reducir el número de elementos a procesar.

La combinación de capas convolucionales y de capas de *pooling* constituye el plan 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 elaboradas y discriminativas de la misma gracias a la combinación de convoluciones y operaciones de *pooling*.

Yendo un paso más allá, es incluso posible combinar la funcionalidad de las capas convolucionales y las capas de *pooling* en capas convolucionales con desplazamiento –o *stride*– [39]. El *stride* permite a una convolución omitir una parte de los elementos en la información a procesar; por ejemplo, un *stride* de 2 hará que la capa convolucional procese uno de cada dos elementos que se encuentre en la información de entrada. De este modo, además de extraer características visuales, la capa convolucional también estará reduciendo la dimensionalidad de la información a cada paso.

El interés del *pooling* reside en que, al incorporar la responsabilidad de la reducción de dimensionalidad a una capa con parámetros ajustables, se permite en cierto modo que la red también "aprenda" durante su entrenamiento cuáles son los tipos de reducción de dimensionalidad que le resultan más útiles, en lugar de prefijar por ejemplo operaciones de *valor máximo* o de *valor medio*. Es decir, se dota de una mayor flexibilidad a la arquitectura neuronal para aprender más sobre la tarea que debe realizar.

Tanto si se emplean parejas de capas convolucionales y de *pooling*, como si tan solo se incorporan capas convolucionales con *stride*, todo lo presentado hasta el momento se podría considerar como una primera subred presente en toda CNN.

Tras ello, la CNN habitúa incorporar una segunda subred compuesta exclusivamente por capas densas. La tarea de estas consiste en tomar todas las características espectrales y espaciales extraídas por la primera subred, y procesarlas para clasificar la información de entrada dada, asignándole la categoría más apropiada de entre todas las posibles. Para ello, la subred transforma las características en un vector unidimensional, y lo procesa progresivamente hasta terminar con un vector resumen con tantos elementos como posibles categorías. Al aplicar la función *softmax* a este último se asigna una probabilidad P de pertenencia a cada categoría i, de modo que la suma de todas las probabilidades sea 1:

$$P(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{C} e^{x_j}}, \tag{2.2}$$

$$\sum_{i=1}^{C} P(x_i) = 1, \tag{2.3}$$

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

En la práctica, toda la CNN se trata como una arquitectura monolítica 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 de búsqueda de los filtros más adecuados 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 seleccionar los filtros manualmente.

2.2.2. Las Redes Convolucionales Residuales

Considerando el caso concreto del diseño de una CNN, la intuición nos dice que, cuanto más profunda sea la red, podrá extraer características más elaboradas de la información para discriminar mejor una categoría dada de las demás.

Ahora bien, para conseguir este objetivo no se puede encadenar un gran número de capas unas tras otras sin más, dado que esto suele dar lugar a diversos problemas que dificultan enormemente el aprendizaje de la red. Entre estos efectos no deseados se encuentran la degradación de las características extraídas en el proceso de inferencia [40], o un deterioro del gradiente en el proceso de retropropagación [41]. Es por ello que terminó surgiendo el concepto de *aprendizaje residual* [40], que introduce "atajos" –también conocidos como conexiones de salto– de capas de bajo nivel de la red a capas de alto nivel. Estas evitan el deterioro de la información a medida que se propaga a lo largo de arquitecturas muy profundas, para solventar así dichos problemas.

Las redes muy profundas que incorporan conexiones de salto se conocen comúnmente como Redes Residuales –ResNet–, y se construyen encadenando una multitud de elementos conocidos como *bloques residuales*. La Figura 2.8 representa la arquitectura de uno de estos bloques.

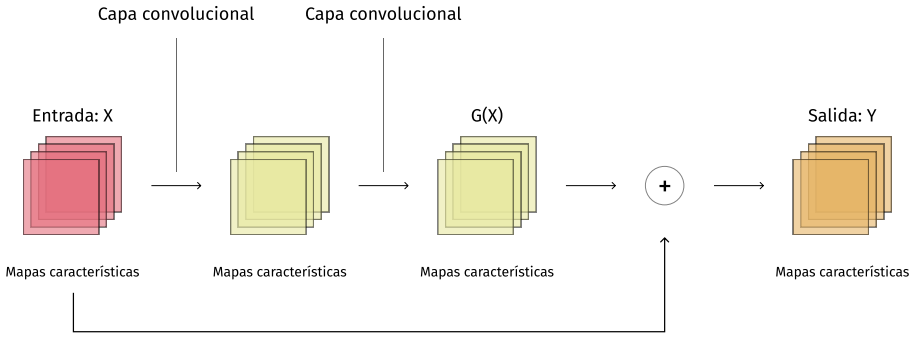


Figura 2.8: Esquema de la arquitectura de un bloque residual en una CNN. Nótese cómo la información de entrada X se propaga también hacia la salida sin haber sufrido alteraciones.

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

$$Y = G(X) + X. \tag{2.4}$$

De este modo, un bloque residual consigue que su información original de entrada se propague a capas que se encuentren tras él, saltando las capas intermedias, lo cual permitirá reducir problemas tales como la degradación de las características o el deterioro del gradiente, a la hora de diseñar una red muy profunda.

Gracias a ello y con el paso de los años, la tendencia en el diseño de redes CNN ha pasado de arquitecturas con dos o tres capas [14], a arquitecturas con decenas o cientos de capas [42], [43] que constituyen la base de las arquitecturas neuronales modernas para visión por computador por sus altas precisiones, a la vez que resultan fáciles de comprender y de entrenar. Por supuesto, la tendencia también se ha trasladado al ámbito de la clasificación en teledetección [19], [20], [44], y es por ello

que el clasificador del esquema que se desarrolla en este trabajo se tratará de una CNN residual, cuya estructura se presentará a continuación.

2.2.3. CNN residual como clasificador

Antes de presentar la arquitectura residual a integrar en el esquema de clasificación propuesto, es necesario definir en primer lugar un componente de mayor complejidad que un simple bloque residual, y que será sobre el que se base la arquitectura. 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, de modo que el número de mapas de características generados tras cada convolución se mantendrá constante por cada etapa. Las redes que contienen este tipo de módulos se conocen normalmente con CNN residuales o ResNet.

Además, teniendo en cuenta la definición dada en la Ecuación (2.4), es importante asegurarse de que X presente el mismo número de mapas de características que Y para poder realizar la suma de ambos. Dado que al primer bloque residual de una etapa nada le asegura que X contenga el mismo número de bandas que las que se generarán en Y, se introduce al inicio de cada etapa residual una capa convolucional adicional que se encarga de cambiar la dimensionalidad de la información de entrada dada, de modo que coincida con la esperada.

El clasificador contendrá tres etapas residuales, componiéndose cada una de tres bloques residuales. Las convoluciones irán incrementando el número de filtros tras cada etapa, para extraer características cada vez más abstractas y discriminativas con las que realizar la clasificación final. Se incorporará además un *stride* de 2 elementos en la primera convolución de las etapas 2 y 3, para realizar también una reducción de dimensionalidad en estos puntos.

Con todo ello, la salida de la primera subred de la CNN residual no constará exclusivamente de 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, basándose en las características que la CNN profunda haya conseguido extraer.

La arquitectura neuronal residual que se propone se tratará de un modelo con un gran número de parámetros que ajustar durante el aprendizaje. Estos casos tienden fácilmente a sufrir de problemas como el *overfitting*, en comparación a modelos más sencillos, y precisamente por ello es común incorporar una capa de *dropout* tras cada capa convolucional para estabilizar el entrenamiento de redes CNN profundas [45]. La idea detrás de esta técnica consiste en desactivar aleatoriamente unidades de la arquitectura neuronal y sus conexiones, eliminándolas a efectos prácticos del procesamiento de datos para forzar al modelo a no depender demasiado de

ninguno de sus componentes, lo cual desemboca normalmente en entrenamientos más estables. Se incorporará entonces una capa de *dropout* a toda capa convolucional del clasificador, y la probabilidad con la que se desactivarán aleatoriamente las unidades de la arquitectura es un parámetro de gran importancia que será fijado con detenimiento más adelante en este trabajo.

La función de activación también es otro factor que influye de forma considerable en la estabilidad y desempeño de una arquitectura neuronal. Existen una multitud de funciones de activación habitualmente empleadas en redes CNN, como Rectified Linear Unit (ReLU) [46], Exponential Linear Unit (ELU) [47], Parametric Rectified Linear Unit (PReLU) [48], o Leaky Rectified Linear Unit (LeakyReLU) [49]. No es posible determinar que una sea necesariamente mejor que las demás, y por ello será otro factor que se estudiará con detenimiento más adelante en el trabajo para fijarlo.

Toda la arquitectura descrita en esta sección del documento se refleja visualmente en la Figura 2.9 por una mayor claridad. A su vez, el Cuadro 2.1 detalla las características de cada capa de la arquitectura. Cabe resaltar que Z se trata de un parámetro de dimensionalidad que se definirá en la siguiente sección de este capítulo.

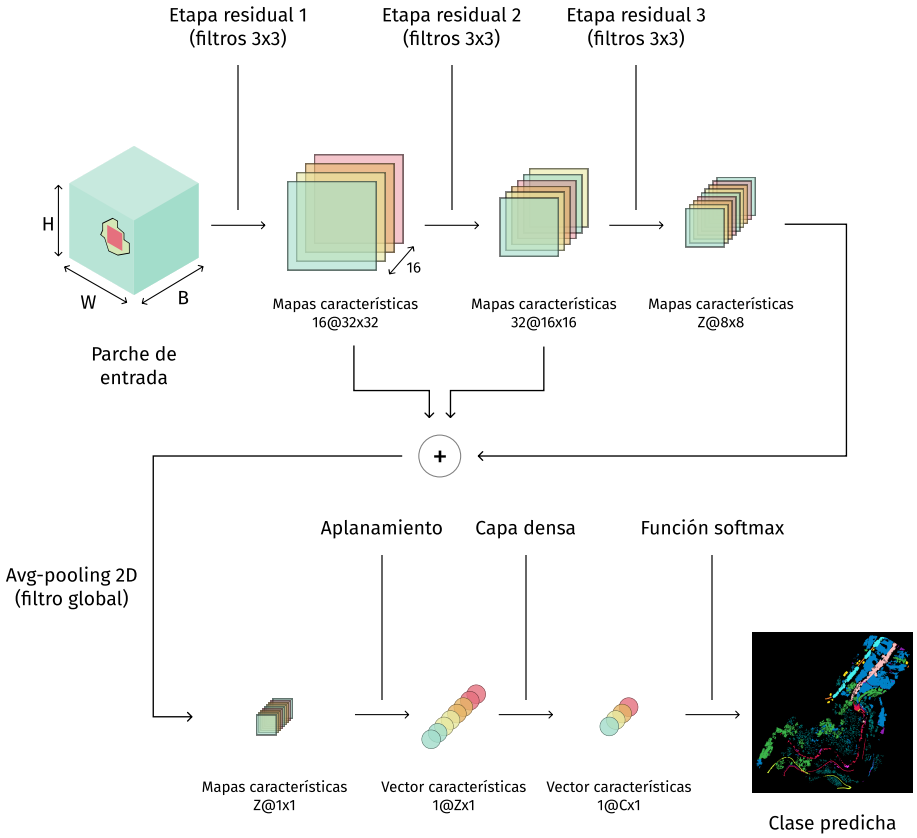


Figura 2.9: Esquema de la arquitectura de la CNN residual propuesta como clasificador del esquema de clasificación a desarrollar. Las dimensiones de salida de cada capa son las correspondientes a una entrada con dimensiones $H \times W \times B = 32 \times 32 \times B$. Nótese cómo las características empleadas en clasificación final se extraen de las tres etapas residuales, cada una centrada en extraer características más abstractas que la anterior.

Cuadro 2.1: Detalles de la arquitectura de la CNN residual propuesta como clasificador del esquema de clasificación a desarrollar. Las dimensiones de salida de cada capa son las correspondientes a una entrada con dimensiones $H \times W \times B = 32 \times 32 \times B$. 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 3×3, y les seguiría también la función de activación que se escoja para las demás capas en posteriores experimentos de este trabajo.

# etapa	Componente	Dim. salida	Activación	Tam. filtro	Desplazamiento
1	Bloque residual	$32 \times 32 \times 16$	(ver Capítulo 4)	3×3	1×1
	Bloque residual	$32 \times 32 \times 16$	(ver Capítulo 4)	3×3	1×1
	Bloque residual	$32 \times 32 \times 16$	(ver Capítulo 4)	3×3	1×1
2	Bloque residual	$16 \times 16 \times 32$	(ver Capítulo 4)	3×3	2×2
	Bloque residual	$16 \times 16 \times 32$	(ver Capítulo 4)	3×3	1×1
	Bloque residual	$16 \times 16 \times 32$	(ver Capítulo 4)	3×3	1×1
3	Bloque residual	$8 \times 8 \times Z$	(ver Capítulo 4)	3×3	2×2
	Bloque residual	$8 \times 8 \times Z$	(ver Capítulo 4)	3×3	1×1
	Bloque residual	$8 \times 8 \times Z$	(ver Capítulo 4)	3×3	1×1
–	Avg-pool. 2D	$1 \times 1 \times Z$	–	8×8	8×8
–	Aplanamiento	$Z \times 1$	–	–	–
–	Capa densa	$C \times 1$	Softmax	–	–

2.3. Aumentado de datos con Redes Generativas Adversarias

Tal y como se explicó anteriormente, todo método de aprendizaje automático requiere datos de entrenamiento para aprender a realizar la tarea para la cual se diseña. Cuanta más información de referencia haya disponible, más completo será el entendimiento que genere acerca de la tarea en cuestión, pudiendo alcanzar así un mejor desempeño. Esta necesidad se ve todavía más incrementada al emplear técnicas de aprendizaje profundo, que pueden alcanzar mayores rendimientos que métodos de aprendizaje automático tradicional, aunque también a costa de requerir una mayor cantidad de datos de los que aprender.

El aumentado de datos es un procedimiento común que se aplica durante el entrenamiento de un método de aprendizaje automático. Su objetivo es generar sintéticamente nuevas muestras de entrenamiento para suministrar durante el aprendizaje, además de las ya disponibles en el conjunto de datos del problema, con el resultado de que se enriquece la variedad de la información de referencia, permitiendo que el clasificador adquiera un conocimiento más completo del problema objetivo.

Aunque existen multitud de técnicas de aumentado [50], en general se pueden clasificar en los siguientes tres grupos:

- Técnicas que aplican transformaciones a una muestra para convertirla en una nueva. Pueden ir desde simples transformaciones geométricas, como rotaciones o traslaciones, hasta procesos más complejos.
- Técnicas que combinan diferentes muestras de la misma categoría para crear nuevas muestras. Pueden ir desde simples combinaciones lineales hasta alternativas más elaboradas.
- Técnicas basadas en la arquitectura GAN, que es un tipo de esquema neuronal que crea nuevas muestras a partir de una estimación de la distribución de los

datos originales [21].

Este último tipo de técnicas de aumentado de datos basadas en aprendizaje profundo tienen un gran potencial en comparación a las técnicas tradicionales, llegando a ser propuestas como un paso adelante en la resolución de este problema [23]. Como es de esperar, su uso se ha extendido también al ámbito de la clasificación en teledetección, para mejorar las prestaciones de los clasificadores desarrollados [22], [51].

2.3.1. La arquitectura GAN

La invención de las arquitecturas GAN en el año 2014 [21] marcó una gran innovación en el ámbito del aprendizaje profundo: supuso pasar de diseñar arquitecturas neuronales en las que una red trabaja de forma autónoma, a diseñar arquitecturas en las que diferentes redes neuronales colaboran entre sí para alcanzar un objetivo.

La arquitectura GAN se originó como una técnica para el modelado generativo de datos. Es decir, mientras que el objetivo de una arquitectura clasificadora es predecir las categorías a las que pertenecerían muestras dadas de un problema, el objetivo de una arquitectura generativa sería aprender a componer "de la nada" muestras que podrían haber formado parte de los datos disponibles para el problema.

Para ello, la arquitectura GAN involucra dos redes neuronales que se entrenan de forma competitiva –es decir, siendo adversarias–: un Discriminador (D) y un Generador (G). D se trata de una red clasificadora cualquiera, como una CNN[3], que recibe durante su aprendizaje tanto muestras reales – del conjunto de datos del problema–, como muestras falsas –generadas artificialmente por G–. Mientras que el objetivo de D es discernir si las muestras son reales o no, el objetivo de G es generar muestras indistinguibles de las reales. Las dos redes se entrenan de forma adversaria, de modo que una mejora en una de ellas causa un incentivo en la otra red para mejorar, y no quedarse atrás. El punto de equilibrio en el aprendizaje se alcanza cuando las muestras reales son indistinguibles de las generadas por G, de modo que a D no le queda más alternativa que recurrir al puro azar para determinar si una muestra dada es real o no. Por claridad, la Figura 2.10 representa la estructura de una arquitectura GAN.

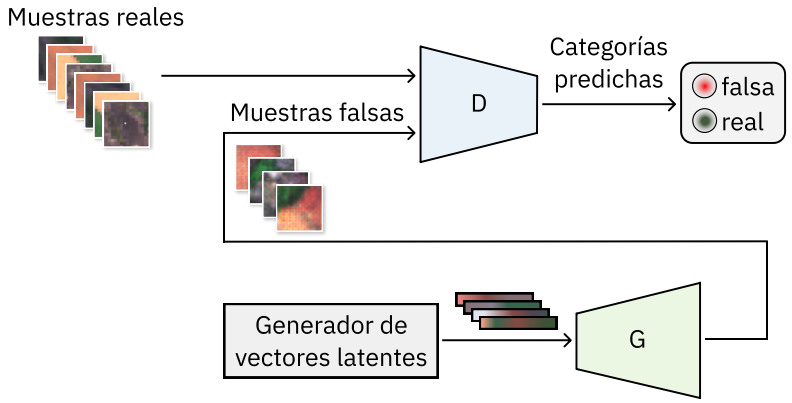


Figura 2.10: Esquema de una arquitectura GAN. Nótese cómo D trabaja tanto con las muestras reales del problema, como con las muestras creadas por G.

[3]Por simplicidad, se trasladará la explicación de la arquitectura GAN a las aplicaciones de visión por computador. En cualquier caso, los fundamentos detrás del funcionamiento de la arquitectura GAN son perfectamente extrapolables a la aplicación de técnicas de aprendizaje profundo a cualquier otro campo.

Al igual que se comentó anteriormente cómo una red discriminadora es capaz de modelar la categoría a la que pertenece una muestra, también es conveniente concretar cómo una red generadora puede llegar a crear muestras para un problema desde cero. Para ello, es necesario introducir antes un nuevo tipo de capa convolucional: la *convolución traspuesta*.

Una convolución traspuesta presenta múltiples diferencias en su comportamiento frente a una convolución estándar. La más interesante radica en que, en una convolución estándar, para procesar una imagen de entrada se centra el filtro convolucional sobre cada elemento disponible, dando lugar a un elemento de salida por cada elemento de entrada. En lugar de ello, una convolución traspuesta proporciona más libertad al posicionamiento del filtro, haciendo que además de poder centrarse sobre cada píxel de la entrada, también se puedan situar, por ejemplo, los bordes del filtro sobre ciertos píxeles de la entrada. Por claridad, la Figura 2.11 ejemplifica la aplicación de una convolución traspuesta de 3×3 elementos a una entrada de 5×5 elementos, con el resultado de que la salida generada presenta unas dimensiones espaciales mayores gracias a la mayor libertad en cuanto a posicionamiento de los filtros.

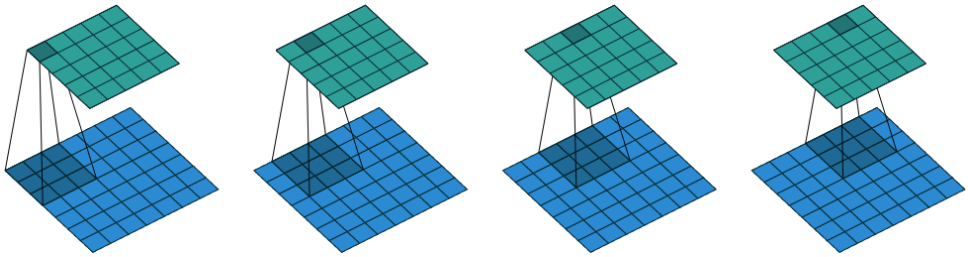


Figura 2.11: Resultado de aplicar una convolución traspuesta de 3×3 elementos a una entrada de 5×5 elementos [52].

El funcionamiento concreto de esta operación se detalla en [52], pero en cualquier caso, lo interesante al final es que las propiedades de la convolución traspuesta hacen que, si se aplica con los mismos parámetros que una convolución estándar, pueda revertir la transformación espacial de esta última. Por ejemplo, al aplicar una convolución estándar con un *stride* de dos elementos se reducirá a la mitad la dimensionalidad espacial de la información de entrada, y si se aplica a continuación una convolución traspuesta con los mismos parámetros, se duplicará la dimensionalidad espacial de la información para dejarla como al inicio.

Este comportamiento de sobremuestreo, junto con el hecho de que los filtros convolucionales sean parámetros ajustables durante el entrenamiento, convierte a la convolución traspuesta en la herramienta idónea para crear una red neuronal generadora. Teniendo presente que:

- D toma una muestra, y la procesa con sucesivas convoluciones hasta reducirla a características representativas que asociar a una categoría.
- Y el objetivo de G es alcanzar información análoga a la que entra en D.

Se podría diseñar una red generadora con convoluciones traspuestas para realizar el proceso de D a la inversa, y alcanzar así el funcionamiento deseado para G. En concreto:

- D reduce una muestra dada a un vector unidimensional de características que se procesan en la subred clasificadora. En G podría emplearse un generador de números aleatorios para conformar el vector de características de partida, de tamaño fijo Z.

- Este vector aleatorio –normalmente denominado *vector latente*– podría reordenarse para conformar una muestra visual de baja resolución; por ejemplo, de 4×4 elementos.
- La aplicación de sucesivas convoluciones traspuestas a esta muestra inicial terminarían convirtiéndola en una muestra de mayor resolución, hasta igualar la dimensionalidad de las muestras reales. que entran en D.

Durante el entrenamiento se ajustarán los parámetros de D de modo que sus convoluciones aprendan a extraer características cada vez más abstractas de la información visual que se le suministra, de cara a realizar la clasificación final. De forma inversa, los parámetros de las convoluciones de G se ajustarán de modo que puedan convertir información abstracta del vector latente –valores aleatorios– en características cada vez más concretas, para terminar conformando una muestra que pudiese parecer real.

2.3.2. La arquitectura CGAN

Aunque la generación de muestras con una GAN es muy interesante, también presenta la gran limitación de que no se puede controlar en ningún momento el tipo de muestras generadas en caso de estar trabajando ante un problema multiclase. Precisamente para solventar esta limitación, la arquitectura GAN evolucionó con el paso del tiempo a la arquitectura Conditional Generative Adversarial Networks (CGAN) [53].

La idea fundamental de la arquitectura CGAN es introducir, de algún modo, información relativa a una categoría y en cada muestra suministrada a las redes:

- Si la generación del vector latente se encuentra condicionada por y, G generará muestras falsas para cualquier categoría del problema bajo demanda.
- Y D recibe con la muestra a discriminar entre real o falsa, la categoría que pretende representar, para tener también en cuenta la correspondencia con las muestras reales de dicha categoría.

El procedimiento más habitual para incorporar la información y en una arquitectura CGAN consiste en recurrir a capas de *embedding* [54]. El origen de este tipo de capa neuronal se remonta al procesamiento de lenguaje natural. En este campo, una capa de *embedding* se configura para generar vectores unidimensionales de un tamaño fijo n a partir de un rango de entradas limitado; por ejemplo, se configura para ser capaz de generar un vector diferente para 1000 posibles entradas. En la práctica, la utilidad del *embedding* para el procesamiento de lenguaje natural es que los parámetros se ajustan durante el entrenamiento, de modo que dentro de ese conjunto de 1000 palabras, aquellas con significado similar generen vectores también semejantes.

Al trasladar la capa de *embedding* a la generación de imágenes, lo habitual es crear vectores de la misma dimensionalidad Z que el espacio latente. De este modo, al generar un vector latente, puede introducirse la información de categoría y en él mediante operaciones que lo combinen con el vector resultante del *embedding*; por ejemplo, una simple multiplicación. El efecto resultante durante el entrenamiento de G será que la capa de *embedding* se ajustará de modo que genere representaciones diferentes para cada posible categoría del problema, con el consecuente de que los vectores latentes –valores aleatorios– que se introduzcan en G terminen siendo considerablemente distintos en función de la categoría asociada. Así se dotará al esquema CGAN de la capacidad de generar muestras para cualquier categoría bajo demanda.

Con todo ello, la arquitectura presentada anteriormente en la Figura 2.10 se traduciría ahora en la representada en la Figura 2.12.

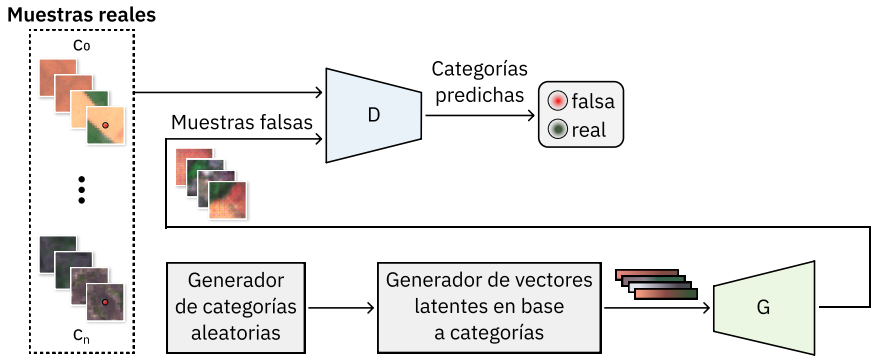


Figura 2.12: Esquema de una arquitectura CGAN. Nótese cómo los vectores latentes generados dependen ahora de las categorías deseadas.

2.3.3. Las arquitecturas ACGAN y BAGAN

A pesar de que una CGAN ya sea capaz de aprovechar la información de clases del problema a tratar, el discriminador de la arquitectura no es capaz de realizar ninguna función más allá de determinar si las muestras presentadas son reales o no. Una evolución natural del esquema CGAN consiste en dotar a D de la capacidad de determinar la categoría perteneciente a las muestras procesadas, además de discernir si son reales o falsas. Así es como terminan surgiendo las arquitecturas Auxiliary Classifier Generative Adversarial Networks (ACGAN) [55] y BAGAN [25]. Esta última arquitectura se trata de una evolución de la ACGAN que trata de solventar ciertas carencias, consiguiendo mejores desempeños en la práctica, y por ello será sobre la que se centren las siguientes explicaciones.

Para que D sea ahora capaz de predecir la categoría de las muestras presentadas, aparte de si son reales o falsas, es necesario alterar la estructura de su salida. Concretamente, ante un problema multiclase con C categorías, una arquitectura BAGAN hace que D categorice cada muestra de entrada en $C + 1$ posibles clases: una clase por cada categoría real del problema, y una clase adicional para indicar que la muestra analizada es falsa, independientemente de la clase que pretenda modelar. Es decir, D realizará ahora una clasificación multiclase en lugar de una binaria. Esto da lugar a que la arquitectura presentada anteriormente en la Figura 2.12 se convierta en la reflejada en la Figura 2.13.

De entre las diferencias que se pueden citar entre la arquitectura ACGAN y su evolución BAGAN, la primera de ellas es que una ACGAN recurre a un enfoque diferente para que D categorice las muestras. Concretamente, preserva la salida binaria original que discrimina entre muestras reales y falsas, e incorpora en paralelo una salida que categoriza la muestra en una de las posibles C clases, independientemente de que sea real o falsa. Consecuentemente, D se entrena en base a dos funciones objetivo diferentes. Este enfoque tiene la consecuencia negativa de que, al aplicar una ACGAN a un problema con un conjunto de datos con desbalance entre clases, se dificulta enormemente que G aprenda a modelar cualquier categoría poco representada, con la consecuencia de que se tiende a generar siempre la misma muestra falsa para ella, o incluso puro ruido. Es precisamente por ello que la arquitectura BAGAN altera la salida de D de modo que se optimice en base a una única función objetivo [25].

La segunda gran diferencia entre una ACGAN y una BAGAN es que esta última

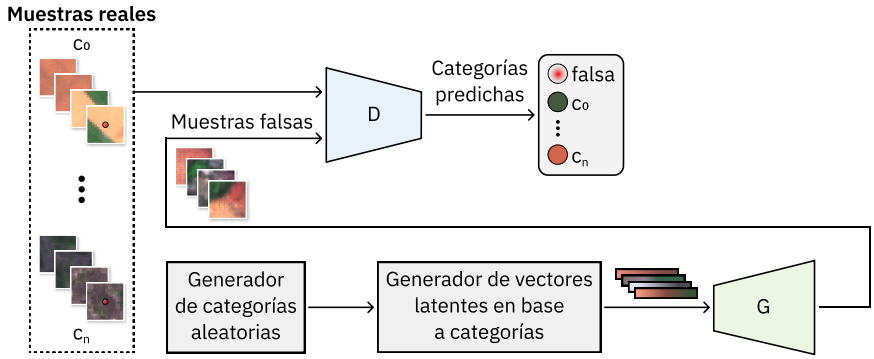


Figura 2.13: Esquema de una arquitectura BAGAN. Nótese cómo D determina ahora la categoría de las muestras presentadas, además de indicar si son reales en los casos apropiados, o directamente falsas.

incorpora a mayores, antes del entrenamiento de la arquitectura GAN como tal, una fase de entrenamiento de un autoencoder [25], [56]. El objetivo de esta arquitectura neuronal no supervisada es aprender a codificar la información suministrada para generar representaciones resumidas de la misma, y ser capaz de reconstruir –decodificar– la información original a partir de estos resúmenes lo mejor posible. En el caso de procesar información visual, la parte de codificación se corresponde con la extracción de características al igual que lo hace D, y la parte de decodificación se corresponde con la generación de imágenes a partir de características al igual que lo hace G. Como esta última etapa se realiza con representaciones que pierden parte de la información original al estar comprimidas, se fuerza al decodificador a encontrar similitudes entre las muestras suministradas para facilitar la reconstrucción, haciendo que, en cierto sentido, capture la distribución de los datos del problema objetivo.

La cuestión es que las arquitecturas GAN son muy complejas de entrenar, al requerir un delicado equilibrio en el aprendizaje de sus dos redes para que progresen simultáneamente, en lugar de que una se quede detrás de la otra permanentemente, rompiendo el entrenamiento adversario basado en que una incentive a la otra a mejorar. Las GAN sufren sobre todo ante problemas con escasez de datos de entrenamiento [24], algo muy común en el campo de la teledetección. Por otra parte, un autoencoder es mucho más fácil de entrenar que una GAN, pero no tiene la capacidad de generar muestras para categorías determinadas bajo demanda.

Al final, el interés reside en compensar las debilidades de una arquitectura con las fortalezas de la otra, dando lugar a una arquitectura neuronal mayor que se entrenará en tres etapas:

E1. El autoencoder será la primera parte a entrenar. Se le suministrarán todas las muestras reales del problema durante esta fase, ignorando las categorías de las mismas, para capturar la distribución de los datos a tratar.

E2. Haciendo que la estructura de D sea análoga a la del codificador, y que la de G sea análoga al decodificador, se podrán transferir los parámetros del autoencoder a la arquitectura GAN una vez entrenado el primero. A efectos prácticos, se transfiere a la GAN el conocimiento adquirido por el autoencoder.

E3. El entrenamiento de la GAN comenzará desde este punto, en lugar de hacerlo desde una inicialización aleatoria de sus parámetros. Esto permite a la GAN comenzar desde un punto de conocimiento más estable, al tener ya cierta familiaridad con el problema objetivo, por lo que debería encontrarse muchos menos problemas durante su entrenamiento.

Precisamente por las características de escasez de datos y de desbalance entre clases que suelen darse en los conjuntos de datos de teledetección, la BAGAN es una arquitectura ideal para incorporar al esquema de clasificación que se propone en este trabajo. Su propósito será englobar al clasificador descrito previamente en el Apartado 2.2.3 –red D–, para proveerle de un aumentado de datos basado en aprendizaje profundo al suministrarle otra red G durante el entrenamiento las muestras falsas generadas. De este modo, se conseguiría que D construya un conocimiento más completo del problema objetivo para terminar alcanzando un mejor desempeño.

Los detalles concretos de la arquitectura de G se reflejan en el Cuadro 2.2. En general, cabe resaltar que Z es un factor que puede influir considerablemente en el desempeño de la arquitectura desarrollada; si se parte de vectores latentes muy pequeños, a las capas convolucionales de G se les dificultará convertirlos en imágenes realistas al contener tan poca información, mientras que si se parte de vectores latentes muy grandes, a las capas convolucionales se les dificultará manejar tanta información. Por ello, se estudiará el impacto de Z con detenimiento más adelante en el trabajo para fijar su valor. Sucede lo mismo con las funciones de activación de G, que también se concretarán posteriormente, al igual que para D. La excepción es la activación de la última capa convolucional, que se trata de la función tangente. El motivo es que los datos del problema a tratar serán escalados al rango $[-1, 1]$ antes de suministrárselos al esquema de clasificación, por lo que es necesario forzar que la salida de G se adhiera a este rango para hacerse pasar por muestras reales. Otro aspecto a resaltar es que no es común incorporar capas de *dropout* a los componentes de un generador en una arquitectura GAN, dado que suele perjudicar el rendimiento global. Ahora bien, sí es común observar mejoras en el desempeño al incorporar etapas de normalización a las salidas de las capas convolucionales de D y de G, y por ello se aplicará una normalización espectral [57] a todas ellas.

Cuadro 2.2: Detalles de la arquitectura de la red generadora de datos a integrar en el esquema de clasificación desarrollado. Las dimensiones de salida son las correspondientes a un vector latente de entrada de Z elementos. Antes de cada convolución se aplica un relleno de un elemento en las dimensiones H y W de la información. La capa de *embedding* transforma la categoría dada en un vector de Z elementos. Este se combina con el vector latente generado mediante una multiplicación elemento a elemento. El resultado se reestructura en una imagen de dimensiones $H \times W \times B = 1 \times 1 \times Z$ que se introduce en la primera convolución traspuesta para proceder a la generación de la imagen falsa.

Componente	**Dim. salida**	**Activación**	**Tam. filtro**	**Desplazamiento**
Embedding	$Z \times 1$	–	C	–
Conv. traspuesta	$2 \times 2 \times 128$	(ver Capítulo 4)	4×4	2×2
Conv. traspuesta	$4 \times 4 \times 64$	(ver Capítulo 4)	4×4	2×2
Conv. traspuesta	$8 \times 8 \times 32$	(ver Capítulo 4)	4×4	2×2
Conv. traspuesta	$16 \times 16 \times 16$	(ver Capítulo 4)	4×4	2×2
Conv. traspuesta	$32 \times 32 \times 5$	Tangente	4×4	2×2

2.4. Esquema de clasificación final

Tras todas las secciones anteriores que describen las técnicas a incorporar en el esquema de clasificación desarrollado en este trabajo, la Figura 2.14 representa de nuevo cómo se integrarán todas ellas para poder colaborar entre sí, ahora con mayor detalle:

1. En primer lugar, la imagen –conjunto de datos– sobre el que entrenar el esquema de clasificación será segmentado en superpíxeles con el algoritmo WP.

2. Una vez concluida la segmentación, se incorporará a la imagen toda la información de referencia disponible para determinar qué superpíxeles contienen píxeles categorizados y, por lo tanto, serán utilizables para el entrenamiento y evaluación del esquema de clasificación. Como es habitual, las muestras que se vean involucradas de algún modo en la etapa de entrenamiento serán excluidas en la posterior fase de prueba.

3. Se extraerá una muestra de cada superpíxel utilizable, y se aplicará un aumentado de datos tradicional a todas aquellas que se vean involucradas en el aprendizaje del esquema, conformando así el subconjunto de muestras de entrenamiento. Este aumentado tradicional apoyará el aumentado basado en aprendizaje profundo que realizará la arquitectura BAGAN. Concretamente, el aumentado se basará en la aplicación de rotaciones, y de volteos verticales y horizontales.

4. El entrenamiento del esquema BAGAN comenzará por el aprendizaje del autoencoder. El ajuste de sus parámetros se regirá por el error cometido al comparar las muestras reales suministradas con las reconstrucciones creadas para las mismas.

5. Una vez concluido el entrenamiento del autoencoder, se transferirán todos los parámetros al submódulo GAN para inicializarlo desde un punto de aprendizaje más estable.

6. Tras ello, se iniciará el aprendizaje del submódulo GAN, durante el cual G apoyará el aprendizaje de D con el suministro de las muestras falsas que genere, a modo de aumentado de datos basado en aprendizaje profundo. Una vez concluido el entrenamiento, D será el clasificador que resulte para el problema de clasificación de teledetección que se afronta.

   El ajuste de parámetros de D se guiará por el objetivo de categorizar adecuadamente las muestras reales, y de asignar la categoría falsa a las muestras de G. Por otra parte, esta última red se guiará por el objetivo de que sus muestras falsas terminen siendo categorizadas como las clases que pretenden imitar, engañando a D.

7. Finalmente, se pondrá a prueba el desempeño de D y, por ende, de todo el esquema de clasificación, mediante su capacidad para generalizar el conocimiento adquirido al clasificar las muestras del subconjunto de prueba.

La combinación de todas estas técnicas permite cubrir los objetivos propuestos inicialmente en el Capítulo 1, acerca de desarrollar un esquema de clasificación para teledetección que:

1. **Explote al máximo la información espectral y espacial disponible en las escasas muestras de aprendizaje:** gracias al uso de una ResNet como clasificador.

2. **Incremente la riqueza de la información de entrenamiento para una mejor generalización del clasificador:**

   - Por una parte, se consigue gracias a la integración del clasificador en una arquitectura BAGAN para que las muestras falsas generadas por G actúen a modo de aumentado de datos basado en aprendizaje profundo.

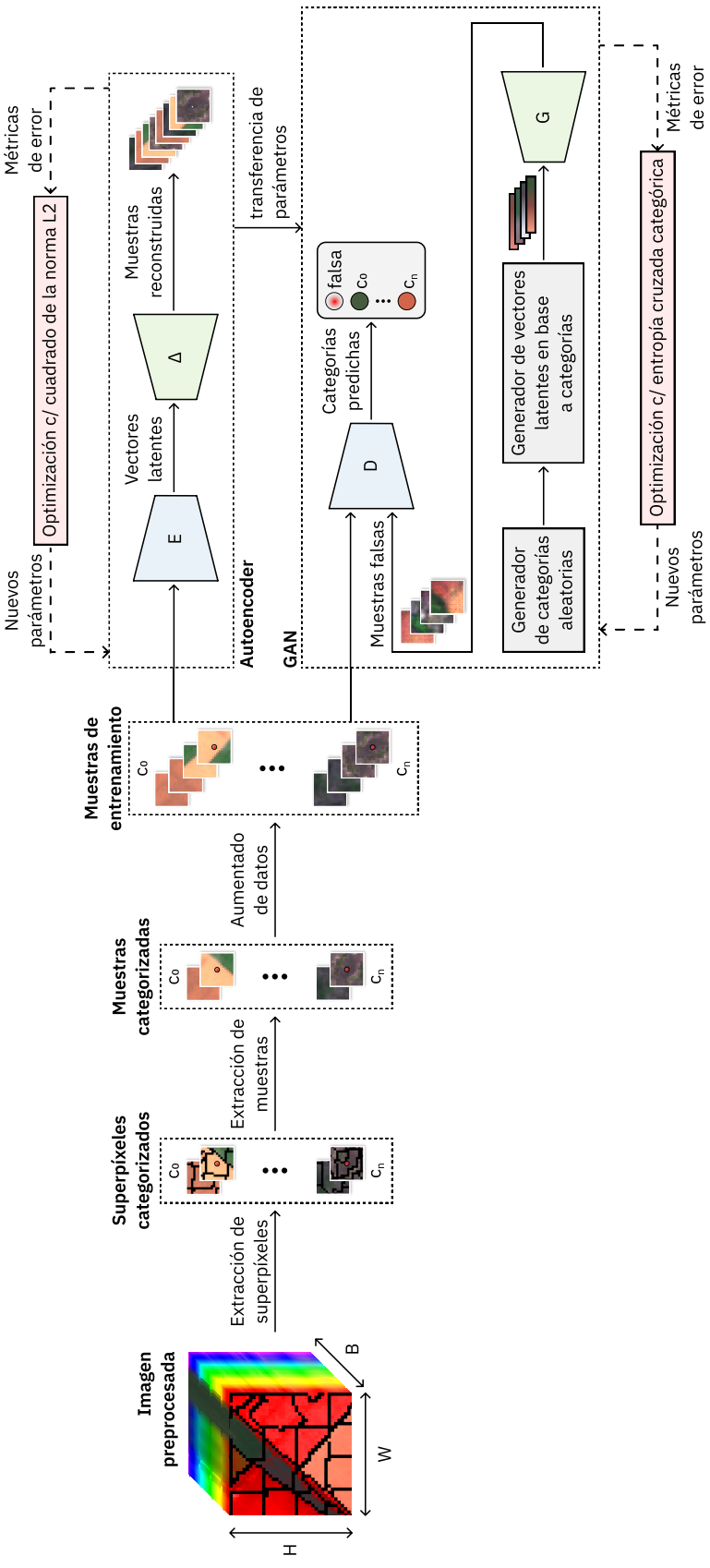


Figura 2.14: Esquema de clasificación de imágenes para teledetección desarrollado en este trabajo. Nótese cómo se combinan técnicas para extraer muestras de entrenamiento de mayor calidad, para extraer características profundas de las mismas, y de aumentado de datos de entrenamiento, todo ello con el objetivo de maximizar el desempeño de la clasificación final de las imágenes de teledetección.

- Por otra parte, también cabe resaltar la contribución de las técnicas de aumentado tradicionales para apoyar el aumentado de G.

3. **Mejore la calidad de las muestras de entrenamiento extraídas:** gracias a que la segmentación en superpíxeles de la imagen guíe la posterior generación de muestras.

3 | 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. En primer lugar, se introducirán los conjuntos de datos con los que evaluar del esquema de clasificación desarrollado, y se concretarán todas las métricas a utilizar para determinar su desempeño. 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.

3.1. Conjuntos de datos experimentales

El esquema de clasificación propuesto será evaluado 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 [58], 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. **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 (*alto* × *ancho*), cubriendo un total de 643,40 × 2256,55 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. **Río Oitavén:** región del río localizada en 42°22′15,48″ N 8°25′47,07″ W. Sus dimensiones son de 6689 × 6722 píxeles, cubriendo un total de 758,30 × 759,41 m.

[1] Aunque pueda parecer que esta cantidad de bandas sea demasiado reducida, al poder llegar a contar las imágenes de teledetección con decenas o incluso cientos de ellas, estas imágenes multiespectrales permitirán obtener observaciones perfectamente válidas dado que: (1) los algoritmos empleados son directamente adaptables a imágenes con un mayor número de bandas, (2) y de hecho en muchos esquemas de clasificación propuestos en la literatura se reduce la dimensionalidad espectral de las imágenes de teledetección a unas pocas bandas, con Principal Component Analysis (PCA) [59] habitualmente, para hacerlas tratables ante técnicas computacionalmente costosas [16], [44], [60].

[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 [61].

En la Figura 3.1 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 procesamiento de imágenes hiperespectrales del Centro Singular de Investigación en Tecnoloxías Intelixentes [62] de la Universidade de Santiago de Compostela. El Cuadro 3.1 recoge las diez 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 empleará WP, indicando una media de 400 píxeles/superpíxel, permitiendo un tamaño mínimo de 100 píxeles/superpíxel, y empleando un factor de compacidad de 0,5 puntos, tomando como referencia [3]. Los parches a extraer de la imagen tendrán unas dimensiones de $H \times W = 32 \times 32$ píxeles; se ha optado por este tamaño puesto que permitirá que las muestras en base a superpíxeles contengan principalmente píxeles de la categoría asociada al segmento, a la vez que se deja cierto margen a capturar información adicional de contexto con píxeles circundantes al segmento. Además, como es habitual, todas las muestras que se extraigan de las imágenes experimentales serán normalizadas para facilitar a las arquitecturas neuronales su aprendizaje. Concretamente, serán normalizadas al rango $[-1, 1]$.

Cuadro 3.1: Categorías identificables en las tres imágenes multiespectrales de cuencas de ríos gallegos. Se indica además el número de muestras identificables por categoría en cada imagen, tanto sin haberlas segmentado, como tras haberlo hecho empleando WP.

			Oitavén		**Ermidas**		**Eiras**	
#	**color**	**Categoría**	segmentos	píxeles	segmentos	píxeles	segmentos	píxeles
1.		Agua	1540	309 248	908	163 930	3558	734 614
2.		Tierra	970	113 324	870	123 416	851	96 935
3.		Piedra	724	79 152	1828	174 088	2928	144 800
4.		Asfalto	202	43 861	3594	737 350	340	85 209
5.		Cemento	770	128 022	126	32 866	177	27 061
6.		Tejado	446	78 785	606	138 678	47	8232
7.		Prado	9404	2 428 482	13 060	3 419 579	5745	744 949
8.		Vegetación nativa	11 287	1 829 360	4560	804 037	13 458	2 077 751
9.		Pino	2111	193 884	1527	184 547	686	95 132
10.		Eucalipto	8384	863 061	8984	1 135 995	75	8451
		Muestras totales:	35 838	6 067 179	36 063	6 914 486	27 865	4 023 134

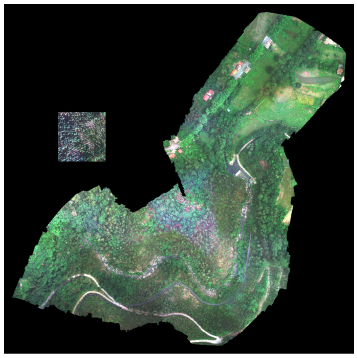
3.2. Métricas experimentales

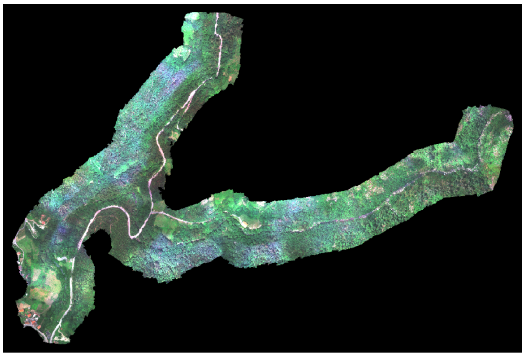
Durante la fase experimental de este trabajo, el desempeño del esquema de clasificación desarrollado se valorará tanto en términos de precisión, como en términos de rendimiento computacional.

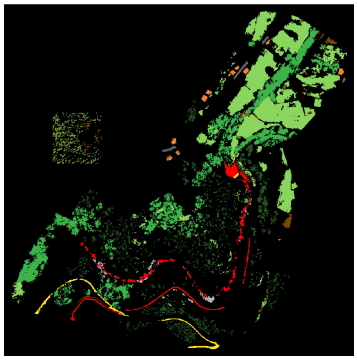
3.2.1. Métricas de precisión

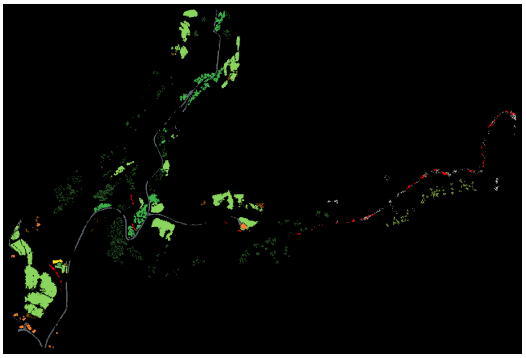
Tras entrenar un clasificador de aprendizaje supervisado, siempre se evalúa su capacidad de generalización suministrándole muestras del problema objetivo a las que no haya sido expuesto. En el caso de la clasificación de imágenes de teledetección, el clasificador deberá predecir la categoría de cada 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.

Para ello, se emplearán las siguientes medidas estándar en el ámbito de la clasificación en teledetección [63]:



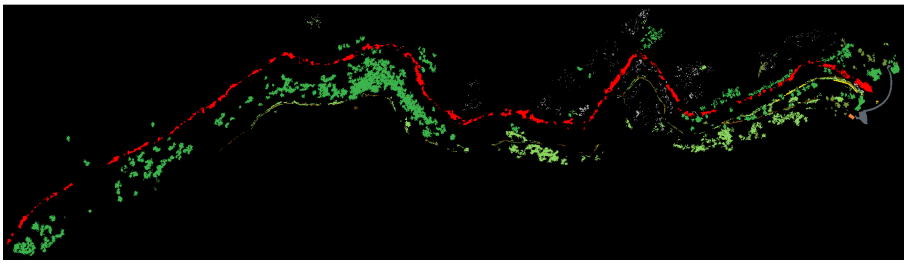






(a) Río Oitavén

(b) Riachuelo Ermidas



(c) Embalse de Eiras

Figura 3.1: Imágenes en color compuesto de las imágenes multiespectrales de cuencas de ríos gallegos, junto con la correspondiente información de referencia de cada una.

- **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 (k):** porcentaje de acierto total en píxeles corregido tras tener en cuenta cuántos aciertos se pueden esperar por mero azar.

3.2.2. Métricas de rendimiento

El entrenamiento y prueba del esquema de clasificación sobre una imagen también será evaluado en cuanto a tiempo de ejecución requerido, y eficiencia implicada. Por conveniencia, se puede dividir la ejecución de un experimento en las siguientes fases:

F1. **Preprocesamiento del conjunto de datos:** abarca desde la lectura de la imagen en disco, hasta la identificación y registro de las muestras disponibles en ella para poder pasar al entrenamiento del esquema de clasificación. También se incluye el tiempo de segmentación en caso de querer procesar la imagen en base a superpíxeles.

F2. **Entrenamiento del esquema de clasificación:** abarca el proceso iterativo en que se suministran repetidamente las muestras de entrenamiento al esquema de clasificación –tanto al submódulo del autoencoder, como al submódulo GAN– para que adquiera conocimiento sobre el problema a resolver.

F3. **Prueba del esquema de clasificación:** abarca el procedimiento en que se suministran todas las muestras de prueba al esquema de clasificación entrenado para evaluar su capacidad final para resolver el problema objetivo.

En concreto, será de interés registrar el tiempo de ejecución requerido por estas tres fases, detallando además el coste de la segmentación en superpíxeles dentro del preprocesamiento, para determinar el impacto de rendimiento al incorporar las diferentes técnicas del esquema de clasificación diseñado. También será de interés determinar 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 $speedup = t_b/t_o$.

3.3. Entorno de pruebas

3.3.1. Configuración de hardware

Serán dos las plataformas de cómputo disponibles para lanzar los experimentos de este trabajo:

P1. El primer computador dispondrá de dos Central Processing Unit (CPU) Intel Xeon Gold 5220 [64], 192 GB de Random Access Memory (RAM), y dos Graphics Processing Unit (GPU) NVIDIA Tesla V100S con 32 GB de Video Random Access Memory (VRAM) [65]. Estas últimas permiten acelerar enormemente la ejecución de arquitecturas neuronales, entrenándolas y evaluándolas utilizando aritmética de coma flotante en simple precisión.

P2. El segundo computador dispondrá de una CPU Intel Core i7-11700K [66], 128 GB de RAM, y una GPU NVIDIA RTX 3080 Ti con 12 GB de VRAM [67]. Al igual que en el anterior caso, este componente permitirá acelerar enormemente la ejecución de las arquitecturas neuronales.

El motivo de emplear dos plataformas de cómputo diferentes es que, mientras que la primera dispone de una mayor capacidad computacional total, se trata de un servidor de cómputo compartido con otros usuarios, sin acceso a recursos exclusivos. Por ende, es posible que los experimentos lanzados a ejecución vean sus tiempos de cómputo alterados en ocasiones por la compartición de recursos con trabajos de otros usuarios. Esto no es deseable para la obtención de unos resultados de rendimiento precisos, lo cual sí es posible en la segunda plataforma al tratarse de un computador de uso exclusivo. Aun así, existe por suerte una primera fase experimental del trabajo en la que se refinará el diseño del esquema de clasificación propuesto, y se podrá tomar partido de la primera plataforma de cómputo al no ser relevantes los tiempos que se midan. Tras ello se dará paso a los experimentos finales con los que evaluar el esquema de clasificación en su estado definitivo, y ya se hará sobre la segunda plataforma de cómputo, que aunque sea algo menos capaz, sí permitirá obtener unas métricas precisas en todos los aspectos.

3.3.2. Configuración de software

En cuanto al entorno de software involucrado, se empleará el sistema operativo Ubuntu 20.04 [68], junto con Python 3.8.13 [69], GNU Compiler Collection (GCC) 9.4.0 [70], y Compute Unified Device Architecture (CUDA) 11.6.2 [71] con soporte para Compute Unified Device Architecture Deep Neural Network (cuDNN) 8.4.0.27 [72]. Esta última característica permitirá acelerar todavía más la ejecución de las arquitecturas neuronales sobre dispositivos GPU.

El lenguaje Python se empleará para prácticamente la totalidad de los códigos desarrollados durante este trabajo. Cabe destacar que el paquete PyTorch 1.12.0 [73] proporcionará la base sobre la cual desarrollar y ejecutar las arquitecturas neuronales. De forma complementaria, el lenguaje C++ [74] se utilizará para desarrollar códigos auxiliares que se comuniquen con los códigos principales, principalmente para acelerar el preprocesamiento de las imágenes experimentales. La implementación de WP que se incorporará al flujo de clasificación se tomará directamente de [3], estando desarrollada sobre el lenguaje C [75].

4 | Mejora del esquema de clasificación diseñado

Este capítulo de la memoria se dedicará a explicar diversas mejoras que aplicar al esquema de clasificación diseñado, antes de proceder al desempeño de su evaluación final. Concretamente, se aplicarán cambios que permitan reducir el uso de recursos computacionales del esquema, y otros ajustes que incrementen la capacidad del esquema para realizar su tarea de clasificación.

4.1. Reducción del tiempo de ejecución

Como ya se explicó anteriormente, los métodos de aprendizaje automático basados en aprendizaje profundo resultan mucho más atractivos que los basados en métodos tradicionales para la clasificación en teledetección, dadas las mayores precisiones que permiten obtener [11]-[13]. Sin embargo, también cabe destacar que el foco de la comunidad científica durante estos años respecto a estos nuevos métodos ha sido, fundamentalmente, el incremento de la precisión de clasificación, mientras que el incremento del uso de recursos computacionales ha quedado relegado a un segundo plano [76]. Este inconveniente no debe ser obviado, puesto que los esquemas de clasificación desarrollados son cada vez más costosos de entrenar y ejecutar, llegando al punto de comprometer al diseñador entre si priorizar la precisión de clasificación, o la rapidez de construcción y puesta en marcha del esquema.

A grandes rasgos, todo esquema de aprendizaje automático pasa durante su vida útil por dos etapas:

E1. La etapa de entrenamiento en que se prepara para afrontar el problema objetivo.

E2. Y la etapa de prueba en que se evalúa su desempeño para resolver el problema[1].

En general, podría decirse que la etapa de entrenamiento del esquema resulta la más costosa de ambas por los siguientes motivos:

M1. El entrenamiento de arquitecturas neuronales consiste en un proceso iterativo que se repite hasta alcanzar, o bien un límite de iteraciones prefijado –épocas de entrenamiento–, o bien una precisión objetivo. En cada época se suministran todas las muestras de entrenamiento a la arquitectura neuronal, y los parámetros de la red se van ajustando poco a poco con un proceso de retropropagación [77] en función del criterio del optimizador, comparando la salida dada por la arquitectura y la salida esperada.

    Como las muestras de entrenamiento del conjunto de datos deben ser procesadas reiteradamente, es de esperar que el coste final de la etapa de entrenamiento sea muy elevado.

[1] Por supuesto, con posibles vistas a trasladar el esquema a aplicaciones prácticas con las que resolver problemas reales, en caso de estar satisfecho con su desempeño final.

M2. Como se introdujo en el Capítulo 2, los hiperparámetros son unos factores que condicionan en gran medida el desempeño de las arquitecturas neuronales, ya que afectan a su comportamiento durante el entrenamiento. Además, a diferencia de parámetros como los pesos y sesgos, no se pueden aprender de los datos del problema, por lo que no queda más alternativa que responsabilizar al diseñador de la arquitectura neuronal de establecer sus valores. La elección de hiperparámetros no es para nada trivial, ya que cada problema concreto a resolver puede requerir unos hiperparámetros diferentes para un buen comportamiento de las arquitecturas neuronales diseñadas. Aunque se puede optar por prefijar valores que habitúan funcionar "bien", para obtener el máximo rendimiento de una arquitectura neuronal es común tener que terminar probando una gran variedad de configuraciones de valores, entrenando y evaluando la arquitectura neuronal repetidamente.

    Dado este condicionante, el ya elevado coste de la etapa de entrenamiento se verá magnificado por toda la cantidad de veces que se tendrá que repetir bajo diferentes configuraciones de hiperparámetros, hasta encontrar aquellos más óptimos de entre los probados.

Ante esta situación, los beneficios de acelerar la etapa de entrenamiento son claros: un menor tiempo de aprendizaje facilitará probar más configuraciones de hiperparámetros en la misma cantidad de tiempo, pudiendo maximizar en consecuencia el desempeño de las arquitecturas neuronales diseñadas. Es por ello que en este trabajo se prestará atención a diversos aspectos de la implementación del esquema de clasificación propuesto, de cara a optimizar su entrenamiento y hacerlo lo más veloz posible. Además, es muy probable que todas o casi todas estas observaciones a nivel de implementación sean trasladables a futuras arquitecturas neuronales con las que se trabaje en el *framework* PyTorch, sirviendo el trabajo de esta sección también para la optimización de futuros trabajos.

De forma simplificada, el entrenamiento del esquema propuesto consiste en el siguiente bucle computacional:

```
for epoch in range(hyperparams["epochs"]):

    for batch_id, (samples, targets, _) in enumerate(data_loader):

        # 1. Train the discriminator on real samples

        real_samples, real_targets = samples.to(networks.device, non_blocking=True),
        ↪  targets.to(networks.device, non_blocking=True)

        # Gradients from the previous batch need to be cleared, as PyTorch
        ↪  accumulates them if
        # not told otherwise
        networks.optimizer_D.zero_grad(set_to_none=True)
        networks.optimizer_G.zero_grad(set_to_none=True)

        # Runs all real samples through the network
        D_output_real, _ = discriminator(real_samples)

        # By how much the network has missed the correct predictions
        D_loss_real = networks.classifier(D_output_real, real_targets)


        # 2. Train the discriminator on fake samples

        # On each batch, 1 / (classes_count + 1) of the samples are fake
        batch_size = real_samples.shape[0]
        fake_samples_count = max(batch_size // dataset.classes_count, 1)

```

```
27          # No conditional noise distribution is taken from the generator
28          noise = torch.randn(
29              (fake_samples_count, hyperparams["latent_size"]), dtype=torch.float32,
                ↪  device=networks.device
30          )
31
32          fake_targets = torch.full(
33              (fake_samples_count,), dataset.classes_count, dtype=torch.int64,
                ↪  device=networks.device
34          )
35
36          random_labels = torch.randint(
37              0, dataset.classes_count, (fake_samples_count,), dtype=torch.int64,
                ↪  device=networks.device
38          )
39
40          # Runs all fake samples through the networks
41          fake_samples     = generator((noise, random_labels))
42          D_output_fake, _ = discriminator(fake_samples)
43
44          # By how much the network has missed the correct predictions
45          D_loss_fake = networks.classifier(D_output_fake, fake_targets)
46
47          # Computes the corresponding gradient for each component of the network
48          D_loss = D_loss_real + D_loss_fake
49          D_loss.backward()
50
51          # And adjusts all weights & biases using those gradients
52          networks.optimizer_D.step()
53
54          # Some metrics
55          epoch_losses_d.append(D_loss.item())
56          epoch_losses_d_real.append(D_loss_real.item())
57          epoch_losses_d_fake.append(D_loss_fake.item())
58
59
60          # 3. Train the generator
61
62          # Gradients from the previous batch need to be cleared, as PyTorch
            ↪  accumulates them if
63          # not told otherwise
64          networks.optimizer_D.zero_grad(set_to_none=True)
65          networks.optimizer_G.zero_grad(set_to_none=True)
66
67          # The generator receives as many samples as the batch size
68
69          # No conditional noise distribution is taken from the generator
70          noise = torch.randn(
71              (batch_size + fake_samples_count, hyperparams["latent_size"]),
                ↪  dtype=torch.float32, device=networks.device
72          )
73
74          random_labels = torch.randint(
75              0, dataset.classes_count, (batch_size + fake_samples_count,),
                ↪  dtype=torch.int64, device=networks.device
76          )
77
78          # Runs all fake samples through the networks
79          fake_samples     = generator((noise, random_labels))
80          D_output_fake, _ = discriminator(fake_samples)
81
82          # By how much the network has missed the correct predictions
83          G_loss = networks.classifier(D_output_fake, random_labels)
84
85          # Computes the corresponding gradient for each component of the network
86          G_loss.backward()
87
```

```
88          # And adjusts all weights & biases using those gradients
89          networks.optimizer_G.step()
90
91          # Some metrics
92          epoch_losses_g.append(G_loss.item())
```

En el anterior fragmento de código se refleja la implementación desarrollada sobre el *framework* PyTorch para entrenar el submódulo GAN de la arquitectura BAGAN incluida en el esquema de clasificación propuesto. Es cierto que dentro de esta arquitectura también se incluye el entrenamiento de un autoencoder, e incluso se entrenará en futuros experimentos aislados el clasificador ResNet sin la arquitectura BAGAN, para determinar en qué medida esta última contribuye a mejorar el desempeño final del esquema. En cualquier caso, las explicaciones de esta sección del documento se limitarán a este fragmento de código por simplicidad, puesto que las implementaciones desarrolladas para entrenar el autoencoder y la ResNet por separado son más simples, y como comparten los mismos fundamentos que el entrenamiento del submódulo GAN, también se podrán ver beneficiados simplemente aplicando el mismo tipo de mejoras que se propongan más adelante.

Volviendo ahora sí al análisis del fragmento de código, se puede ver que el entrenamiento se prolonga durante tantas épocas como se haya especificado, suministrando dentro de cada época las muestras de entrenamiento en lotes –o *batches*– de un tamaño predefinido. El entrenamiento se puede dividir en las siguientes tres fases:

F1. En primer lugar, se suministran todas las muestras reales del lote a D para determinar un error en su desempeño en función de qué muestras ha categorizado correctamente y cuáles no.

F2. A continuación, G genera un lote de muestras falsas que se suministran a D, y se determina otro error en su desempeño en función de qué muestras ha podido detectar como falsas y cuáles no.

Llegado este punto, es posible realizar el ajuste de los parámetros de D mediante retropropagación, en función de los errores cometidos con la iteración actual.

F3. En tercer y último lugar, G genera un segundo lote de muestras falsas que se suministran a D, y se determina el error en el desempeño de G en función de cuántas muestras han conseguido engañar a D y cuántas no.

Este error es suficiente para realizar el ajuste de los parámetros de G mediante retropropagación.

Aunque a nivel conceptual este procedimiento de entrenamiento es relativamente fácil de comprender y de implementar sobre PyTorch, es importante no dejarse engañar. En lenguajes de alto nivel como Python, y empleando además librerías tan complejas como una que dé soporte para la ejecución de arquitecturas neuronales, el nivel de abstracción resultante para el usuario es tal que, por simplicidad, tienen que ocultar o dar por asumidos muchos aspectos de la ejecución de los códigos desarrollados. Para aclarar qué implicaciones puede llegar a tener cada segmento del código de entrenamiento, es útil desgranar más las operaciones que deben realizarse durante el mismo [78]:

1. **Composición de un lote de muestras:** para ello, la CPU debe acceder a los datos de la imagen con la que se trabaja, para extraer determinadas muestras de la misma, y aplicarles las técnicas de aumentado tradicionales pertinentes –por ejemplo, rotaciones–, en caso de haberlas.

2. **Transferencia del lote a la GPU:** dado que las arquitecturas neuronales se ejecutan sobre GPU, es necesario mover los datos del lote de la RAM a este dispositivo.

3. **Ejecución de las arquitecturas neuronales:** para entrenar las arquitecturas es necesario procesar el lote recibido, computar las métricas de error, generar gradientes a partir de estas, y aplicarlos a los parámetros entrenables de las arquitecturas para actualizar sus valores. Este procedimiento es especialmente costoso al entrenar una arquitectura GAN, al involucrar la ejecución de múltiples redes, y la generación de datos auxiliares como vectores latentes.

4. **Registro de métricas:** este paso se intercala en medio del anterior, al ir registrando la CPU las métricas de error que cometen las arquitecturas neuronales, para propósitos de monitorización.

Ahora que se ha desgranado un poco más el bucle de entrenamiento, es posible comenzar a vislumbrar determinados aspectos que pueden llegar a impactar de forma considerable en el rendimiento del código; principalmente, por implicar una interacción entre dos dispositivos, CPU y GPU. En resumen:

- **CPU:** se emplea durante el proceso de entrenamiento para componer los lotes de muestras, para ejecutar operaciones no soportadas por la GPU, y para registrar métricas de entrenamiento.

- **GPU:** se emplea durante el proceso de entrenamiento para procesar las muestras con las arquitecturas neuronales y realizar la optimización de parámetros mediante la retropropagación.

Por norma general, las herramientas con un gran nivel de abstracción para el usuario habitúan poner a disposición de este una variedad de opciones que controlen su comportamiento, y que pueden llegar a tener un impacto apreciable en el rendimiento de los códigos desarrollados sobre ellas. A continuación se analizará el caso concreto de PyTorch, de cara a tratar de optimizar el consumo de recursos computacionales durante el entrenamiento del esquema de clasificación propuesto:

O1. La composición del lote de entrenamiento es un paso relativamente costoso, y que debe realizar necesariamente la CPU. Por defecto, PyTorch emplea un único proceso para el entrenamiento. Consecuentemente, cada vez que se solicita un nuevo lote, este único proceso en ejecución debe realizar todas las operaciones pertinentes para componerlo, paralizando a efectos prácticos el entrenamiento hasta que haya compuesto el lote.

    No obstante, PyTorch ofrece la posibilidad de lanzar más procesos que apoyen al proceso principal, encargándose de preparar en segundo plano los lotes a la vez que se ejecuta el bucle de entrenamiento. Gracias a ello, es posible conseguir que nada más se pida un nuevo lote, este ya esté inmediatamente disponible [79] para no detener el entrenamiento. Es decir, se consigue solapar una gran cantidad de cómputo en CPU con el cómputo en GPU en pos de un mayor rendimiento.

    Esto se puede conseguir configurando el valor del parámetro `num_workers` de la clase `DataLoader` de PyTorch que suple las muestras durante el entrenamiento. Cualquier valor mayor $x \mid x > 0$ indica que se lancen x procesos de soporte al proceso principal.

O2. Aunque ahora los lotes de muestras se preparen de antemano, sigue existiendo el inconveniente de que deben ser transferidos desde la memoria principal del

computador hasta la memoria de la GPU. Por defecto, la transferencia se iniciará una vez se requiera el lote para realizar una nueva iteración del bucle de entrenamiento, por lo cual sigue produciéndose una parada del mismo cada vez que se comienza una iteración.

Las transferencias de memoria entre CPU y GPU pueden llegar a ser relativamente costosas, al tratarse de dispositivos con memorias físicamente diferentes. Para tratar de minimizar el impacto de estas comunicaciones sobre CUDA, es común recurrir al uso de memoria *pinned* y de transferencias asíncronas [80]. La memoria *pinned* se trata de una región de la RAM a la que la GPU puede acceder directamente, acelerando así la velocidad de las transferencias de memoria. Además, usar este tipo de memoria tiene el beneficio añadido de que posibilita lanzar las transferencias de memoria de forma asíncrona. Puesto que se trata de una región accesible directamente por la GPU, no es necesario bloquear a la CPU para que colabore directamente con la GPU para posibilitar la comunicación, sino que es posible hacer que la CPU lance a la GPU la orden de transferencia para que la primera siga realizando otras tareas, y que la GPU complete la transferencia cuando le sea posible. A efectos prácticos, esto permite solapar todavía más el cómputo de los dispositivos; por ejemplo, en el fragmento de código mostrado anteriormente se solapa la transferencia de las muestras con el borrado de los gradientes de la última iteración.

PyTorch facilita el uso de memoria *pinned* a través del argumento `pin_memory = True` de la clase `DataLoader`, mientras que las transferencias de datos a GPU pueden realizarse de forma asíncrona simplemente añadiendo el parámetro `non_blocking = True` a las mismas, e intercalando otras operaciones antes de acceder a los datos que deben transmitirse.

O3. En relación con la transferencia de datos a GPU, durante el entrenamiento se están creando en diversas ocasiones conjuntos de datos auxiliares, como los vectores latentes. A priori, la opción más común es relegar a la CPU la creación de estos datos, y transferirlos posteriormente a la GPU. Por supuesto, esto incurre en una mayor cantidad de sincronizaciones entre ambos dispositivos, que conllevan además costosas transferencias de datos.

Afortunadamente, PyTorch ofrece implementaciones en GPU para las funcionalidades empleadas al inicializar estos datos auxiliares –por ejemplo, la generación de números aleatorios–, de modo que se pueden crear directamente en la memoria de la GPU. Así se facilita todavía más que ambos dispositivos solapen sus cómputos para alcanzar un mayor rendimiento.

Para reservar e inicializar un tensor de datos en PyTorch directamente en la GPU, tan solo es necesario indicarla como dispositivo destino del tensor mediante el argumento `device = ...`.

O4. La ejecución de las arquitecturas neuronales es un proceso computacionalmente intenso, involucrando una gran cantidad de operaciones matriciales que ejecutar en la GPU.

Para tratar de acelerar los cómputos involucrados en cualquier tipo de arquitectura neuronal, las tarjetas gráficas NVIDIA cuentan con soporte nativo para la librería cuDNN. Esta librería contiene una gran variedad de primitivas propias de redes neuronales, que han sido implementadas sobre CUDA y se han optimizado especialmente para su ejecución en este tipo de tarjetas gráficas, con el objetivo de acelerar todo lo posible la ejecución de arquitecturas neuronales sobre ellas.

El *framework* PyTorch cuenta con soporte nativo para esta librería, para lo cual simplemente hay que activar las opciones `torch.backends.cudnn.enabled`

`= True` y `torch.backends.cudnn.benchmark = True` para que, de fondo, las arquitecturas neuronales que se desarrollen se ejecuten internamente aprovechando las funciones especialmente optimizadas de cuDNN.

O5. Más allá de todo lo comentado hasta el momento, el único punto que todavía no se ha tocado son las sincronizaciones entre CPU y GPU que resultan del registro de métricas de entrenamiento; en este caso concreto, sería principalmente la monitorización del error de las redes. El problema radica en que, como las arquitecturas neuronales se ejecutan sobre GPU, cualquier registro de su desempeño implicará una sincronización entre CPU y GPU para obtener las métricas pertinentes.

PyTorch no ofrece una opción para realizar estas transferencias de forma asíncrona, así que la alternativa más directa sería reducir el número de puntos de sincronización durante el entrenamiento. Por supuesto, no es deseable eliminar por completo el registro de las métricas de entrenamiento, al reportar información muy valiosa. En lugar de ello, simplemente podría reducirse la frecuencia con la que se registran, por ejemplo, haciéndolo cada cierto número de iteraciones en lugar de en todas ellas. Así se conseguiría reducir el total de sincronizaciones entre CPU y GPU que se deben realizar a lo largo de toda la etapa de entrenamiento para registrar las métricas.

Para tratar de evaluar el impacto que estas optimizaciones puedan llegar a tener sobre el código desarrollado, se partirá de un código que no haga uso de ninguna de ellas, y se irán incorporando incrementalmente a la vez que se analiza el rendimiento computacional resultante en el bucle de entrenamiento. Para esto, PyTorch ofrece a sus usuarios diversas herramientas de *profiling* [81] con las que se ha construido el Cuadro 4.1.

Cuadro 4.1: Tiempos de cómputo medidos sobre el bucle de entrenamiento del submódulo GAN del esquema de clasificación propuesto, a medida que se incorporan diversas optimizaciones con las que tratar de mejorar su rendimiento. Para evaluar el consumo del bucle en cada nivel de optimización, se registran sus métricas computacionales en repetidas iteraciones, y se terminan reportando los tiempos de ejecución medios. Todas las pruebas se han realizado sobre la segunda plataforma de cómputo presentada en el Apartado 3.3.1, que será la que se emplee para ejecutar los experimentos finales del trabajo.

Optimización	**Tiempo (ms)**							
	Total	Speedup	GPU			DataLoader	CPU	Otros
			kernels	memoria	overhead CPU			
–	159,85	–	63,78	0,49	20,55	7,78	56,70	10,55
O1	155,39	1,02×	63,85	0,45	21,26	2,11	56,99	10,69
O2	151,90	1,05×	63,87	0,50	20,50	0,08	56,49	10,49
O3	149,17	1,07×	63,51	0,50	19,89	0,09	55,14	10,05
O4	63,32	2,52×	13,51	0,71	8,66	0,07	31,79	8,58
O5	68,74	2,32×	13,66	0,70	9,73	0,09	34,69	9,86

Este cuadro desgrana el tiempo de ejecución medido sobre el bucle de entrenamiento con cada paso de optimización, empleando las siguientes categorías:

- **GPU, kernels:** tiempo requerido para ejecutar funciones de cómputo en la GPU.
- **GPU, memoria:** tiempo requerido para gestiones de memoria en la GPU, incluyendo transferencias de datos entre CPU y GPU.

- **GPU, overhead en CPU:** tiempo que la CPU se encuentra ocupada comunicándose con la GPU; por ejemplo, para lanzar kernels a ejecución, o por transferencias de memoria bloqueantes.
- **DataLoader:** tiempo requerido al inicio de cada iteración para obtener el siguiente lote de muestras y poder proseguir.
- **CPU:** tiempo requerido para todas las operaciones que la CPU realiza más allá de la interacción con la GPU.
- **Otros:** tiempos que no se pueden incluir en ninguna de las categorías anteriores.

Dado este nivel de detalle, es posible observar con claridad el impacto esperado para cada una de las mejoras descritas anteriormente, tal que:

O1. **Procesos auxiliares que preparan lotes de muestras en paralelo al bucle de entrenamiento:** se consigue reducir considerablemente el tiempo que el bucle de entrenamiento se queda parado al solicitar el siguiente lote de muestras (columna *DataLoader*). Gracias a ello, se obtiene en torno a un 3 % de aceleración sobre el tiempo total del bucle de entrenamiento.

Concretamente, se han probado configuraciones con 4 y 8 procesos auxiliares, determinando que la primera cantidad es suficiente para maximizar las mejoras de esta optimización –al menos, bajo el entorno de pruebas descrito en el Apartado 3.3–.

O2. **Transferencias CPU–GPU asíncronas sobre memoria *pinned*:** con esta optimización se consigue reducir todavía más el tiempo que el bucle de entrenamiento se queda esperando por el próximo lote de muestras, haciendo que sea prácticamente despreciable. Con ello, se incrementa a un 5 % la aceleración sobre el tiempo total original.

O3. **Inicializaciones de datos en GPU:** las mejoras que se pueden observar al añadir esta optimización son relativamente pequeñas, al no haber una categoría de tiempo que se vea especialmente beneficiada. En lugar de ello, se pueden observar ligeras reducciones en categorías como el tiempo de ejecución de la CPU, que terminan repercutiendo en un 2 % más de aceleración sobre el tiempo de ejecución total inicial.

O4. **Uso de funciones en GPU optimizadas:** esta mejora es, sin duda alguna, la que repercute en mayor medida sobre el tiempo de ejecución del bucle de entrenamiento. Permite reducir en un factor próximo al 75 % el tiempo de ejecución de los kernels en GPU, y en torno a un 50 % el *overhead* que se causa sobre la CPU al lanzar instrucciones a la GPU. Es más, incluso se está reduciendo también el tiempo de cómputo de operaciones sobre CPU, por lo que es posible que PyTorch esté aplicando internamente algún tipo de mejora adicional gracias a recurrir a las funciones de GPU optimizadas. Con todo ello, se consigue incrementar la aceleración sobre el tiempo total de entrenamiento a un considerable 152 %.

O5. **Reducción de la frecuencia de registro de métricas de entrenamiento:** extrañamente, esta optimización ha sido la única de todas que no ha conseguido reducir el tiempo de ejecución del bucle de entrenamiento, terminando incluso aumentándolo ligeramente en las medidas realizadas. No obstante, lo más probable es que el incremento de tiempos se deba más bien a ruido en las mediciones, dado que no tiene sentido que el registrar en menos ocasiones las métricas desemboque en un peor rendimiento. En cualquier caso, como

se está apreciando que registrar frecuentemente las métricas no tiene ningún impacto negativo sobre los tiempos totales, al final se optará por no reducir la frecuencia con la que se toman, para recopilar así una mayor información sobre el proceso de entrenamiento.

Tras aplicar todas estas mejoras al código desarrollado, la Figura 4.1 representa el uso resultante de CPU y GPU a lo largo de una iteración del bucle de entrenamiento. Este tipo de visualización es muy útil para comprobar en qué medida se consigue paralelizar la utilización de estos dos dispositivos de cómputo. En este caso concreto, puede verse cómo la GPU está siendo utilizada en prácticamente todo momento –sección inferior de la figura–, al igual que el proceso principal del entrenamiento explota constantemente el uso de CPU, por lo que se puede concluir que se está consiguiendo solapar de forma satisfactoria el uso de ambos dispositivos para aprovechar las capacidades de cómputo disponibles.

Además, esta línea temporal es todavía más interesante por el hecho de que desgrana en cuanto a operaciones el uso de recursos computacionales, por lo que es posible determinar qué regiones del bucle de entrenamiento son las más costosas. Concretamente, se puede observar cómo casi la totalidad del tiempo del bucle se consume procesando las muestras con las arquitecturas neuronales y, sobre todo, al efectuar el procedimiento de actualización de parámetros con la retropropagación. Respecto al primer ítem, ya se ha tratado de acelerar todo lo posible la ejecución de las redes con el uso de operaciones sobre GPU especialmente optimizadas para aprendizaje profundo. Respecto al segundo ítem, el proceso de retropropagación está controlando íntegramente por PyTorch, así que el rendimiento que se obtiene de él es una consecuencia directa de cómo de optimizada esté para ello este *framework*, no pudiendo hacer nada más al respecto el usuario.

4.2. Optimización de hiperparámetros

Ahora que se ha acelerado considerablemente el entrenamiento del esquema de clasificación desarrollado, es necesario proceder a la etapa de optimización de hiperparámetros antes de evaluar su desempeño final, dado que estará condicionado en gran medida por ellos. El objetivo de esta fase es asignar unos valores concretos a todos los factores que no se han podido concretar en el Capítulo 2, de cara a maximizar el rendimiento de las arquitecturas neuronales contenidas en el esquema para afrontar el problema objetivo de este trabajo.

Para realizar la optimización de hiperparámetros es muy común recurrir a herramientas que automaticen el entrenamiento y prueba de las arquitecturas neuronales bajo una multitud de configuraciones de valores, recopilando y facilitando la comparación de sus desempeños. Esto relega al usuario de una tarea que sería muy tediosa de realizar a mano, y permite aprovechar además lo máximo posible los recursos computacionales disponibles al no detener en ningún momento su uso, lanzando a ejecución una nueva configuración nada más termine otra. Para este trabajo concreto, se ha optado por incorporar la herramienta GuildAI [82] a la implementación desarrollada para el esquema de clasificación propuesto, para que asista en esta fase de optimización de hiperparámetros. Se ha optado por esta herramienta dada su facilidad para integrarla en cualquier código en Python ya desarrollado, resultando mucho menos intrusiva que alternativas más populares como [83]-[85].

Retomando las explicaciones del Capítulo 2, los hiperparámetros a optimizar en el esquema de clasificación serían:

- **La función de activación a aplicar tras las capas convolucionales y densas:** se probarán las funciones ELU, LeakyReLU, y PReLU, al ser las más

Figura 4.1: Línea temporal que representa el uso medido de CPU y GPU en una iteración del bucle de entrenamiento implementado sobre PyTorch para el submódulo GAN del esquema de clasificación diseñado. Cabe resaltar que se han aplicado todas las optimizaciones de rendimiento descritas en esta sección del documento. Nótese cómo se consigue solapar constantemente el cómputo en CPU y GPU, en pos de un mayor rendimiento.

empleadas en la actualidad por sus mejores desempeños frente a funciones de activación más antiguas, como ReLU.

- **La dimensionalidad de los vectores latentes de la arquitectura BAGAN:** basándose en las dimensionalidades con las que trabajan las diferentes capas convolucionales de las redes del esquema de clasificación, se probarán los valores $Z = \{32, 64, 128\}$.
- **La probabilidad de desactivación en las capas de *dropout*:** al no tener ninguna referencia para escoger sus valores, a diferencia de los demás hiperparámetros, se tratará de cubrir un rango muy amplio de posibles valores, evaluando las configuraciones con $p_{dropout} = \{0{,}05 \dots 0{,}50\}$, con un paso de 0,15.

Adicionalmente, en todo entrenamiento de una arquitectura neuronal es necesario fijar otra serie de factores que se encuentran relacionados con el ajuste de los parámetros a aprender. En cierto sentido, también pueden considerarse hiperparámetros, ya que afectan de forma considerable al desempeño final de la arquitectura, y no existe un valor óptimo para ellos, sino que dependen de la estructura de la arquitectura y del problema objetivo concretos:

- **El ratio de aprendizaje, α:** controla la medida en que se actualizan los parámetros de las redes durante la retroprogapación. Un valor muy alto acelerará el proceso de entrenamiento, pero se corre el riesgo de que las arquitecturas no sean capaces de estabilizarse en un óptimo local, o que directamente se desvíen del mismo. Por el contrario, un valor muy bajo corre el riesgo de que el entrenamiento sea demasiado lento.

  En las arquitecturas GAN es común probar ratios de aprendizaje que vayan desde valores tan reducidos como 0,0001, hasta valores tan elevados como 0,01. En este caso concreto, se probarán los valores $\alpha = \{0{,}0005 \dots 0{,}0030\}$, con un paso de 0,0005.

- **El tamaño de los lotes de muestras:** este valor está muy relacionado con α, al ser común emplear mayores ratios de aprendizaje cuantas más muestras se suministren a la vez a una arquitectura neuronal, al influir esta cantidad directamente en la magnitud de los gradientes resultantes.

  Por norma general, lotes de mayor tamaño aceleran el entrenamiento de la red, tanto en cuanto a aprendizaje, como en cuanto a velocidad de ejecución por saturar mejor las capacidades de cómputo del computador. Por desgracia, la capacidad de generalización de las redes neuronales también suele degradarse con mayores lotes. Por ello, se probarán los valores relativamente moderados $batch_{size} = \{32, 64, 128\}$.

Teniendo en cuenta todas las posibles combinaciones de estos valores, es necesario lanzar y evaluar 648 configuraciones diferentes de hiperparámetros. Dado que el coste de esta fase será muy elevado, se empleará la primera plataforma de cómputo descrita en el Apartado 3.3.1, dado que GuildAI ofrece soporte para procesar en paralelo dos configuraciones si se dispone de dos GPUs. Aún así, para que esta fase sea computacionalmente abarcable, también se optará por reducir el número de épocas de entrenamiento en todas las configuraciones a tan solo 20 iteraciones; puede que este valor sea relativamente pequeño, pero también es cierto que la mayor evolución de los métodos basados en aprendizaje profundo suele producirse en las iteraciones iniciales del entrenamiento, de modo que dar margen a 20 épocas debería ser suficiente para discernir con suficiente confianza aquellas configuraciones de

hiperparámetros que terminasen resultando más prometedoras al final, de aquellas que no.

Una vez concluida la ejecución de todas estas configuraciones, los resultados de la optimización de hiperparámetros son los que se representan en la Figura 4.2. En este tipo de representación se traza una línea por cada configuración de hiperparámetros probada, recorriendo en cada columna los valores que le hayan sido asignados y desempeños resultantes. Su utilidad reside en que permite apreciar a simple vista determinadas tendencias en función de los valores configurados, que serían muy complicadas de intuir de otro modo dada la gran dimensionalidad del espacio de búsqueda de valores.

Precisamente para tratar de identificar similitudes entre aquellas configuraciones más prometedoras, se limitará la representación a aquellos experimentos que reporten:

- **Mejores precisiones de clasificación en la etapa de prueba:** determinadas por las métricas OA, AA, y k.
- **Menores errores en el desempeño de las redes:** aunque a priori puede entenderse como "configuración más óptima" aquella que reduzca tanto el error total cometido por D como el cometido por G, es necesario tener presente que estas dos redes se entrenan de forma adversaria. Consecuentemente, la "configuración más óptima" será aquella en la que:
  - G alcance el mejor desempeño posible, de modo que esté generando muestras lo más realistas posible para dar lugar a un buen aumentado de datos.
  - D cometa en torno a un 50 % de fallos sobre las muestras falsas de G, al tratarse del punto de equilibrio en un entrenamiento de una arquitectura GAN, tal y como se explicó en el Apartado 2.3.1.

Esto es precisamente lo que se refleja en la Figura 4.3. En esta se puede comenzar a apreciar cómo ciertas configuraciones de hiperparámetros terminan reportando normalmente aquellos esquemas de clasificación con mejor desempeño. En concreto, los valores que aparentan tener más probabilidades de dar lugar a buenas ejecuciones son:

- La función de activación LeakyReLU.
- Lotes de 32 elementos.
- Vectores latentes de tamaño $Z = 128$.
- Ratios de aprendizaje en torno a 0,001.
- Un 5 % de probabilidad de desactivación de componentes por las capas de *dropout*.

Así pues, se fijarán los hiperparámetros a estos valores de cara a la evaluación final de esquema de clasificación propuesto.

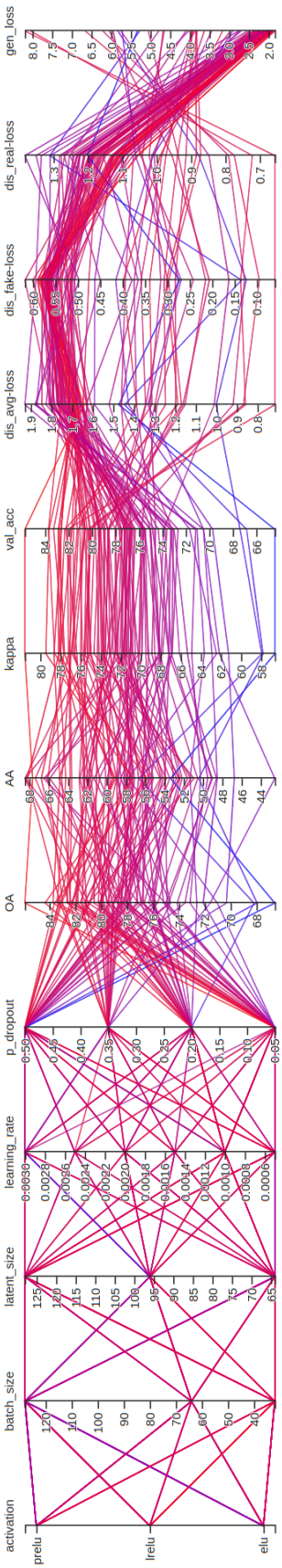


Figura 4.2: Resultados de la optimización de hiperparámetros sobre el esquema de clasificación desarrollado. Se han probado y evaluado un total de 648 configuraciones.

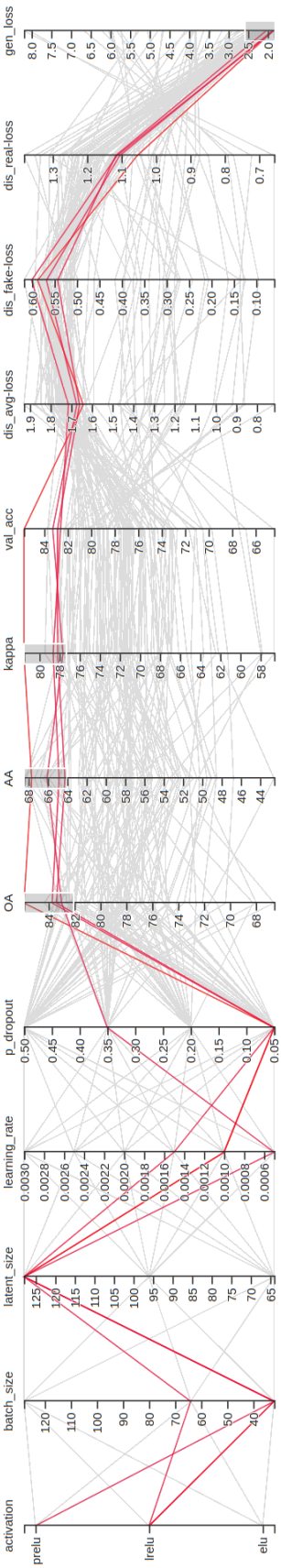


Figura 4.3: Resultados de la optimización de hiperparámetros sobre el esquema de clasificación desarrollado. Se resaltan aquellas configuraciones de las 648 probadas que consiguen dar lugar a esquemas de clasificación con el mejor desempeño.

5 | Pruebas

Este capítulo presentará y desarrollará el plan experimental que permitirá comprobar el desempeño del esquema de clasificación propuesto en este trabajo. En primer lugar, será necesario concretar los experimentos en cuestión a realizar, y tras ello se podrán recopilar y analizar todos los resultados experimentales que se puedan extraer de los mismos.

5.1. Plan experimental

Para evaluar con detenimiento el desempeño del esquema de clasificación propuesto en este trabajo, así como la contribución de cada técnica adoptada, se llevará a cabo una serie de experimentos en los que:

- Se evaluarán las arquitecturas neuronales en dos configuraciones diferentes:

  1. En la primera configuración se empleará la red ResNet de forma autónoma, sin ningún tipo de aumentado de datos.
  2. En la segunda configuración se integrará esta ResNet en la arquitectura BAGAN descrita anteriormente, además de aplicar las técnicas de aumentado de datos tradicionales para apoyar el funcionamiento de esta última. Por simplicidad, se empleará el término Residual Balancing Generative Adversarial Networks (ResGAN) para referirse a esta configuración.

  El objetivo de emplear estas dos configuraciones es determinar la medida en que la aplicación de técnicas de aumentado de datos contribuyen a mejorar el desempeño final del esquema de clasificación desarrollado.

  De forma adicional, se incorporará a la evaluación la sencilla arquitectura CNN que se detalla en el Cuadro 5.1, trabajando de forma autónoma sin ningún aumentado de datos. Con ello se pretende poder determinar también la medida en que una arquitectura profunda como la ResNet propuesta contribuye a mejorar el desempeño global del esquema, frente a optar por un clasificador más simple.

- Cada una de las tres anteriores configuraciones será entrenada y evaluada sobre las tres imágenes descritas en el Apartado 3.1, tanto procesándolas en base a píxeles, como en base a superpíxeles. Con ello se pretende poder determinar el impacto que la segmentación en superpíxeles pueda tener en la calidad de las muestras extraídas de la imagen, tanto en el esquema de clasificación propuesto, como en las otras dos configuraciones adicionales por completitud.

  En un experimento con segmentación en superpíxeles se suministrarán al esquema de clasificación el 15 % de las muestras disponibles; en su versión análoga sin segmentación 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. 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 durante su entrenamiento, y tratar de identificar fenómenos como el sobreajuste –*overfitting*– o el *underfitting*. Finalmente, todas las muestras restantes se incorporarán al subconjunto de prueba sobre el que se evaluará la capacidad de generalización del esquema una vez haya sido entrenado.

Cuadro 5.1: Detalles de la arquitectura de la CNN simple con la que comparar el desempeño de la ResNet desarrollada. Las dimensiones de salida son las correspondientes a una entrada con dimensiones $H \times W \times B = 32 \times 32 \times B$. Antes de cada convolución se aplica un relleno en las dimensiones H y W de la información, para preservar el tamaño de las mismas. También se aplica una capa de *dropout* a toda capa convolucional y a la primera capa densa, con una probabilidad de desactivación $p_{dropout} = 0{,}05$.

Componente	**Dim. salida**	**Activación**	**Tam. filtro**	**Desplazamiento**
Convolución	$16 \times 16 \times 32$	LeakyReLU	3×3	2×2
Convolución	$8 \times 8 \times 64$	LeakyReLU	3×3	2×2
Convolución	$4 \times 4 \times Z$	LeakyReLU	3×3	2×2
Aplanamiento	$(Z \cdot 4 \cdot 4) \times 1$	–	–	–
Capa densa	$Z \times 1$	LeakyReLU	–	–
Capa densa	$C \times 1$	Softmax	–	–

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 arquitecturas neuronales, se repetirán todas las pruebas un total de tres 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.

5.2. Sobre el ajuste de los parámetros neuronales

A lo largo de los años, se han desarrollado una variedad de algoritmos de optimización para entrenar arquitecturas neuronales, y que habitúan seguir la filosofía del descenso del gradiente. El ejemplo más notable de ello es el descenso del gradiente estocástico [86]. Sin embargo, en los últimos años Adaptive Moment Estimation (Adam) [87] ha resultado ser una opción muy popular dentro del campo del aprendizaje profundo, sobre todo para entrenar arquitecturas 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 [87], y por ello se empleará durante el entrenamiento de todas las

arquitecturas neuronales en este trabajo, con parámetros $\beta_1 = 0{,}5$ y $\beta_2 = 0{,}999$, al ser lo habitual en arquitecturas GAN [88].

Por supuesto, es necesario concretar en algún momento de qué valores parten los parámetros –pesos y sesgos– a ajustar en las diferentes arquitecturas neuronales. Un primer enfoque para ello habituaba ser la simple inicialización aleatoria de los valores de los parámetros, pero esto comenzó a dificultar el entrenamiento de redes que ya comenzasen a contar con múltiples capas [89]. Es por ello que surgieron nuevos métodos de inicialización como el propuesto en [89], comúnmente denominado como Inicialización de Xavier –o de Glorot–, y que será al que se recurra en estos experimentos.

Una vez determinadas todas estas opciones, es necesario concretar en último lugar el número de épocas durante las que serán entrenadas las diferentes arquitecturas. De forma resumida, se ha experimentado con una variedad de opciones para este factor, monitorizando la evolución del error cometido durante todo el entrenamiento por las redes del esquema de clasificación propuesto. A raíz de estas pruebas, se ha comprobado cómo la arquitectura de D es más veloz de entrenarse, siendo G la arquitectura que requiere más épocas de entrenamiento para alcanzar un punto más estable. En concreto, se ha observado que dotar al esquema de clasificación de unas 600 épocas de entrenamiento es suficiente para permitir tanto a D como a G alcanzar sus máximos desempeños. Al resto de arquitecturas a entrenar en estos experimentos –el autoencoder, y la CNN simple y ResNet de forma autónoma ambas– también se les permitirá aprender durante 600 épocas para ofrecer unas condiciones justas.

5.3. Resultados experimentales

El Cuadro 5.2 refleja los resultados experimentales obtenidos para los experimentos descritos anteriormente.

Cuadro 5.2: Resultados experimentales para el esquema de clasificación propuesto sobre las tres imágenes multiespectrales presentadas en la Apartado 3.1. El *speedup* para un tiempo dado se calcula pasando de realizar la tarea con muestras en base a píxeles, a muestras en base a superpíxeles.

Imagen		**CNN**		**ResNet**		**ResGAN**	
		píxeles	superpíxeles	píxeles	superpíxeles	píxeles	superpíxeles
Embalse de Eiras	OA (%)	89.47	90.48	96.65	97.50	96.12	97.62
	AA (%)	61.96	60.09	86.26	86.57	81.24	88.54
	k (%)	84.03	83.60	94.93	95.74	94.14	95.95
	tiempo entrenamiento (s)	1576.35	1634.58	1749.56	1986.68	4812.09	4928.38
	speedup entrenamiento	0.96×		0.88×		0.98×	
	tiempo prueba (s)	237.60	5.14	664.13	6.52	670.88	6.56
	speedup prueba	46.25×		101.92×		102.33×	
Riachuelo Ermidas	OA (%)	90.73	94.34	97.49	98.50	97.66	98.76
	AA (%)	79.59	83.76	93.72	94.92	94.07	96.62
	k (%)	86.75	89.64	96.41	97.26	96.65	97.74
	tiempo entrenamiento (s)	2186.10	2243.33	2548.16	2521.90	6742.24	6417.81
	speedup entrenamiento	0.97×		1.01×		1.05×	
	tiempo prueba (s)	410.80	6.92	1135.66	8.32	1163.77	8.64
	speedup prueba	59.34×		136.50×		134.73×	
Río Oitavén	OA (%)	84.66	90.49	94.35	96.63	95.55	95.85
	AA (%)	74.27	78.99	91.50	91.94	91.81	93.58
	k (%)	78.75	84.85	92.60	94.64	93.84	93.47
	tiempo entrenamiento (s)	1416.59	1466.48	2508.46	2362.02	6254.52	6068.06
	speedup entrenamiento	0.97×		1.06×		1.03×	
	tiempo prueba (s)	354.39	4.97	1015.80	7.91	987.02	8.51
	speedup prueba	71.26×		128.36×		115.94×	

Para simplificar el análisis de los resultados experimentales, se puede prestar atención en primer lugar a las métricas relativas a la precisión del esquema de clasificación entrenado, y a continuación las métricas de rendimiento asociadas.

5.3.1. Análisis de las métricas de precisión

Pasar de utilizar un clasificador basado en una arquitectura neuronal simple como una sencilla CNN, a emplear una arquitectura profunda como la ResNet propuesta en este trabajo, tiene un gran impacto en las precisiones que un esquema de clasificación para teledetección puede alcanzar. En este caso concreto, se pueden observar mejoras muy significativas en las tres métricas de precisión al pasar de la primera a la segunda columna del cuadro. Esto se encuentra en la línea de lo esperable, puesto que las CNN profundas son capaces de explotar mejor la información espectral y espacial de las muestras analizas para terminar alcanzando mejores desempeños de clasificación que redes más simples.

Por otra parte, la tercera columna del cuadro reflejaría el impacto de incorporar las técnicas de aumentado de datos, tanto tradicionales como a través de una arquitectura GAN, al clasificador profundo. Aunque lo esperable sería observar también una considerable mejora de las precisiones de clasificación, no está siendo el caso exactamente.

En general, se puede ver cierta fluctuación al comparar las precisiones de un experimento con ResNet frente al mismo experimento con la ResGAN. Existen ocasiones en las que las precisiones son ligeramente superiores con esta última configuración, y ocasiones es las que son ligeramente inferiores.

De todos modos, la mayor parte de estas diferencias de precisiones son muy pequeñas, al no superar normalmente el 1 % de variación. Consecuentemente, es muy probable que la mayor parte de estos sutiles cambios no tengan un motivo de peso detrás, sino que simplemente sean la manifestación de "ruido" en las mediciones por la aleatoriedad inherente a los experimentos. Es decir, que si se realizasen más repeticiones de los experimentos, sería esperable que a la larga se acabasen asemejando cada vez más la gran parte de métricas de precisión de estas dos configuraciones.

Ahora bien, en cualquier caso cabe resaltar que sí hay cuatro mediciones concretas del esquema ResGAN que se distancian de forma considerable de las precisiones de la ResNet:

- La métrica AA sufre un importante empeoramiento en el caso de entrenar sobre la imagen *Embalse de Eiras* a nivel de píxel. El motivo detrás de ello es incierto, puesto que es algo que no se observa para la métrica en las otras dos imágenes, así que puede que simplemente haya sido una cuestión de coincidencia en las veces que se ha repetido este experimento, por la aleatoriedad inherente a las pruebas.

- La métrica AA se ve considerablemente mejorada en el caso de entrenar con muestras a nivel de superpíxeles, independientemente de la imagen empleada. Es decir, las técnicas de aumentado de datos no han fracasado al final en el intento de mejorar las precisiones del esquema de clasificación, sino que han conseguido incrementar una de las tres métricas de precisión empleadas.

  Para entender mejor el porqué, se puede analizar la matriz de confusión de cualquier experimento concreto al pasar de emplear la ResNet a emplear la ResGAN. Por ejemplo, el Cuadro 5.3 refleja la matriz de confusión para una de las repeticiones del experimento de ResNet sobre la imagen *Río Oitavén*, mientras que el Cuadro 5.4 lo hace para la ResGAN. En este tipo de representación, cada celda del cuadro indica cuántas muestras de prueba de la categoría dada

por la fila han sido categorizadas como la clase dada por la columna; cuantos más elementos estén presentes en la diagonal, más aciertos habrá conseguido el clasificador.

Si se analiza el cambio de precisiones al pasar a emplear la ResGAN, se puede ver cómo los incrementos más grandes se dan sobre aquellas clases con menos muestras, –acorde al Cuadro 3.1 en la página 32–; por ejemplo, las clases *Tierra*, *Cemento*, y *Tejado* sufren aumentos de precisión de unos 13, 5, y 3 puntos respectivamente, encontrándose las mejoras de las demás clases en torno a 1 o 2 puntos a lo sumo. Esto es coherente, puesto que lo que más sentido tiene es que las técnicas de aumentado de datos estén contribuyendo a mejorar sobre todo el rendimiento en clases poco representadas, porque las demás categorías ya cuentan con muchas más muestras reales como para que el clasificador pueda extraer suficiente conocimiento de ellas, sin tener que depender del aumentado para modelarlas apropiadamente.

Por otra parte, las métricas OA y k consideran todos los píxeles de prueba de forma conjunta, mientras que la métrica AA dota del mismo peso a todas las categorías, al realizarse una media entre sus precisiones. Una consecuencia de ello es que los malos desempeños en categorías con pocas muestras tienden a quedar eclipsados a no ser que se preste atención a la métrica AA. Por lo tanto, es perfectamente coherente que estos experimentos reflejen la mejora de precisión principalmente a través de la mejora de la métrica AA, puesto que el principal beneficio del aumentado de datos es aumentar el rendimiento del esquema sobre aquellas clases con menos muestras de referencia disponibles.

Por poner un ejemplo concreto, la Figura 5.1 representa una serie de muestras falsas creadas por G como parte del aumentado de datos en uno de los experimentos realizados. Es interesante resaltar cómo el esquema BAGAN está consiguiendo modelar adecuadamente la distribución de datos de todas las categorías del problema de clasificación, pudiendo generar en consecuencia una variedad de muestras falsas para cualquier categoría bajo demanda.

Tocando en último lugar el impacto de incorporar la segmentación en superpíxeles al flujo de clasificación, puede observarse una tendencia global de que entrenar un clasificador con muestras de superpíxeles termina repercutiendo, casi siempre, en una mejora de las métricas de precisión. Existen casos en los que la mejora es sutil, estando por debajo de 1 punto, pero también existen casos en los que algunas métricas se ven mejoradas en incluso 2 o 3 puntos. En definitiva, se podría decir que hacer que la segmentación en superpíxeles guíe la extracción de píxeles permite mejorar, tal y como se pretendía, la calidad de la información suministrada al clasificador durante el entrenamiento para que aprenda a realizar mejor su tarea.

En relación con los superpíxeles, es interesante destacar también el hecho de que son necesarios para permitir que la ResGAN termine mejorando las precisiones de la ResNet. El motivo detrás de ello posiblemente sea que a la arquitectura GAN le resulte complicado aprender a modelar las diferentes clases del problema al estar recibiendo muestras en base a píxeles, que tienen muchas más probabilidades de congregar una alta cantidad de píxeles diferentes a los de la categoría objetivo. Por ello, el extraer muestras en base a superpíxeles –muestras que contengan fundamentalmente píxeles de la categoría objetivo– eliminaría esta complejidad, permitiendo a G modelar con más facilidad la distribución de datos en cada categoría.

5.3.2. Análisis de las métricas de rendimiento

En general, exceptuando el caso atípico del entrenamiento de la ResNet sobre la imagen *Embalse de Eiras*, la incorporación de superpíxeles al flujo de clasificación no

Cuadro 5.3: Matriz de confusión resultante para una de las repeticiones del experimento de ResNet sobre la imagen *Río Oitavén* trabajando con superpíxeles.

Categoría	**Agua**	**Tierra**	**Piedra**	**Asfalto**	**Cemento**	**Tejado**	**Prado**	**Veg. nativa**	**Pino**	**Eucalipto**
Agua	299 364	246	4959	919	2097	1	325	557	250	530
Tierra	281	86 348	3808	608	4399	2679	12 786	664	1685	66
Piedra	4809	1133	65 262	0	1554	4	5104	952	8	326
Asfalto	1444	265	37	41 596	229	235	1	37	0	17
Cemento	8585	796	1321	831	115 377	23	579	152	137	221
Tejado	66	3307	356	82	345	72 668	1678	270	1	12
Prado	972	8080	12 501	0	1956	568	2 334 841	56 955	966	11 643
Veg. nativa	1557	1190	524	202	302	202	38 964	1 750 366	805	35 248
Pino	2	451	70	0	0	0	1	749	192 412	199
Eucalipto	1001	160	636	0	275	0	13 252	64 763	267	782 707
Precisión categoría (%):	96.80	76.20	82.45	94.84	90.12	92.24	96.14	95.68	99.24	90.69

Cuadro 5.4: Matriz de confusión resultante para una de las repeticiones del experimento de ResGAN sobre la imagen *Río Oitavén* trabajando con superpíxeles.

Categoría	**Agua**	**Tierra**	**Piedra**	**Asfalto**	**Cemento**	**Tejado**	**Prado**	**Veg. nativa**	**Pino**	**Eucalipto**
Agua	108 450	0	309	0	1570	0	192	40	0	35
Tierra	733	41 532	5	0	0	0	3819	481	6	17
Piedra	1905	28	20 263	0	278	0	1845	29	0	392
Asfalto	977	0	0	15 756	365	0	2	2	0	0
Cemento	2152	72	14	0	52 409	0	135	79	0	26
Tejado	5	1508	0	182	2	46 965	344	175	0	0
Prado	102	1505	656	0	0	0	1 803 557	16 259	3	2896
Veg. nativa	108	241	102	0	105	1	17 487	1 023 398	0	21 673
Pino	0	483	0	0	0	0	0	8	84 564	14
Eucalipto	1	0	0	0	0	0	516	3083	0	216 508
Precisión categoría (%):	98.06	89.14	81.90	92.13	95.49	95.49	98.83	96.26	99.41	98.36

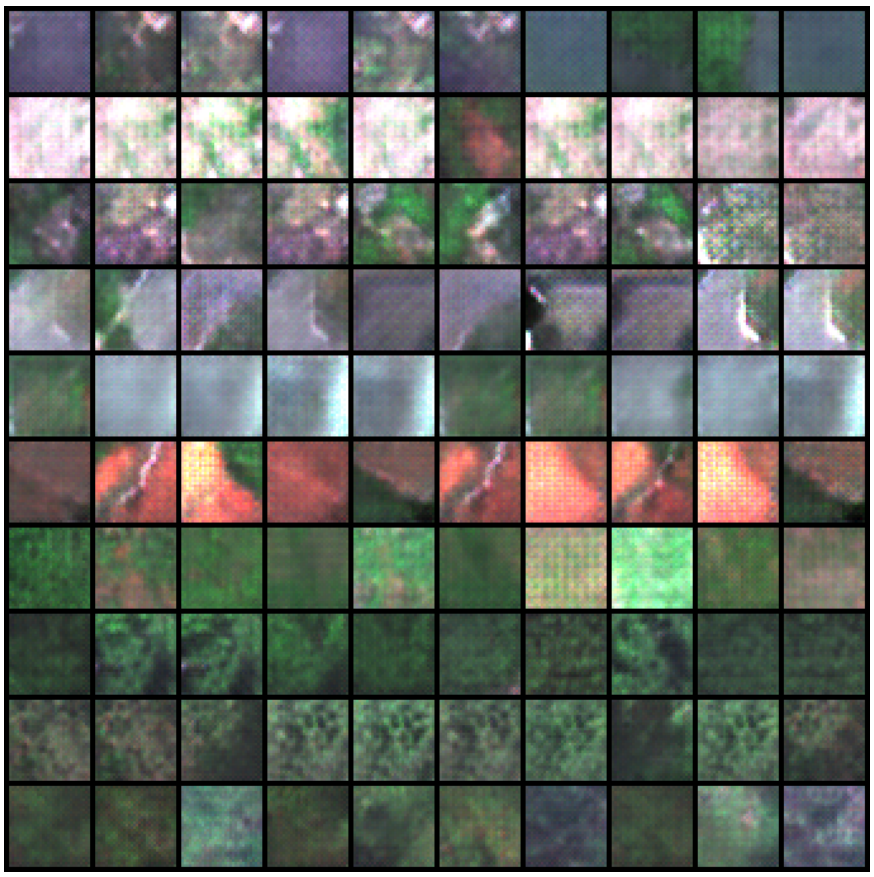


Figura 5.1: Muestras falsas generadas por G en uno de los experimentos realizados sobre la imagen *Riachuelo de Ermidas* trabajando con superpíxeles. Cada fila se corresponde con una de las categorías de muestras disponibles en la imagen, siguiendo el orden empleando ya en otras ocasiones a lo largo de este trabajo. Cabe destacar la capacidad de G para generar una variedad de muestras falsas para las diferentes categorías del problema, representando su capacidad para modelar adecuadamente la distribución de datos del problema, incluso para aquellas categorías con escasa información de referencia.

tiene ni un impacto negativo ni un impacto positivo en el tiempo de entrenamiento. Las ligeras variaciones que se pueden observar posiblemente sean ruido introducido por otras tareas que consumiesen algunos recursos del computador de pruebas, y el caso atípico mencionado posiblemente no sea más que ruido magnificado. Todo esto es coherente, puesto que la extracción de muestras en base a píxeles y en base a superpíxeles realizan prácticamente las mismas operaciones, por lo que ninguna debería dar lugar a una ralentización del bucle de entrenamiento, al estar suministrándose además el mismo número absoluto de muestras en cada pareja de experimentos píxel–superpíxel.

Por otra parte, sí se puede observar cómo la clasificación final de las muestras de prueba se ve enormemente acelerada al realizarla en base a superpíxeles. Esto es perfectamente esperable, tal y como se explicó en el Apartado 2.1.2. Teniendo ahora los superpíxeles un tamaño medio de 400 píxeles, se pasa a realizar una sola predicción por cientos de predicciones originales a nivel de píxel, dando lugar a aceleraciones que superan incluso el orden de 100× bajo determinados experimentos.

Finalmente, también sería interesante determinar el coste a pagar por incorporar la segmentación en superpíxeles al flujo de clasificación, tal y como se comentó en la Apartado 3.2.2. Concretamente, además de tomar los tiempos de entrenamiento y prueba de las arquitecturas neuronales, también se registraría el tiempo requerido para preprocesar las imágenes de cara al flujo de clasificación, incluyendo en este el tiempo de segmentación en superpíxeles.

El Cuadro 5.5 refleja estos costes, desglosados por cada imagen. Como se puede observar, se está incurriendo en notables penalizaciones sobre los tiempos de preprocesamiento de la imagen al segmentarla en superpíxeles. En cualquier caso, la segmentación sigue siendo realmente interesante porque es una apuesta segura para incrementar las precisiones de clasificación, y realmente el sobrecoste de tiempo a pagar en el preprocesamiento es relativamente despreciable en comparación a los tiempos de entrenamiento tan elevados de las arquitecturas neuronales. Al final, no resulta realmente problemático, en cuanto a consumo de recursos computacionales se refiere, el incorporar la segmentación en superpíxeles al flujo de clasificación en teledetección, considerando todos los beneficios que puede aportar.

Cuadro 5.5: Coste de incorporar la segmentación con WP 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. Cabe resaltar que el esquema de clasificación empleado no influye en el preprocesamiento de una imagen.

	Eiras		**Ermidas**		**Oitavén**	
	píxeles	superpíxeles	píxeles	superpíxeles	píxeles	superpíxeles
tiempo segmentación (s)	0.00	25.49	0.00	59.84	0.00	12.00
tiempo preprocesamiento (s)	13.31	33.10	24.19	73.67	14.49	17.57
speedup preprocesamiento	0.40×		0.33×		0.82×	

6 | Conclusiones y trabajo futuro

En este Trabajo de Fin de Máster se ha diseñado y puesto a prueba un esquema de clasificación de imágenes de teledetección basado en aprendizaje profundo. El objetivo propuesto era integrar diferentes técnicas para clasificación que se habitúan emplear por separado en la literatura, todo ello con el objetivo de minimizar el impacto de las restricciones de escasez de muestras y desbalance de clases que se suelen dar en los conjuntos de datos de teledetección, para no limitar el desempeño del clasificador neuronal.

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 como etapa de preprocesamiento en el flujo de clasificación, para extraer muestras en base a superpíxeles en lugar de en base a píxeles, consigue mejorar la calidad de la información suplida al esquema de aprendizaje automático, repercutiendo en una mejora de su capacidad de clasificación final.

2. Se ha optado por diseñar una arquitectura neuronal profunda como clasificador del esquema, comprobando cómo las CNN residuales son capaces de aprovechar mejor la información espectral y espacial de las escasas muestras de aprendizaje, y terminar realizando así mejores clasificaciones que redes CNN más simples.

3. Para apoyar el aprendizaje del clasificador, se han aplicado diferentes técnicas de aumentado de datos con la intención de incrementar la riqueza de la escasa información de entrenamiento y, por lo tanto, mejorar la generalización del clasificador:

   - El clasificador se ha integrado en una arquitectura BAGAN de modo que una red neuronal auxiliar asista con la generación de nuevas muestras a modo de aumentado de datos neuronal.
   - Y se aplican de forma complementaria técnicas de aumentado de datos tradicionales para apoyar el trabajo de la arquitectura BAGAN.

   Se ha comprobado cómo la combinación de estas técnicas ha conseguido mejorar el desempeño del clasificador neuronal, sobre todo ante aquellas categorías del conjunto de datos con menos muestras disponibles. Es más, también se ha comprobado cómo la arquitectura BAGAN adoptada permite aprender a modelar correctamente todas las categorías del problema tratado, generando una variedad de nuevas muestras para todas ellas, incluso para aquellas menos representadas, que podrían resultar fácilmente problemáticas para una arquitectura GAN estándar.

4. Gracias a la aplicación de diversas optimizaciones computacionales, se ha reducido considerablemente el tiempo de ejecución requerido para entrenar las arquitecturas neuronales del esquema de clasificación propuesto. Además, se han evaluado y comparado una gran variedad de configuraciones de estas

arquitecturas, de modo que se han terminado incorporando los criterios de diseño que maximizan la calidad de las clasificaciones resultantes.

5. Hasta donde nuestro conocimiento alcanza, este trabajo es una aportación original, en el sentido de que no se ha encontrado en la literatura ningún trabajo que haya evaluado la efectividad de un esquema BAGAN para aplicaciones de clasificación en teledetección, ni que combine esta variedad de técnicas en un único esquema de clasificación.

Por consiguiente, se han podido cumplir plenamente los objetivos planteados para este Trabajo de Fin de Máster. En cuanto a las posibles ampliaciones del trabajo realizado, se han abierto diversas vías, tales como:

- Reducir el uso de recursos computacionales del algoritmo de segmentación WP y aplicarle técnicas de paralelización, de cara a anular a efectos prácticos un sobrecoste apreciable en tiempo de ejecución por incorporar esta segmentación en superpíxeles al esquema de clasificación de imágenes de teledetección desarrollado.

- Determinar hasta qué punto el algoritmo de segmentación en superpíxeles puede llegar a influir en el desempeño del esquema de clasificación desarrollado. Por ejemplo, podría darse el caso de que otros algoritmos de segmentación basados en el descenso del gradiente[1] como Simple Linear Iterative Clustering (SLIC) generen segmentaciones más precisas que WP, incrementando aún más la calidad de las muestras extraídas en base a superpíxeles, o que generen en lugar de ello segmentaciones de peor calidad.

- Replicar los experimentos con otros conjuntos de datos de imágenes multi e hiperespectrales a modo de completitud, ya que en este trabajo se han empleado imágenes de un único conjunto de datos de cuencas de ríos gallegos.

[1] Cabe recordar que este tipo de algoritmos de segmentación en superpíxeles son los que suelen permitir personalizar las características de los superpíxeles generados, lo cual es un aspecto de gran interés para la clasificación en teledetección, tal y como se explicó en el Apartado 2.1.2.

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. DOI: 10.5772/intechopen.71049. dirección: https://doi.org/10.5772/intechopen.71049.

[2] B. Lu, P. D. Dao, J. Liu, Y. He y J. Shang, «Recent Advances of Hyperspectral Imaging Technology and Applications in Agriculture,» *Remote Sensing*, vol. 12, n.º 16, 2020, ISSN: 2072-4292. DOI: 10.3390/rs12162659. dirección: https://www.mdpi.com/2072-4292/12/16/2659.

[3] F. Argüello, D. B. Heras, A. S. Garea y P. Quesada-Barriuso, «Watershed Monitoring in Galicia from UAV Multispectral Imagery Using Advanced Texture Methods,» *Remote Sensing*, vol. 13, n.º 14, 2021, ISSN: 2072-4292. DOI: 10.3390/rs13142687. dirección: https://www.mdpi.com/2072-4292/13/14/2687.

[4] Z. Shao, P. Tang, Z. Wang, N. Saleem, S. Yam y C. Sommai, «BRRNet: A Fully Convolutional Neural Network for Automatic Building Extraction From High-Resolution Remote Sensing Images,» *Remote Sensing*, vol. 12, n.º 6, 2020, ISSN: 2072-4292. DOI: 10.3390/rs12061050. dirección: https://www.mdpi.com/2072-4292/12/6/1050.

[5] 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. DOI: https://doi.org/10.1111/tgis.12164. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/tgis.12164. dirección: https://onlinelibrary.wiley.com/doi/abs/10.1111/tgis.12164.

[6] A. E. Maxwell, T. A. Warner y F. Fang, «Implementation of machine-learning classification in remote sensing: an applied review,» *International Journal of Remote Sensing*, vol. 39, n.º 9, págs. 2784-2817, 2018. DOI: 10.1080/01431161.2018.1433343. eprint: https://doi.org/10.1080/01431161.2018.1433343. dirección: https://doi.org/10.1080/01431161.2018.1433343.

[7] 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, ago. de 2004, ISSN: 1558-0644. DOI: 10.1109/TGRS.2004.831865.

[8] J. Ham, Y. Chen, M. Crawford y J. Ghosh, «Investigation of the random forest framework for classification of hyperspectral data,» *IEEE Transactions on Geoscience and Remote Sensing*, vol. 43, n.º 3, págs. 492-501, mar. de 2005, ISSN: 1558-0644. DOI: 10.1109/TGRS.2004.842481.

[9] 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. DOI: 10.1561/2200000006. dirección: https://doi.org/10.1561/2200000006.

[10] 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, jun. de 2016, ISSN: 2168-6831. DOI: 10.1109/MGRS.2016.2540798.

[11] 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. DOI: https://doi.org/10.1016/j.isprsjprs.2019.09.006. dirección: https://www.sciencedirect.com/science/article/pii/S0924271619302187.

[12] 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, sep. de 2019, ISSN: 1558-0644. DOI: 10.1109/TGRS.2019.2907932.

[13] 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, jun. de 2019, ISSN: 2168-6831. DOI: 10.1109/MGRS.2019.2912563.

[14] 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, nov. de 1998, ISSN: 1558-2256. DOI: 10.1109/5.726791.

[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. DOI: 10.1155/2015/258619.

[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. DOI: 10.1109/TGRS.2016.2584107. dirección: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84978805819&doi=10.1109%2fTGRS.2016.2584107&partnerID=40&md5=53b27fc18d51389f30fcd9e48cc950c5.

[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. DOI: 10.1109/TGRS.2018.2818945.

[18] Y. Li, H. Zhang y Q. Shen, «SpectralSpatial Classification of Hyperspectral Imagery with 3D Convolutional Neural Network,» *Remote Sensing*, vol. 9, n.º 1, 2017, ISSN: 2072-4292. DOI: 10.3390/rs9010067. dirección: https://www.mdpi.com/2072-4292/9/1/67.

[19] Z. Zhong, J. Li, Z. Luo y M. Chapman, «SpectralSpatial Residual Network for Hyperspectral Image Classification: A 3-D Deep Learning Framework,» *IEEE Transactions on Geoscience and Remote Sensing*, vol. 56, n.º 2, págs. 847-858, feb. de 2018, ISSN: 1558-0644. DOI: 10.1109/TGRS.2017.2755542.

[20] M. E. Paoletti, J. M. Haut, R. Fernandez-Beltran, J. Plaza, A. J. Plaza y F. Pla, «Deep Pyramidal Residual Networks for SpectralSpatial Hyperspectral Image Classification,» *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, n.º 2, págs. 740-754, feb. de 2019, ISSN: 1558-0644. DOI: 10.1109/TGRS.2018.2860125.

[21] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza y col., *Generative Adversarial Networks*, 2014. DOI: 10.48550/ARXIV.1406.2661. dirección: https://arxiv.org/abs/1406.2661.

[22] L. Zhu, Y. Chen, P. Ghamisi y J. A. Benediktsson, «Generative Adversarial Networks for Hyperspectral Image Classification,» *IEEE Transactions on Geoscience and Remote Sensing*, vol. 56, n.º 9, págs. 5046-5063, sep. de 2018, ISSN: 1558-0644. DOI: 10.1109/TGRS.2018.2805286.

[23] M. Frid-Adar, I. Diamant, E. Klang, M. Amitai, J. Goldberger y H. Greenspan, «GAN-based synthetic medical image augmentation for increased CNN performance in liver lesion classification,» *Neurocomputing*, vol. 321, págs. 321-331, 2018, ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2018.09.013. dirección: https://www.sciencedirect.com/science/article/pii/S0925231218310749.

[24] M. Lucic, M. Tschannen, M. Ritter, X. Zhai, O. Bachem y S. Gelly, «High-Fidelity Image Generation With Fewer Labels,» *CoRR*, vol. abs/1903.02271, 2019. arXiv: 1903.02271. dirección: http://arxiv.org/abs/1903.02271.

[25] G. Mariani, F. Scheidegger, R. Istrate, C. Bekas y A. C. I. Malossi, «BAGAN: Data Augmentation with Balancing GAN,» *CoRR*, vol. abs/1803.09655, 2018. arXiv: 1803.09655. dirección: http://arxiv.org/abs/1803.09655.

[26] Ren y Malik, «Learning a classification model for segmentation,» en *Proceedings Ninth IEEE International Conference on Computer Vision*, 2003, 10-17 vol.1. DOI: 10.1109/ICCV.2003.1238308.

[27] 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. DOI: https://doi.org/10.1016/j.cviu.2017.03.007. dirección: https://www.sciencedirect.com/science/article/pii/S1077314217300589.

[28] P. G. Bascoy, A. S. Garea, D. B. Heras, F. Argüello y A. Ordóñez, «Texture-based analysis of hydrographical basins with multispectral imagery,» en *Remote Sensing for Agriculture, Ecosystems, and Hydrology XXI*, C. M. U. Neale y A. Maltese, eds., International Society for Optics y Photonics, vol. 11149, SPIE, 2019, págs. 225-234. DOI: 10.1117/12.2532760. dirección: https://doi.org/10.1117/12.2532760.

[29] 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. DOI: 10.3390/rs12162633. dirección: https://www.mdpi.com/2072-4292/12/16/2633.

[30] 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. DOI: 10.3390/app10248833. dirección: https://www.mdpi.com/2076-3417/10/24/8833.

[31] V. Machairas, M. Faessel, D. Cárdenas-Peña, T. Chabardes, T. Walter y E. Decencière, «Waterpixels,» *IEEE Transactions on Image Processing*, vol. 24, n.º 11, págs. 3707-3716, 2015. DOI: 10.1109/TIP.2015.2451011.

[32] 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. DOI: 10.1117/12.2532760. dirección: http://dx.doi.org/10.1117/12.2532760.

[33] 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.

[34] 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.

[35] 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.

[36] 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. DOI: 10.1109/TPAMI.2012.120.

[37] V. Powell, *Image Kernels explained visually*, En línea; consultado el 11 de junio de 2022. dirección: https://setosa.io/ev/image-kernels/.

[38] S. U. Computer Science Deparment, *CS231n Convolutional Neural Networks for Visual Recognition*, Consultado el 11 de junio de 2022. dirección: https://cs231n.github.io/convolutional-networks/.

[39] J. T. Springenberg, A. Dosovitskiy, T. Brox y M. Riedmiller, *Striving for Simplicity: The All Convolutional Net*, 2014. DOI: 10.48550/ARXIV.1412.6806. dirección: https://arxiv.org/abs/1412.6806.

[40] 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)*, jun. de 2016, págs. 770-778. DOI: 10.1109/CVPR.2016.90.

[41] 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.

[42] M. Tan y Q. V. Le, «EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks,» 2019. DOI: 10.48550/ARXIV.1905.11946. dirección: https://arxiv.org/abs/1905.11946.

[43] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell y S. Xie, *A ConvNet for the 2020s*, 2022. DOI: 10.48550/ARXIV.2201.03545. dirección: https://arxiv.org/abs/2201.03545.

[44] 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. DOI: 10.1109/TGRS.2018.2794326.

[45] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever y R. Salakhutdinov, «Dropout: A Simple Way to Prevent Neural Networks from Overfitting,» *Journal of Machine Learning Research*, vol. 15, n.º 56, págs. 1929-1958, 2014. dirección: http://jmlr.org/papers/v15/srivastava14a.html.

[46] V. Nair y G. E. Hinton, «Rectified Linear Units Improve Restricted Boltzmann Machines,» en *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ép. ICML'10, Haifa, Israel: Omnipress, 2010, págs. 807-814, ISBN: 9781605589077.

[47] D.-A. Clevert, T. Unterthiner y S. Hochreiter, *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)*, 2015. DOI: 10.48550/ARXIV.1511.07289. dirección: https://arxiv.org/abs/1511.07289.

[48] K. He, X. Zhang, S. Ren y J. Sun, *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*, 2015. DOI: 10.48550/ARXIV.1502.01852. dirección: https://arxiv.org/abs/1502.01852.

[49] A. L. Maas, «Rectifier Nonlinearities Improve Neural Network Acoustic Models,» 2013.

[50] C. Shorten y T. M. Khoshgoftaar, «A survey on Image Data Augmentation for Deep Learning,» *Journal of Big Data*, vol. 6, n.º 1, pág. 60, jul. de 2019, ISSN: 2196-1115. DOI: 10.1186/s40537-019-0197-0. dirección: https://doi.org/10.1186/s40537-019-0197-0.

[51] N. Audebert, B. Le Saux y S. Lefevre, «Generative Adversarial Networks for Realistic Synthesis of Hyperspectral Samples,» en *IGARSS 2018 - 2018 IEEE International Geoscience and Remote Sensing Symposium*, jul. de 2018, págs. 4359-4362. DOI: 10.1109/IGARSS.2018.8518321.

[52] V. Dumoulin y F. Visin, *A guide to convolution arithmetic for deep learning*, 2016. DOI: 10.48550/ARXIV.1603.07285. dirección: https://arxiv.org/abs/1603.07285.

[53] M. Mirza y S. Osindero, *Conditional Generative Adversarial Nets*, 2014. DOI: 10.48550/ARXIV.1411.1784. dirección: https://arxiv.org/abs/1411.1784.

[54] Daniel Jurafsky, and James H. Martin, *Speech and Language Processing – An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, 3.ª ed., Draft, ed. 2022, En línea; consultado el 11 de junio de 2022. dirección: https://web.stanford.edu/~jurafsky/slp3/ed3book_jan122022.pdf.

[55] A. Odena, C. Olah y J. Shlens, *Conditional Image Synthesis With Auxiliary Classifier GANs*, 2016. DOI: 10.48550/ARXIV.1610.09585. dirección: https://arxiv.org/abs/1610.09585.

[56] M. A. Kramer, «Nonlinear principal component analysis using autoassociative neural networks,» *AIChE Journal*, vol. 37, n.º 2, págs. 233-243, 1991. DOI: https://doi.org/10.1002/aic.690370209. eprint: https://aiche.onlinelibrary.wiley.com/doi/pdf/10.1002/aic.690370209. dirección: https://aiche.onlinelibrary.wiley.com/doi/abs/10.1002/aic.690370209.

[57] T. Miyato, T. Kataoka, M. Koyama e Y. Yoshida, «Spectral Normalization for Generative Adversarial Networks,» en *International Conference on Learning Representations*, 2018. dirección: https://openreview.net/forum?id=B1QRgziT-.

[58] AgEagle Sensor Systems Inc., d/b/a MicaSense, *REDEDGE–MX DUAL CAMERA IMAGING SYSTEM*, En línea; consultado el 11 de junio de 2022, 2019. dirección: https://micasense.com/wp-content/uploads/2019/11/Trifold-Dual-Camera-Product-Sheet.pdf.

[59] 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. DOI: https://doi.org/10.1016/0169-7439(87)80084-9. dirección: https://www.sciencedirect.com/science/article/pii/0169743987800849.

[60] Y. Xu, L. Zhang, B. Du y F. Zhang, «SpectralSpatial Unified Networks for Hyperspectral Image Classification,» *IEEE Transactions on Geoscience and Remote Sensing*, vol. 56, n.º 10, págs. 5893-5909, 2018. DOI: 10.1109/TGRS.2018.2827407.

[61] RapidEye AG, *The RapidEye Red Edge Band*, En línea; consultado el 11 de junio de 2022. dirección: https://web.archive.org/web/20210312150444/https://www.geoimage.com.au/CaseStudies/Red_Edge_White_Paper.pdf.

[62] Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS), *CiTIUS | Centro Singular de Investigación en Tecnoloxías Intelixentes da USC*, En línea; consultado el 11 de junio de 2022, 2022. dirección: https://citius.gal/.

[63] 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. DOI: https://doi.org/10.1016/0034-4257(91)90048-B. dirección: https://www.sciencedirect.com/science/article/pii/003442579190048B.

[64] Colaboradores de WikiChip, *Xeon Gold 5220 – Intel – WikiChip*, En línea; consultado el 11 de junio de 2022. dirección: https://en.wikichip.org/wiki/intel/xeon_gold/5220.

[65] NVIDIA Corporation, *NVIDIA Tesla V100 GPU Accelerator*, En línea; consultado el 11 de junio de 2022. dirección: https://images.nvidia.com/content/technologies/volta/pdf/tesla-volta-v100-datasheet-letter-fnl-web.pdf.

[66] AnandTech, *Intel Core i7–11700K Review: Blasting Off with Rocket Lake*, En línea; consultado el 11 de junio de 2022. dirección: https://www.anandtech.com/show/16535/intel-core-i7-11700k-review-blasting-off-with-rocket-lake.

[67] www.techpowerup.com, *NVIDIA GeForce RTX 3080 Ti Specs | TechPowerUp GPU Database*, En línea; consultado el 11 de junio de 2022. dirección: https://www.techpowerup.com/gpu-specs/geforce-rtx-3080-ti.c3735.

[68] Canonical Ltd., *Enterprise Open Source and Linux | Ubuntu*, En línea; consultado el 11 de junio de 2022. dirección: https://ubuntu.com/.

[69] Python Software Foundation, *Welcome to Python.org*, En línea; consultado el 11 de junio de 2022. dirección: https://www.python.org/.

[70] Free Software Foundation, Inc., *GCC, the GNU Compiler Collection – GNU Project*, En línea; consultado el 11 de junio de 2022. dirección: https://gcc.gnu.org/.

[71] NVIDIA Corporation, *CUDA Zone – Library of Resources | NVIDIA Developer*, En línea; consultado el 11 de junio de 2022. dirección: https://developer.nvidia.com/cuda-zone.

[72] S. Chetlur, C. Woolley, P. Vandermersch y col., *cuDNN: Efficient Primitives for Deep Learning*, 2014. arXiv: 1410.0759 [cs.NE].

[73] A. Paszke, S. Gross, F. Massa y col., «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. dirección: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[74] Standard C++ Foundation, *Standard C++*, En línea; consultado el 11 de junio de 2022. dirección: https://isocpp.org/.

[75] ISO/IEC JTC 1 (Joint Technical Committee 1) / SC 22 (Subcommittee 22) / WG 14 (Working Group 14), *ISO/IEC JTC1/SC22/WG14 – C*, En línea; consultado el 11 de junio de 2022. dirección: https://www.open-std.org/jtc1/sc22/wg14/.

[76] 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.

[77] Y. LeCun, B. Boser, J. S. Denker y col., «Backpropagation Applied to Handwritten Zip Code Recognition,» *Neural Computation*, vol. 1, n.º 4, págs. 541-551, 1989. DOI: 10.1162/neco.1989.1.4.541.

[78] S. Serebryakov, *How to speed up your PyTorch training | megaserg blog*, En línea; consultado el 13 de junio de 2022, oct. de 2020. dirección: https://sergey.party/2020/10/13/pytorch-performance-guide.html.

[79] S. Migacz, *Performance Tuning Guide – PyTorch Tutorials 1.11.0+cu102 documentation*, En línea; consultado el 13 de junio de 2022, 2022. dirección: https://pytorch.org/tutorials/recipes/recipes/tuning_guide.html.

[80] M. Harris, *How to Optimize Data Transfers in CUDA C/C++ | NVIDIA Technical Blog*, En línea; consultado el 13 de junio de 2022, dic. de 2012. dirección: https://developer.nvidia.com/blog/how-optimize-data-transfers-cuda-cc/.

[81] C. de PyTorch, *PyTorch Profiler With TensorBoard – PyTorch Tutorials 1.11.0+cu102 documentation*, En línea; consultado el 13 de junio de 2022, 2022. dirección: https://pytorch.org/tutorials/intermediate/tensorboard_profiler_tutorial.html.

[82] TensorHub, Inc., *GuildAI – Experiment tracking, ML developer tools*, En línea; consultado el 13 de junio de 2022. dirección: https://guild.ai/.

[83] Weights & Biases, *Weights & Biases – Developer tools for ML*, En línea; consultado el 13 de junio de 2022. dirección: https://wandb.ai/site.

[84] Neptune Labs, *Neptune.ai | Metadata Store for MLOps*, En línea; consultado el 13 de junio de 2022. dirección: https://neptune.ai/.

[85] LF Projects, LLC., *MLflow – A platform for the machine learning lifecycle | MLflow*, En línea; consultado el 13 de junio de 2022. dirección: https://mlflow.org/.

[86] 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. DOI: 10.1007/978-3-642-35289-8_3. dirección: https://doi.org/10.1007/978-3-642-35289-8_3.

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

[88] A. Radford, L. Metz y S. Chintala, *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*, 2015. DOI: 10.48550/ARXIV.1511.06434. dirección: https://arxiv.org/abs/1511.06434.

[89] 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. dirección: http://proceedings.mlr.press/v9/glorot10a.html.

Índice de figuras

Índice de cuadros