
***Facultad de Ciencias***

# DESARROLLO DE MAPAS DE VEGETACIÓN A TRAVÉS DE IMÁGENES DE SATÉLITE CON TÉCNICAS DE DEEP LEERNING

(Development of Vegetation Maps from Satellite Imagery using Deep Learning Techniques)

Trabajo de Fin de Grado
para acceder al

**GRADO EN MATEMÁTICAS**

Autor: Mario Santa Cruz López

Director: Sixto Herrera García

Co-Director: Elena Suarez Peña

JUNIO - 2020

# Resumen

Los mapas de vegetación son una herramienta fundamental en la gestión del territorio. Actualmente para su construcción se aplican diferentes técnicas de teledetección, entrenadas con bases de datos obtenidas en sucesivas campañas de muestreo e imágenes de satélite cubriendo el área a caracterizar.

En el presente estudio, se consideran un conjunto de imágenes desarrollado en la iniciativa Copernicus y los datos de muestreo obtenidos por el Instituto de Hidráulica de Cantabria para obtener, a través de técnicas de aprendizaje profundo, un mapa de vegetación de Cantabria. En particular, se explora el estado del arte en este ámbito y se extienden las redes de convolución tridimensionales, las cuales consideran el eje temporal de las imágenes de satélite, para su aplicación en este problema de teledetección para obtener las hábitats de vegetación presentes en la Comunidad Autónoma de Cantabria a una resolución de $10 \times 10$ metros.

**Palabras clave:** mapas de vegetación, teledetección, redes neuronales, convolución 3D

# Abstract

Vegetation maps are a fundamental tool in land management. Currently different remote sensing techniques are currently applied for its construction, trained with databases obtained in successive sampling campaigns and satellite imagery covering the area to be characterized.

This study considers a set of images developed in the Copernicus program as well as the sampling data obtained by the Instituto de Hidráulica de Cantabria in order to obtain, through Deep Learning techniques, a vegetation map of Cantabria. In particular, the state of the art architectures in this field are explored and the three-dimensional convolution networks, which consider the time axis of the satellite images, are extended to be applied in this remote sensing problem to obtain vegetation habitats present in the Autonomous Community of Cantabria at a spatial resolution of $10 \times 10$ meters.

**Key words:** vegetation maps, remote sensing, neural networks, 3D convolution

# Índice general

# CAPÍTULO 1

# Introducción

## 1.1. Motivación

El término teledetección hace referencia al estudio de las características físicas del entorno a partir de imágenes, generalmente tomadas de satélites, aeronaves o embarcaciones en el caso del suelo marino. Sus aplicaciones son numerosas y se enmarcan en ámbitos muy diversos como la monitorización de ciclones (EUMETSAT), el desarrollo de mapas como *Google Maps* (Ceinsys) o la alerta de talas furtivas en bosques (Fuller, 2006).

Otro ejemplo de aplicación es la realización de mapas de vegetación, los cuales son herramientas muy útiles para monitorización del medio natural y la gestión de políticas territoriales y medioambientales. Para obtener mapas suficientemente representativos y fiables con el mayor nivel de detalle posible se requiere de un gran trabajo de muestreo de la vegetación existente. Además, este trabajo de muestreo debe tener cierta periodicidad o constancia a lo largo del tiempo de modo que dichos mapas se mantengan actualizados y sean una representación fiel de la distribución de la vegetación en cada momento.

Las técnicas de teledetección permiten considerar el conjunto de muestras de campo y las imágenes de satélite correspondientes a las fechas de realización de dicho muestreo para entrenar un modelo de clasificación de los tipos de vegetación. Dicho modelo puede ser aplicado a nuevas imágenes y áreas para generar un mapa de vegetación de la región de interés.

Así, la necesidad antes señalada de recurrencia en la toma de nuevas muestras de campo puede resolverse mayoritariamente con las técnicas de aprendizaje automático y/o profundo, las cuales permiten la creación del modelo de clasificación de vegetación previamente mencionado. El uso de estas técnicas en teledetección

se agrupan en 4 grandes bloques: el preprocesamiento de imágenes, la detección de cambios, la evaluación de resultados, y la clasificación (Ma et al., 2019). En este trabajo daremos énfasis a las aplicaciones del aprendizaje profundo en este último bloque. En particular, se abordará un problema de clasificación de cobertura y uso del suelo para el desarrollo de un mapa de vegetación de Cantabria.

En este ámbito de estudio es habitual que los datos estén georeferenciados. Esto es, cada píxel de una imagen y cada dato de muestreo se corresponden a un área o localización espacial concreta dentro de la superficie terrestre definida por sus coordenadas (p.e. longitud, latitud y altura) y resolución, entendida como el área correspondiente a cada píxel de la imagen. En este sentido, cuanto más pequeños sean dichos píxeles mayor será la resolución espacial de la imagen. La referencia geoespacial nos permite aprovechar otras fuentes de información, como capas de SIG (Sistemas de Información Geográfica) o datos de otros sensores, para combinarlos y conseguir conjuntos de datos con una alta resolución espectral. Esto significa que los datos tienen el mayor número posible de bandas en el espectro electromagnético representadas (R-red, G-green, B-blue, infrarrojo cercano, etc. . . ) con el objetivo de discriminar mejor los elementos incluidos en la imagen.

En los últimos años se está produciendo un aumento incesante de imágenes satélites gracias a proyectos como los iniciados por el Programa Copernicus (copernicus.eu/en), coordinado por la Comisión Europea a través de la Agencia Espacial Europea (ESA), cuyo objetivo es recabar información a través de diversas fuentes de datos (satélite, modelos climáticos, observaciones, etc. . . ) que nos permita monitorizar y entender mejor el sistema climático y así gestionar de forma sostenible el medio natural (ver Comission, 2017). Del mismo modo, Copernicus proporciona en tiempo casi real datos globales de diversas fuentes, incluyendo imágenes de satélite, para su uso por parte de la comunidad. Como consecuencia, actualmente se dispone de numerosas imágenes en diferentes instantes de tiempo, incrementando la relevancia de este eje temporal en las metodologías habitualmente empleadas para realizar dicha clasificación.

A partir de un conjunto de datos sobre hábitats de vegetación observados por el Instituto de Hidráulica Ambiental de Cantabria (IHCantabria), se ha propuesto crear un modelo que permita generar un mapa de vegetación de Cantabria a partir de imágenes de satélites, estudiando el valor añadido que tiene el tratamiento de la variable temporal.

## 1.2. Estado del arte

Las técnicas de aprendizaje profundo o *deep learning* surgen para solventar problemas como el reconocimiento de objetos o voz, tareas en las cuales los algoritmos tradicionales fracasan a la hora de generalizar correctamente, ya que no son capaces de aprender funciones complejas en espacios de gran dimensionalidad. Asimismo, las técnicas convencionales de aprendizaje automático experimentaban una gran dificultad para procesar los datos en bruto (Goodfellow et al., 2016). Por otra parte, el incesante aumento de información en los últimos años hace cada vez más necesario trabajar con datos de alta dimensionalidad y, por lo tanto, adoptar aquellas técnicas de aprendizaje mejor adaptadas a este nuevo paradigma. Este problema de la alta dimensionalidad empezó a ser estudiado hace varias décadas por sus consecuencias en el reconocimiento de patrones. En Hughes (1968) se presenta el llamado *efecto Hughes*, el cual describe como la precisión del clasificador depende del número de muestras de entrenamiento y de la dimensión del espacio (*curse of dimensionality*), mientras que en Cover (1965) se empieza a tratar matemáticamente este problema de la dimensionalidad en clasificadores tanto lineales como polinomiales.

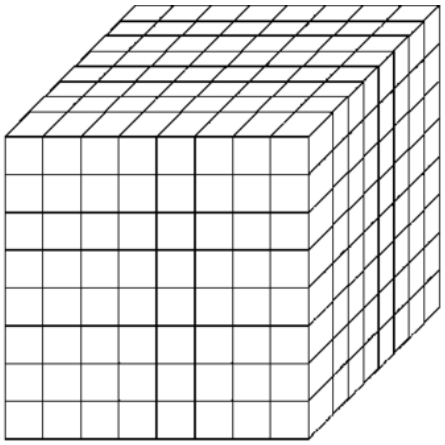

Figura 1.1: Espacio 3D discretizado. Fuente: `http://msvlab.hre.ntou.edu.tw/grades/bem(2008)/IJNME2008.pdf`

Dicho problema supone un importante desafío estadístico. Por ejemplo, agrupemos cada dimensión en distintos valores discretos. De esta manera tenemos todas las muestras de entrenamiento distribuidos por el espacio muestral dentro de regiones, que en el caso de dimensión 2 son cuadrículas, o en el de dimensión 3, cubos (ver figura 1.1). De este modo, para generalizar a nuevos datos, una idea sería estudiar las muestras de entrenamiento de la región a la que pertenece la nueva muestra y

tomar, por ejemplo, la moda o la media según sea un problema de clasificación o regresión.

Consecuentemente, para el caso de $n$ dimensiones con $m$ posibles valores discretos en cada dimensión el número de regiones, y por tanto también el número de muestras de entrenamiento, que necesitaremos es del orden $O(m^n)$, ya que necesitamos al menos 1 ejemplo para cada región posible. En la figura 1.1 se puede apreciar como necesitaríamos al menos $2^3$, $2^6$ y $2^9$ para los casos 1D, 2D y 3D respectivamente, según consideremos un segmento, una cara o el cubo completo.

En el área de la teledetección nos enfrentamos a datos de una gran dimensionalidad debido a toda la información que consideramos en relación a las bandas espectrales y los píxeles de cada imagen. Por ello, en este área, está muy extendido el uso de redes neuronales convolucionales o CNN (del inglés Convolutional Neural Network) diseñadas para extraer información espacial de imágenes (Zhu et al., 2017), que es el formato propio de cualquier problema de teledetección. En el problema abordado en el presente trabajo se ha estudiado extensivamente el uso de estas redes así como de otros tipos de redes neuronales o aproximaciones más clásicas como las máquinas de vectores soporte o SVMs (del inglés Support Vector Machines) y los bosques aleatorios o RF (del inglés Random Forest).

En base a la literatura revisada, las soluciones capaces de capturar mejor las características espaciales han sido las CNNs, con una diferencia significativa respecto al resto de alternativas, como muestran autores como Krizhevsky et al. (2012) donde se consideran varios conjuntos de imágenes distintos. En el caso específico de datos de teledetección, Makantasis et al. (2015) concluye que las CNN baten a las SVMs a la vez que permiten utilizar conjuntos de datos más grandes mientras que Hu et al. (2015) obtiene resultados similares respecto a las SVMs y los extiende a las redes neuronales clásicas. Además se ha visto que también superan en desempeño a RF y Ensambladores de Perceptrones multicapa en la clasificación de cosechas y uso del suelo (Kussul et al., 2017).

En virtud de los resultados obtenidos en diferentes problemas, el uso de CNNs ha crecido enormemente en los últimos años. En particular, en el ámbito de la teledetección más del 60 % de publicaciones científicas de aprendizaje profundo emplean CNNs (Ma et al., 2019), adquiriendo una gran relevancia en problemas de clasificación de cobertura y uso del terreno (Maggiori et al., 2016).

Recientemente se han extendido las CNNs para incluir tanto patrones espaciales como temporales mostrando una mejor capacidad de discriminación para la clasificación de cultivos (Garnot et al., 2019) que aquellas CNNs basadas únicamente en patrones espaciales o temporales. La aproximación considerada en dicho estudio se

basa en el anidamiento de CNNs con redes neuronales recurrentes que permitan la inclusión de dicha componente temporal.

Otros autores han introducido CNNs con convoluciones 3-dimensionales (3-CNNs) para considerar otra dimensión no espacial (p.e. temporal o espectral) obteniendo grandes resultados en ámbitos diversos como la clasificación de acciones humanas o el reconocimiento de objetos tanto en vídeo como en tiempo real (Maturana and Scherer, 2015; Ji et al., 2012), y la clasificación de imágenes (Chen et al., 2016; Hamida et al., 2018; Li et al., 2017). Cabe destacar que en Hamida et al. (2018) se consigue batir a CNNs observando las relaciones espaciales y espectrales de las imágenes a un coste inferior.

Como se ha descrito, el problema de clasificación afrontado en este trabajo se trata de un problema de alta dimensionalidad para el cual se disponen de datos con componentes tanto espaciales como temporales, siendo por tanto las 3-CNNs un candidato adecuado para su resolución en base a la literatura consultada. Por ello, se propone la aplicación de 3-CNNs para el desarrollo de un mapa de vegetación de Cantabria a partir de los datos de muestreo recogido por el IHCantabria y las imágenes de satélite disponibles en el programa Copernicus.

En este caso, y a diferencia de los estudios citados anteriormente, se considerará la variabilidad temporal como tercera dimensión en la arquitectura del modelo 3-CNN propuesto. Dicha componente es muy relevante en la clasificación de hábitats terrestres dado que la evolución a lo largo del tiempo es muy característica en ciertas hábitats. Por ejemplo, la evolución del trigo depende de la época del año, siendo de un color u otro, definiendo así un patrón temporal característico. De modo similar, la vegetación de hoja caduca y perenne puede ser identificada en base a la evolución temporal de su floración y follaje. Es de esperar que la inclusión de esta dimensión al modelo de lugar a un incremento en su capacidad de discriminación.

## 1.3. Objetivos

Una vez establecido el estado del arte y el objetivo general de este trabajo, a continuación detallaremos los objetivos específicos a resolver en el presente estudio.

En líneas generales, el objetivo principal es la asignación del hábitat EUNIS (Davies et al., 2004) correspondiente a cada píxel en base a las imágenes de satélite y a una cartografía de referencia.

Nuestra tarea es evaluar la mejoría en la generación de mapas de vegetación con el uso de CNNs, en línea con lo que se describe en la literatura científica consultada. En concreto, la región de interés será Cantabria sobre la cual se considerará un

conjunto de imágenes de satélite y muestras de campo de hábitats de vegetación.

Estudiaremos la potencialidad de arquitecturas 3-CNN para extraer la información espacio-temporal en comparación con las CNN, que son las arquitecturas de referencia en la actualidad en este sector, pero que únicamente tienen en consideración características espaciales.

Si bien no se presenta en esta memoria, otro objetivo fue la comparación con las técnicas tradicionales más utilizadas, como por ejemplo, aquellas basadas en Máxima Entropía ó *MaxEnt*.

# CAPÍTULO 2

# Datos y Métodos

## 2.1. Descripción de los datos

El conjunto de imágenes de datos utilizadas ha sido extraído por el sensor MSI (Multi Spectra Instrument) de los satélites Sentinel-2A y 2B impulsados por la ESA a través del programa Copernicus. Estos satélites recorren toda la superficie terrestre cada cinco días, proporcionando imágenes multiespectrales de resoluciones espaciales de 10, 20 y 60 metros. Nosotros hemos utilizado las imágenes con sistema de referencia ETRS89 proyección UTM y huso 30.

El presente trabajo considera variables predictoras relativas tanto a factores ambientales limitantes (topografía, clima y suelo) como a teledetección (imágenes de Sentinel-2A, 2B y LiDAR [1], todas ellas cubriendo la extensión total de Cantabria y reinterpoladas a una resolución espacial de $10m$. Dentro de las bandas disponibles, se han utilizado 12 de las bandas climáticas: B2, B3, B4, B5, B6, B7, B8, B8A, B11, B12, EVI, NDVI, cuyo significado se detalla en el Anexo A. Para esta selección se ha tenido en cuenta que la región del infrarrojo cercano (a la cual pertenecen todas a excepción de las bandas B2, B3 y B4) es la más interesante para la discriminación del componente vegetal, Campbell and Wynne (2011).

El IHCantabria ha sido el encargado de la descarga de los datos y de las correcciones tanto atmosféricas como topográficas. Para la obtención de las imágenes finales han enfrentado varios problemas en el procesamiento de estos datos. Por ejemplo, Cantabria es una región con gran predominancia de cielos nubosos a lo largo de todo el año. Estas nubes impiden tener datos en numerosas zonas y, por tanto, cada año se han seleccionado las imágenes con menor cobertura de nubes para

[1] El LiDAR es un escáner láser aerotransportado que nos proporciona modelos de contorno y elevación, de los cuales se puede extraer la elevación de la vegetación.

tener que eliminar la menor cantidad de datos ausentes posible. Por ello, aunque el satélite nos proviene de imágenes cada 5 días, nosotros hemos utilizado imágenes correspondiente a 9 fechas distintas desde el 16 de Julio de 2016 hasta el 1 de Junio de 2019, las cuales son detalladas en el Anexo A.

El objetivo de nuestra red neuronal es predecir la categoría EUNIS (ver Anexo B) de cada píxel de Cantabria a partir de los valores de las variables predictoras. Para el entrenamiento de la red hemos aprovechado el muestreo de datos continuado que realizan botánicos expertos en colaboración con IHCantabria.

A partir de este proceso vamos a construir un modelo para clasificar la vegetación de un total de 88 millones de cuadrículas de $10 \times 10m$ en la región de Cantabria, de los cuales sabemos la clasificación EUNIS real de 21468 de ellas gracias al muestreo de los botánicos, cuya distribución se muestra en la figura 2.1. Estas muestras formarán nuestros conjuntos de entrenamiento y evaluación.

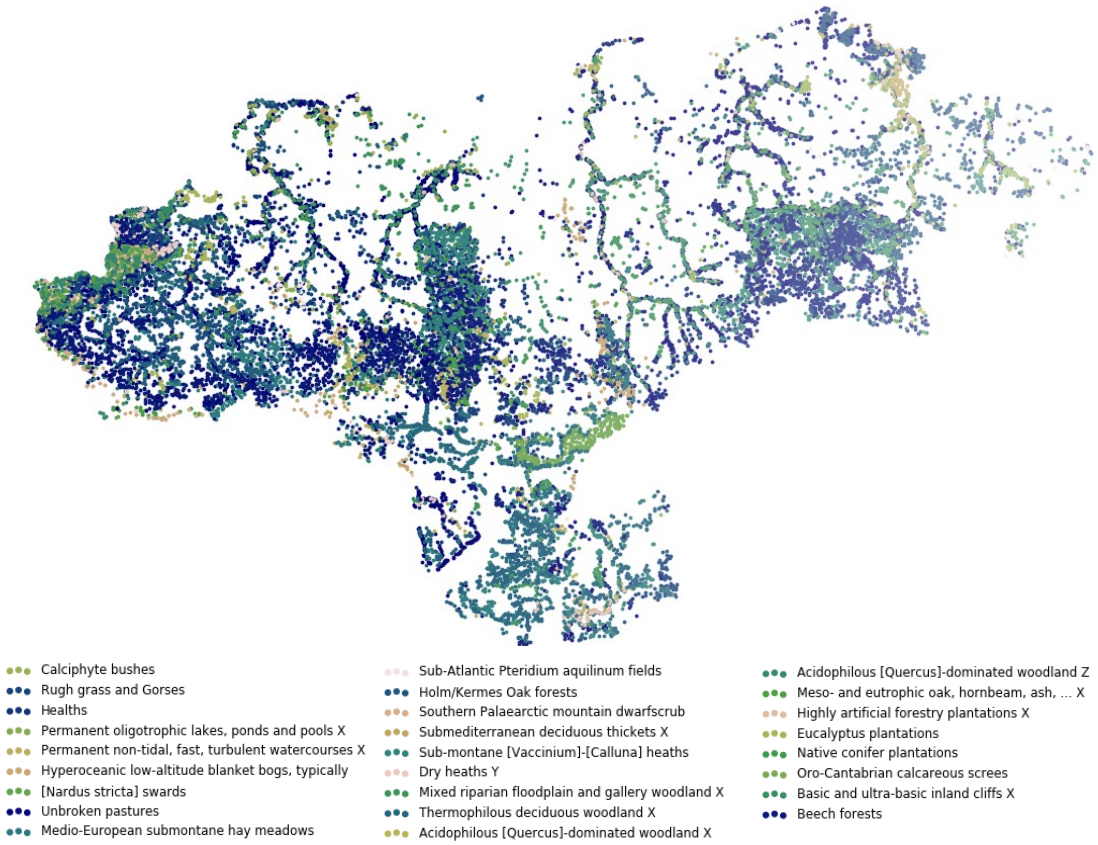


Figura 2.1: Mapa de las muestras tomadas por un equipo de IHCantabria disponibles en la región de Cantabria.

## 2.2. Notación

En esta sección introducimos la notación que utilizaremos a lo largo del resto de la memoria. Sea $S$ un conjunto no vacío, denotamos por $S^n$ al conjunto de vectores con $n$ coordenadas y coeficientes en $S$. Sea un vector, $x = (x_i) \in S^n$,

$$x = (x_i)_{1 \leq i \leq n} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

Sea $S$ un conjunto no vacío, denotamos por $\mathcal{M}_{n,m}(S)$ al conjunto de todas las matrices de tamaño $n \times m$ con coeficientes en $S$. Sea $A \in \mathcal{M}_{n,m}(S)$,

$$A = (a_{i,j})_{1 \leq i \leq n, 1 \leq j \leq m} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,m} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,m} \end{pmatrix}$$

Utilizaremos indistintamente las notaciones $A(i,j)$ y $a_{i,j}$ para referirnos al elemento en la $i$-ésima fila y en la $j$-ésima columna.

Dicha notación se extenderá de forma análoga a matrices de dimensiones superiores, por ejemplo, cuando tratemos la variabilidad temporal usamos matrices tridimensionales $\mathcal{M}_{n,m,l}(S)$.

**Definición 2.1** (**Producto Hadamard**). *Sea $\mathcal{S}$ un conjunto no vacío y $\cdot$ una operación interna definida en dicho conjunto, se define el producto de Hadamard ($C = A \odot B$) como la operación interna en el espacio de matrices $\mathcal{M}_{n,m}(S)$*

$$\begin{aligned} \odot : \mathcal{M}_{n,m}(S) \times \mathcal{M}_{n,m}(S) &\to \mathcal{M}_{n,m}(S) \\ (A, B) &\mapsto C \end{aligned}$$

*donde $C := (c_{i,j})$ queda definida en función de $A = (a_{i,j})$ y $B = (b_{i,j})$ como*

$$c_{i,j} = a_{i,j} \cdot b_{i,j}, \quad \forall i, j \in \mathbb{N} \text{ tal que } 1 \leq i \leq n,\ 1 \leq j \leq m$$

## 2.3. Teoría de la decisión

Es importante definir unos objetivos claros para poder construir un buen modelo. En cualquier problema de modelización con aprendizaje automático tenemos tres espacios comunes: el espacio de entrada, $\mathcal{X}$, el espacio de acción $\mathcal{A}$, y el espacio de salida $\mathcal{Y}$.

Considerando que disponemos de $n_t$ puntos temporales, donde en cada fecha disponemos de $n_b$ bandas distintas y en cada imagen $I$ consideramos un contexto espacial para cada píxel de tamaño $l_x \times l_y$, el espacio de entrada es el conjunto de las variables predictoras en varios puntos temporales en los píxeles de la región $l_x \times l_y$, esto es, $\mathcal{X} = \left(\mathcal{M}_{l_x,l_y}(\mathbb{R})\right)^{n_t \cdot n_b}$ para el caso general, y $\mathcal{X} = \left(\mathcal{M}_{l_x,l_y,n_t}(\mathbb{R})\right)^{n_b}$ cuando estudiemos la variabilidad temporal.

Por otro lado, los espacios de salida y acción se corresponden con,

$$\mathcal{A} = \{x = (x_i) \in [0,1]^h \mid \sum_{i=i}^{h} x_i = 1\}, \quad \mathcal{Y} = \{x = (x_i) \in \{0,1\}^h \mid \sum_{i=i}^{h} x_i = 1\}$$

donde $h$ es el número de hábitats EUNIS distintos y la coordenada $i$-ésima de cada elemento representa la probabilidad de ocurrencia del $i$-ésimo EUNIS.

Agruparemos los datos en tuplas $(x, y)$ donde $x \in \mathcal{X}$ son los datos de entrada del modelo que corresponden a un píxel e $y \in \mathcal{Y}$ corresponde con la categoría EUNIS de dicho píxel. Para ello asumimos que los datos provienen de una distribución $P_{\mathcal{X}\times\mathcal{Y}}$ y que todo par $(x, y) \in \mathcal{X} \times \mathcal{Y}$ constituye una realización de dicha distribución independiente del resto de datos de la muestra. A partir de ahora consideramos el conjunto de entrenamiento $D_{entre} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$ y el conjunto de evaluación $D_{test} = \{(x_{N+1}, y_{N+1}), \ldots, (x_{N+M}, y_{N+M})\}$.

**Definición 2.2.** *Una función de decisión, $f$, toma como valor de entrada un elemento del espacio de entrada y retorna una acción.*

$$\begin{aligned} f : \mathcal{X} &\to \mathcal{A} \\ x &\mapsto f(x) \end{aligned}$$

A esta función de decisión también le llamaremos modelo a partir de ahora. Nuestro objetivo es conseguir dos modelos,$f$ y $g$, uno con convoluciones 2D y otro 3D,

$$f, g : \left(\mathcal{M}_{l_x,l_y}(\mathbb{R})\right)^n \to [0,1]^h$$

**Definición 2.3.** *Una función de pérdida, $\mathcal{L}$, evalúa una acción $a$ en el contexto de un valor de salida $y$,*

$$\begin{aligned} \mathcal{L} : \mathcal{A} \times \mathcal{Y} &\to \mathbb{R} \\ (a, y) &\mapsto \mathcal{L}(a, y) \end{aligned} \tag{2.1}$$

Esta función nos permite evaluar una única acción, por lo que todavía necesitamos definir como evaluamos nuestra función de decisión globalmente.

**Definición 2.4.** *El riesgo de una función de decisión $f : \mathcal{X} \to \mathcal{A}$ es la perdida esperada de una nueva muestra $(x, y)$ tomada aleatoriamente de $P_{\mathcal{X}\times\mathcal{Y}}$ y viene dado por la expresión,*

$$R(f, \theta) \;=\; E[\mathcal{L}(f(x; \theta), y)]$$

*donde $\theta$ son los parámetros correspondientes a la función decisión considerada.*

Sin embargo, el valor de esta esperanza no lo podemos calcular, dado que desconocemos el valor de la distribución anterior, por lo que se estimará a partir del *Riesgo empírico.*

**Definición 2.5.** *Dado $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, el riesgo empírico de $f : \mathcal{X} \to \mathcal{Y}$ en el conjunto $D$ viene dado por la expresión,*

$$\widehat{R}_n(f, \theta) \;=\; \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(f(x_i; \theta), y_i)$$

*donde $\theta$ son los parámetros correspondientes al modelo considerado.*

De este modo, haciendo uso de la *Ley Fuerte de los Grandes Números* tenemos que:

$$\lim_{n\to\infty} \widehat{R}_n(f, \theta) = R(f, \theta)$$

casi seguro.

En este contexto, llamamos función de decisión de Bayes, o función objetivo, a la función que obtiene el menor riesgo entre toda las funciones de $\mathcal{X}$ en $\mathcal{A}$. Al valor del riesgo de esa función lo llamamos riesgo de Bayes. Nuestro modelo $f$ es un minimizador del riesgo empírico cuando sus parámetros tienen valores $\widehat{\theta} = \arg_\theta \min \widehat{R}_n(f, \theta)$ sobre todas las posibles combinaciones de valores. En este caso, en vez de minimizar el riesgo empírico sobre todas las funciones nos restringimos a un espacio de hipótesis, $\mathcal{H} \subseteq \mathcal{X}$, en las cuales consideramos un subconjunto de funciones diferenciales que son las que podemos representar con las redes neuronales consideradas.

## 2.4. Neurona

El elemento base de cualquier red neuronal es la neurona. Una neurona toma un número $m$ de datos numéricos de entrada y realiza una suma ponderada por unos pesos, que son los parámetros de las capas que podemos ajustar.

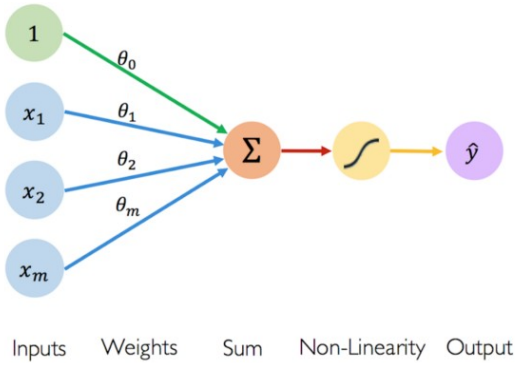


Figura 2.2: Esquema de una neurona. Fuente:MIT Deep Learning 6.S191. `http://introtodeeplearning.com`

Una vez realizada la suma, se añade un bias o sesgo, que también es un parámetro que se ajusta durante el entrenamiento. Después se aplica una transformación no lineal que permite a nuestro modelo capturar relaciones no lineales que de otro modo no sería posible, como se muestra esquemáticamente en la figura 2.3. Dicha transformación se denomina función de activación como se verá mas adelante.

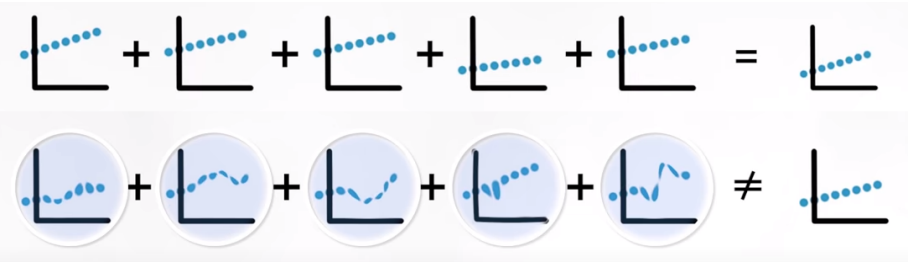

Figura 2.3: Representación de la concatenación de neuronas. Fuente: `https://youtu.be/uwbHOpp9xkc?t=288`

**Definición 2.6.** *Sean $W \in \mathbb{R}^m$ los pesos de la neurona, $b \in \mathbb{R}$ es el sesgo y $\sigma$ es la función de activación. Definimos una neurona como la función $h$ caracterizada por la terna $(W, b, \sigma)$ que está definida como*

$$\begin{aligned} h : \mathbb{R}^m &\to \mathbb{R} \\ x &\mapsto \sigma(W^T \cdot x + b) \end{aligned}$$

donde $W^T$ representa el vector transpuesto de $W$.

### 2.4.1. Funciones de activación

Llamamos funciones de activación a las transformaciones no lineales tratadas anteriormente que son necesarias para aprender relaciones no lineales en los datos. En este trabajo se han considerado dos funciones de activación distintas, conocidas como *ReLU* y *Softmax*, las cuales se definen a continuación para el contexto en el que las utilizamos. Las derivadas de este funciones están definidas en el Anexo C.1.

**Definición 2.7** (ReLU)**.** *Decimos que una neurona tiene como función de activación ReLU (o Rectified Linear Unit) cuando los valores de salida de su neurona $x \in \mathbb{R}$ se les aplica la siguiente función,*

$$\begin{aligned} \sigma : \mathbb{R} &\to \mathbb{R} \\ x &\mapsto \text{máx}(0, x) \end{aligned}$$

La función de activación ReLU no es diferenciable en 0, si bien su derivada es 1 para el resto de valores. Esta es una de las razones por las cuales es una de las funciones más utilizadas en aprendizaje profundo, Glorot et al. (2011), ya que computacionalmente es muy eficiente. Esta eficiencia se debe a que los algoritmos utilizados para el ajuste de los pesos de cada capa (*back-propagation*) utilizan variantes del algoritmo de descenso del gradiente donde es necesario calcular las derivadas de las funciones que componen el modelo.

**Definición 2.8** (Softmax)**.** *Un conjunto de N neuronas tiene función de activación Softmax cuando a los valores de salida $z = (z_i)_{1 \leq i \leq N} \in \mathbb{R}^N$ de sus N neuronas se les aplica la siguiente función,*

$$\begin{aligned} \sigma_N : \mathbb{R}^N &\to [0,1]^N \\ (z_i) &\mapsto \left( \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}} \right) \end{aligned}$$

Esta función de activación es muy usada en la última capa de modelos cuyas variables de salida son probabilidades, dado que los valores obtenidos se encuentran en el intervalo $[0, 1]$, y la suma de todos ellos es 1.

De este modo, usando esta función de activación obtendremos un vector $z \in [0, 1]^h$, donde $h$ es el número de categorías EUNIS consideradas, cuya componente $i$-ésima contendrá la probabilidad de que el píxel analizado se corresponda con la $i$-ésima categoría EUNIS.

## 2.5. Capas

En un modelo de aprendizaje profundo se denomina capa a una agrupación de neuronas que toman un conjunto de datos y realizan transformaciones lineales y no lineales. En particular, decimos que una capa tiene función de activación $\sigma$ cuando las neuronas de dicha capa tienen función de activación $\sigma$.

Las redes neuronales constan de varias capas que identificamos con funciones y que situaremos secuencialmente. Regularmente, se representan por un grafo acíclico dirigido que describe como estas funciones se componen. El número de capas que posee la red se conoce como profundidad de la red. La primera capa de la red se denomina capa de entrada, a la última,capa de salida, y las situadas entre estas se conocen como capas ocultas.

Dada la expresión $y = f(x)$, donde $f$ es nuestro modelo o función de decisión que asigna a cada entrada asociada a un píxel $(x)$ la probabilidad de cada categoría EUNIS $(y = (y_1, ..., y_h))$, tenemos que $f = f^{(1)} \circ f^{(2)} \circ \ldots \circ f^{(m)}$, donde cada función $f^{(i)}$ se identifica con la capa $i$-ésima de la red.

### 2.5.1. Capa densa

Una capa densa consiste en un número finito $n$ de neuronas compartiendo $m$ datos de entrada de tal manera que la salida de la capa es la concatenación de cada valor de salida de las neuronas. En este caso tenemos que la capa densa es una función $h : \mathbb{R}^m \to \mathbb{R}^n$ (ver en figura 2.4 ejemplo de $h$ con $n = 4$ y $m = 3$).

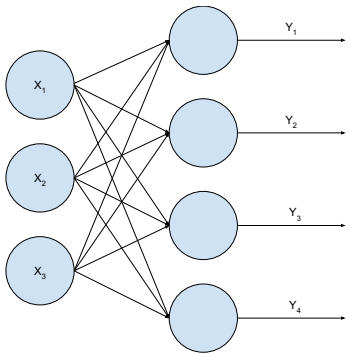


Figura 2.4: Ejemplo de capa densa definida por $h(X_1, X_2, X_3) = (Y_1, Y_2, Y_3, Y_4)$.

**Definición 2.9** (**Capa densa**). *Dados $n \in \mathbb{N}$ el número de neuronas de la capa y $m \in \mathbb{N}$ el número de datos de entrada. Sean $W \in \mathcal{M}_{n,m}(\mathbb{R})$ los pesos de las neuronas, donde la $i$-ésima columna de la matriz representa los pesos de la $i$-ésima*

*neurona,* $b \in \mathbb{R}^m$ *son los sesgos de las* $m$ *neuronas y* $\sigma$ *es la función de activación. Definimos una capa densa de* $m$ *neuronas como la función* $h$ *caracterizada por la terna* $(W, b, \sigma)$ *que está definida como*

$$\begin{aligned} h : \mathbb{R}^m &\to \mathbb{R}^n \\ x &\mapsto \sigma(W \cdot x + b) \end{aligned}$$

La definición de capa densa es una generalización de la definición de neurona. Bastaría tomar el número de neuronas $m = 1$ para obtener la definición de la primera.

### 2.5.2. Capas de convolución

Los datos de entrada a una CNN se corresponden con una imagen multibanda. En el contexto del modelo se utiliza el nombre de canales para referirnos al número de matrices que obtenemos tras cada capa de convolución. Notar que, en este caso, cada matriz se corresponde con una imagen obtenida a través de la transformación por la capa de convolución de las imágenes de entrada. En el caso de la primera capa de convolución el número de canales de entrada a la capa coincide con el número de bandas consideradas en los datos. Cuando tratemos con la variable temporal las capas de nuestra red utilizarán canales que son elementos de $\mathcal{M}_{i,j,k}$ y cuando no tratemos las variables serán elementos de $\mathcal{M}_{i,j}$, que es un caso particular, $\mathcal{M}_{i,j,1}$.

**Definición 2.10 (Convolución).** *Sean* $I \in (\mathcal{M}_{l_x \times l_y \times l_z}(\mathbb{R}))^d$ *y una tupla de d filtros,* $K \in (\mathcal{M}_{k_x \times k_j \times k_z}(\mathbb{R}))^d$*. Sean* $p_x$*,* $p_y$*,* $p_z$ *las distancias, o pasos, a considerar entre dos posiciones consecutivas a lo largo de los ejes* $X$*,* $Y$ *y* $Z$ *y sean* $r_x$*,* $r_y$ *y* $r_z$ *los números de ceros que añadir al principio y al final de cada eje. Definimos la convolución de* $I$ *con filtro* $K$ *con pasos* $(p_x, p_y, p_z)$ *y rellenado* $(r_x, r_y, r_z)$*, como la suma de los resultados de aplicar la convolución* $*$ *aplicada a cada canal,*

$$\begin{aligned} * : \left(\mathcal{M}_{l_x,l_y,l_z}(\mathbb{R})\right)^d \times \left(\mathcal{M}_{k_x,k_j,k_x}(\mathbb{R})\right)^d &\to \mathcal{M}_{n_x,n_y,n_z}(\mathbb{R}) \\ (I, K) \qquad\quad &\mapsto \quad S = \sum_{b=1}^{d} (I_b * K_b) \end{aligned}$$

*donde* $n_x = (l_x - k_x + 2r_x)/p_x + 1$*,* $n_y = (l_y - k_y + 2r_y)/p_y + 1$ *y* $n_z = (l_z - k_z + 2r_z)/p_z + 1$ *y por tanto, definimos la convolución en el b-ésimo canal como*

$$\begin{aligned} (I_b * K_b)(i, j, k) = \sum_{m=0}^{k_y - 1} \sum_{n=0}^{k_x - 1} \sum_{s=0}^{k_z - 1} & K_b(m + 1, n + 1, s + 1) \cdot \\ & \cdot I'_b(p_x(i - 1) + 1 + m, p_y(j - 1) + 1 + n, p_z(k - 1) + 1 + s) \end{aligned}$$

*donde $I'_b$ es la matriz con rellenado $(r_x, r_y, r_z)$. Cuando tratemos matrices de $\mathcal{M}_{l_x,l_y}(\mathbb{R})$ lo consideraremos un caso particular con $l_z = 1$ (a su vez, $k_z = p_z = 1, r_z = 0$).*

Para una capa de convolución se diferencia dos tipos de rellenado. El primero, conocido como *valid padding*, hace referencia a no incluir ninguna fila o columna de 0's, lo que para nosotros significa que $r_x = r_y = r_z = 0$. La otra alternativa es utilizar *same padding* en el cual se incluye el número necesario de columnas y filas de 0's de tal manera que la matriz de salida de la capa tiene el mismo tamaño que la de entrada. En adelante cuando hagamos referencia a una convolución daremos por entendido que utiliza *valid padding* y con pasos $(1, 1, 1)$.

**Definición 2.11** (**Capa de Convolución 2D**). *Dados $n$ el número de canales salientes, $W = \{W_1, \ldots, W_n\}$ un conjunto de $n$ filtros con $W_i \in \left(\mathcal{M}_{k_x,k_y}(\mathbb{R})\right)^d$, $\gamma = \{b_1, \ldots, b_n\}$ un conjunto de sesgos y $\sigma(\cdot)$ una función de activación. Sean $X \in \left(\mathcal{M}_{l_x,l_y}(\mathbb{R})\right)^d$ los datos de entrada de la capa de convolución.*

*Se define como capa de convolución la terna $(W, \gamma, \sigma)$, cuya salida para la entrada $X$ es la $n$-tupla $(Y_1, \ldots, Y_n)$ $\left(\mathcal{M}_{i_x,i_y}(\mathbb{R})\right)^n$, donde cada canal saliente $Y_i \in \mathcal{M}_{n_x,n_y}(\mathbb{R})$ se define*

$$Y_i \;=\; \sigma\left(X * W_i + b_i\right)$$

**Definición 2.12** (**Capa de Convolución 3D**). *Sea $n$ el número de canales saliente, $W = \{W_1, \ldots, W_n\}$ un conjunto de $n$ filtros con $W_i \in \left(\mathcal{M}_{k_x,k_y,k_z}(\mathbb{R})\right)^d$, $\gamma = \{b_1, \ldots, b_n\} \in \mathbb{R}^n$ un conjunto de sesgos y $\sigma(\cdot)$ una función de activación. Sean $X \in \left(\mathcal{M}_{l_x,l_y,l_z}(\mathbb{R})\right)^d$ los datos de entrada de la capa de convolución.*

*Se define como capa de convolución la terna $(W, \gamma, \sigma)$, cuya salida para la entrada $X$ es la $n$-tupla $(Y_1, \ldots, Y_n) \in \left(\mathcal{M}_{i_x,i_y}(\mathbb{R})\right)^n$, donde cada canal saliente $Y_i \in \mathcal{M}_{n_x,n_y,n_z}(\mathbb{R})$ se define*

$$Y_i \;=\; \sigma\left(X * W_i + b_i\right)$$

*donde $+$ se refiere a sumar $b_i$ a todos los elementos de $X * W_i$.*

Por lo tanto, tenemos que cada valor de salida de una capa de convolución es la salida de una neurona que depende de una pequeña región de los canales de entrada de la red. De este modo llamamos campo receptivo de una neurona al área (o volumen) en cada canal de entrada de la red que determina la salida de esa neurona. Cuando consideramos una neurona de la primera capa de convolución, este campo receptivo se corresponde con el tamaño del filtro aplicado en esa primera capa, que en las figura 2.5 y 2.6 corresponde con el área sombreada. En nuestro caso nos interesa el campo receptivo de las neuronas resultantes de cada bloque de convolución.

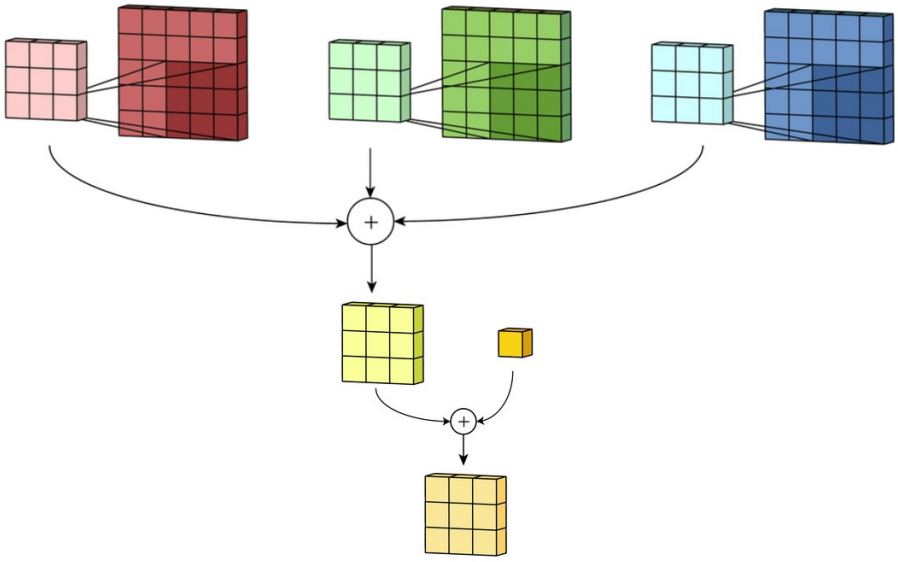

Figura 2.5: Esquema de una capa de convolución que transforma unos datos de entrada de tamaño $5 \times 5$ con 3 canales en rojo, verde, y azul, en un único canal de tamaño $3 \times 3$. Para ello se utiliza 1 filtro de tamaño $3 \times 3 \times 3$. Podemos entender este filtro como la combinación de 3 filtros de tamaño $3 \times 3$ que se aplica cada uno a un canal distinto de los datos de entrada. Por último, se añade un sesgo común para cada canal de salida. Fuente: `https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1`

Durante el entrenamiento queremos aprender características en el campo receptivo. Esta aproximación funciona muy bien en imágenes porque los píxeles tienen un orden consistente en el espacio y los píxeles cercanos tienen influencia en uno mismo. En este idea se basa el Operador de Sobel, que es un filtro con unos valores conocidos que permite la detección de bordes en imágenes. El aprendizaje profundo explora la posibilidad de aprender nuevos filtros útiles para nuestros fines.

Este concepto del campo receptivo es muy importante para entender los diseños de las CNN. El diseño de éstas debe producir que el tamaño de los canales se vaya reduciendo poco a poco a medida que se recorre la red, mientras que el número de canales aumenta. Es claro que el campo receptivo va a aumentar a medida que avanzamos en la red neuronal. A lo largo de la red se aprenden características de las imágenes. Al principio aquellas de más bajo nivel (bordes o líneas), y cada vez dicho nivel aumenta (curvas, texturas, y patrones complejos).

En el caso de las capas convolucionales 3D en nuestros modelos, estamos tratando la dimensión temporal, por lo que las características de más bajo nivel son cambios bruscos en el tiempo (lo que equivale a bordes en las dimensiones espaciales). Por

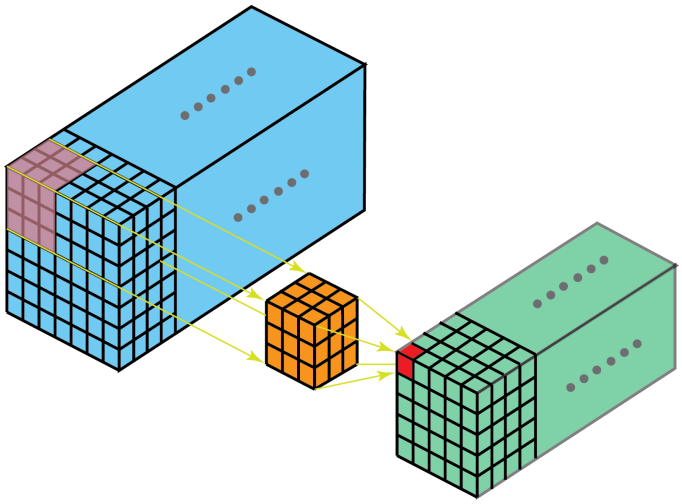

Figura 2.6: Esquema de convolución 3D sobre un canal de tamaño $7 \times 7 \times k$ con un filtro de tamaño $3 \times 3 \times 3$ con pasos $(1, 1, p_z)$ y rellenado $(0, 0, r_z)$. Fuente: `https://towardsdatascience.com/types-of-convolution-kernels-simplified-f040cb307c37`

ejemplo, en los hábitats de vegetación correspondientes a arboles de hoja caduca, si una fecha corresponde a otoño o invierno y la siguiente a primavera, este cambio es una característica que nuestra red neuronal pueda aprender para diferenciar un árbol de hoja caduca de uno de hoja perenne.

### 2.5.3. Capa de reducción

Con la finalidad de reducir el coste computacional de entrenamiento y predicción de la red se necesita reducir el tamaño espacial de las características tras la capa de convolución. Para esto se utiliza una capa de reducción junto con cada capa de convolución. En la figura 2.7 podemos observar el funcionamiento de esta capa que definimos a continuación.

**Definición 2.13** (**Capa de reducción por máximos**). *Sean $X_i \in \mathcal{M}_{l_x,l_y,l_z}(\mathbb{R})$ la i-ésima característica extraída de una convolución y sean $q_x$, $q_y$ y $q_z$ los factores de escalada a lo largo de los eje $X$, $Y$ y $Z$. Denotamos por capa de reducción por máximos, aquella cuya entrada sea $X_i$ es $Y_i = f_{reduc}(X_i) = (f_x \circ f_y \circ f_z)(X_i) = (f_z \circ f_y \circ f_x)(X_i)$, donde $f_x$ representan el reducción de tamaño de factor $q_x$ a lo largo de la primera dimensión similarmente con $f_y$ y $f_z$. Consideramos la reducción*

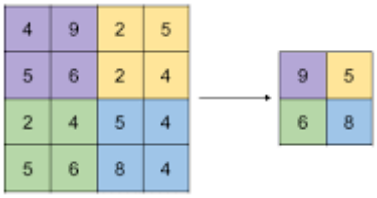


Figura 2.7: Ejemplo de capa de reducción por máximos para un canal de tamaño $4 \times 4$ con factores de escalado 2 a lo largo del eje $X$ e $Y$. Fuente: `https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/`

*de tamaño $q_x$ a lo largo de la primera dimensión,*

$$f_x : \mathcal{M}_{l_x,l_y,l_z}(\mathbb{R}) \to \mathcal{M}_{\lfloor l_x/q_x \rfloor,l_y,l_z}(\mathbb{R})$$
$$X \mapsto X'$$

*donde*

$$X'(i,j,k) = \begin{cases} \max\{X(x,j,k) : x \in \mathbb{N}, q_x(i-1)+1 \leq x < q_x \cdot i + 1\} & si\ i < \lfloor l_x/q_x \rfloor \\ \max\{X(x,j,k) : x \in \mathbb{N}, q_x(i-1)+1 \leq x \leq l_x\} & si\ i = \lfloor l_x/q_x \rfloor \end{cases}$$

*La definición en los otros dos casos es análoga.*

Las capas de reducción por máximos consiguen reducir el coste computacional debido a que reducen considerablemente el tamaño de los canales de los datos que recorren la red, sin necesidad de ajustart ningún parámetro, por lo que no tenemos que calcular el error en función de los parámetros. Existen otros tipos de capas de reducción como la reducción por media o por norma, pero no entraremos en estas alternativas pues ha quedado probado en la bibliografía científica que la reducción por máximos es la mejor opción(François, 2017).

El caso de canales en $\mathcal{M}_{l_x,l_y}(\mathbb{R})$ lo consideramos un caso particular donde $l_z = 1$ y $q_z = 1$ (esto equivale a $f_z = Id$). Llamaremos bloque de convolución a la concatenación de una capa de convolución y una capa de reducción.

### 2.5.4. Capa de normalización por lotes

Los datos de entrada al modelo se deben normalizar (Wiesler and Ney, 2011). Esto es necesario en todas las variables predictoras para que sus magnitudes sean comparables pues, en caso contrario, cuando propagamos los errores nos encontramos con situaciones donde la mayoría de error se asocia a las variables de una mayor magnitud. Esto se soluciona normalizando cada variable de los datos.

Cada vez que se actualizan los pesos la distribución de los datos de entrada de cada capa cambia ya que se han actualizado los parámetros de las capas anteriores. Esta diferencia en las distribuciones de los datos de entrada se denomina *Internal Covariate Shift* y fue introducida en Ioffe and Szegedy (2015), donde se propone la capa de normalización por lotes para solventar este problema que se combina con el entrenamiento por lotes descrito en la sección 2.6.

En el siguiente algoritmo se presenta el funcionamiento de la capa de normalización por lotes utilizado durante el entrenamiento.

---

**Algoritmo 2.1** (**Normalización por lotes**)

---

ENTRADA: *El lote* $L = \{x^1, x^2, \ldots, x^m\} \subset \mathbb{R}^n$*, los parámetros* $\gamma, \beta \in \mathbb{R}^n$.
***para*** **i** *desde* **1** *hasta* **n**:

* ***Calculamos*** *la media y varianza a lo largo de cada dimensión.*

$$\mu_i \longleftarrow \frac{1}{m}\sum_{j=1}^{m} x_i^j, \qquad \sigma_i^2 \longleftarrow \frac{1}{m}\sum_{j=1}^{m}(x_i^j - \mu_i)^2$$

***fin*** *para*
***para*** **i** *desde* **1** *hasta* **n**:
***para*** **j** *desde* **1** *hasta* **m**:

* ***Normalizamos*** *cada valor según a su dimensión.*

$$\widehat{x}_i^j \longleftarrow \frac{x_i^j - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}$$

* ***Escalamos*** *y* ***desplazamos*** *cada valor según los parámetros.*

$$y_i^j \longleftarrow \gamma_i \cdot \widehat{x}_i^j + \beta_i$$

***fin*** *para*
***fin*** *para*
SALIDA: *El lote* $\{y^1, y^2, \ldots, y^m\} \subset \mathbb{R}^n$*, con* $y^i = NL_{\gamma,\beta}(x^i)$ *para todo* $i \in \{1, 2, \ldots, m\}$

---

Las capas de normalización por lotes permiten utilizar ratios de aprendizaje superiores en el algoritmo de optimización utilizado. Además se considera una técnica de regularización pues consigue mejorar la capacidad de generalización de los modelos por introducir ruido a los datos. Esta capacidad junto con la aceleración de

la convergencia se aborda teóricamente en Luo et al. (2019) para los casos de redes neuronales convolucionales.

El valor $\epsilon$ se añade por estabilidad numérica y los valores $\gamma$ y $\beta$ son parámetros de la red que ajustaremos durante el entrenamiento. Estos valores se añaden en la definición de la capa debido a que si exclusivamente normalizásemos los datos podríamos estar acotando las representaciones de cada capa. El ejemplo que se muestra en Ioffe and Szegedy (2015) es el de normalizar los datos previos a una función de activación como la sigmoide $f(x) = \frac{1}{1+\exp(-x)}$, donde estaríamos restringiendo la capacidad de dicha función para capturar relaciones no lineales, pues la mayoría de valores de $x$ se encontrarían en el intervalo $(-1, 1)$. Con ello, si tenemos $n$ canales de entrada a la capa de normalización por lotes, ésta tendrá $2 \cdot n$ parámetros que podemos entrenar.

Este es su funcionamiento durante el entrenamiento. Cuando queremos evaluar la actuación del modelo su comportamiento cambia. Dado que no tendría sentido normalizar los datos de evaluación según su media y varianza, se almacenan la media y la varianza de cada canal de los datos de entrenamiento y estos son los valores que se usan en la capa de normalización por lotes juntos con sus parámetros ajustados durante evaluación.

### 2.5.5. Capa de aplanamiento

Esta capa se utiliza como transformación de los canales extraídos de un bloque de convolución tanto 2D como 3D que permite que los datos sean pasados como entrada a una capa densa. En la figura 2.8 se muestra un ejemplo en el caso de que los datos de entrada de la capa son $(\mathcal{M}_{4\times4}(\mathbb{R}))^1$.

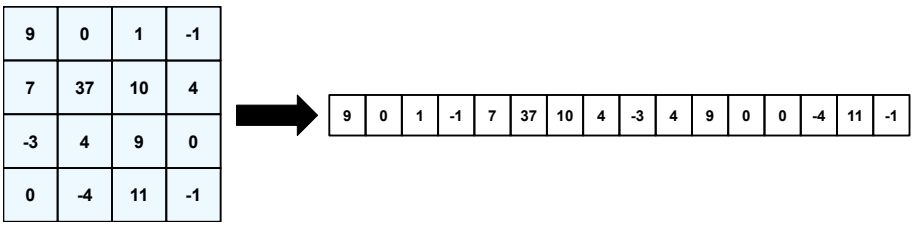


Figura 2.8: Ejemplo de capa de aplanamiento tomando como datos de entrada 1 canal de tamaño $4 \times 4$.

**Definición 2.14.** *Se denomina capa de aplanamiento a cualquier capa representada por una biyección de $X$ en $\mathbb{R}^n$, donde $n$ es la dimensión del conjunto anterior, o lo que es lo mismo en este caso, el número de neuronas del bloque de convolución anterior.*

## 2.6. Entrenamiento

Durante el entrenamiento de un modelo $f$, repetimos durante un número fijo de iteraciones el siguiente proceso. Primero, tomamos una instancia $(x, y)$, pasamos $x$ como datos de entrada al modelo, y obtenemos una predicción, $y_{pred} = f(x)$. Seguidamente, calculamos $\mathcal{L}(y_{pred}, y)$ el error de la predicción, donde $\mathcal{L}$ es la función de pérdida que debemos escoger de acuerdo a las características de nuestro modelo. Una vez obtenido este error utilizamos el algoritmo de *backpropagation* para ir asignando hacia atrás qué fracción del error es causa de cada neurona. Una vez que para cada neurona tenemos una cantidad de error, actualizamos los pesos, o parámetros de dicha neurona utilizando el algoritmo de optimización que mejor se ajuste a nuestro problema.

Consideramos que tenemos un modelo, $\Phi$, que consta de $L$ capas densas encadenadas, $\Phi = h^{(1)} \circ h^{(2)} \circ \ldots \circ h^{(L)}$ donde $\Phi : \mathbb{R}^m \to \mathbb{R}^n$, y $h^{(i)} : \mathbb{R}^{m_i} \to \mathbb{R}^{n_i}$, para todo $i \in \{1, \ldots, L\}$, de tal manera que $n_i = m_{i+1}$ para todo $i \in \{1, \ldots, L-1\}$.

Utilizamos los superíndices para referirnos a la capa a la que hace referencia los elementos y los subíndices para las coordenadas de cada vector o matriz.

$$h^{(i)} = f^{(i)} \in \mathbb{R}^{n_i}, \qquad W^{(i)} \in \mathcal{M}_{n_i,m_i}(\mathbb{R}), \qquad b^{(i)} \in \mathbb{R}^{n_i}$$

Cada capa $h^{(i)} := f^{(i)}(z^{(i)})$ donde $z^{(i)} = W^{(i)} \cdot h^{(i-1)} + b^{(i)}$ hace referencia a la salida de la neurona que tiene como función de activación $f^{(i)}$ y se define para

$$z_j^{(i)} = \left( \sum_{s=1}^{s=m_i} W_{j,s}^{(i)} \cdot h_s^{(i-1)} \right) + b_j^{(i)}, \quad \forall j \in \{1, \ldots, n_i\}$$

por lo cual tenemos que

$$\frac{\partial z_j^{(i)}}{\partial W_{j,k}^{(i)}} = h_k^{(i-1)}, \quad \frac{\partial z_j^{(i)}}{\partial b_j^{(i)}} = 1, \quad \forall i \in \{1, \ldots, L\}, j \in \{1, \ldots, n_i\}, k \in \{1, \ldots, m_i\}$$

Para cada muestra $(x, y)$ de $D_{entre}$, calculamos el error asociado $\mathcal{L} = \mathcal{L}(\Phi(x), y)$. Ahora aplicamos el algoritmo de *backpropagation* para evaluar la cantidad de error de la que es responsable cada parámetro. Para ello queremos calcular los valores de las derivadas en cada parámetro del modelo. Por simplicidad omitiremos los puntos en los que se evalúan las derivadas. Primero, para la ultima capa tenemos aplicando la regla de la cadena que,

$$\frac{\partial \mathcal{L}}{\partial W_{j,k}^{(L)}}(\Phi(x), y) = \frac{\partial \mathcal{L}}{\partial h^{(L)}} \cdot \frac{\partial h^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial z^{(L)}}{\partial W_{j,k}^{(L)}} = \delta^{(L)} \cdot e_j \cdot h_k^{(L-1)} = \delta_j^{(L)} \cdot h_k^{(L-1)}$$

$$\frac{\partial \mathcal{L}}{\partial b_j^{(L)}}(\Phi(x), y) = \frac{\partial \mathcal{L}}{\partial h^{(L)}} \cdot \frac{\partial h^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial z^{(L)}}{\partial b_j^{(L)}} = \delta^{(L)} \cdot e_j = \delta_j^{(L)}$$

donde hemos empleado los cálculos del Anexo C.1. A su vez hemos considerado $\delta_j^{(i)}$ se denomina el error imputado a la neurona $j$ de la $i$-ésima capa siendo,

$$\delta^{(i)} = \frac{\partial \mathcal{L}}{\partial h^{(i)}} \cdot \frac{\partial h^{(i)}}{\partial z^{(i)}} = \left(\delta_1^{(i)}, \delta_2^{(i)}, \ldots, \delta_{n_i}^{(i)}\right) \quad \delta_j^{(i)} = \frac{\partial \mathcal{L}}{\partial z_j^{(i)}} \tag{2.2}$$

Pero ahora podemos utilizar este valor para calcular iterativamente los valores correspondientes a la capa anterior ya que,

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial b_j^{(L-1)}} &= \frac{\partial \mathcal{L}}{\partial h^{(L)}} \cdot \frac{\partial h^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial z^{(L)}}{\partial h^{(L-1)}} \cdot \frac{\partial h^{(L-1)}}{\partial z^{(L-1)}} \cdot \frac{\partial z^{(L-1)}}{\partial b_j^{(L-1)}} = \delta^{(L)} \cdot W^{(L)} \cdot \frac{\partial h^{(L-1)}}{\partial z^{(L-1)}} \cdot e_j = \\
&= \delta^{(L)} \cdot W^{(L)} \cdot \frac{\partial h^{(L-1)}}{\partial z^{(L-1)}} \cdot e_j = \delta^{(L)} \cdot W^{(L)} \cdot e_j \cdot \frac{\partial h_j^{(L-1)}}{\partial z_j^{(L-1)}} k = \\
&= \delta^{(L)} \cdot W_{:,j}^{(L)} \cdot \frac{\partial h_j^{(L-1)}}{\partial z_j^{(L-1)}} \\
\frac{\partial \mathcal{L}}{\partial W_{j,k}^{(L-1)}} &= \frac{\partial \mathcal{L}}{\partial h^{(L)}} \cdot \frac{\partial h^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial z^{(L)}}{\partial h^{(L-1)}} \cdot \frac{\partial h^{(L-1)}}{\partial z^{(L-1)}} \cdot \frac{\partial z^{(L-1)}}{\partial W_{j,k}^{(L-1)}} = \\
&= \delta^{(L)} \cdot W^{(L)} \cdot \frac{\partial h^{(L-1)}}{\partial z^{(L-1)}} \cdot e_j \cdot h_k^{(L-2)} = \delta^{(L)} \cdot W_{:,j}^{(L)} \cdot \frac{\partial h_j^{(L-1)}}{\partial z_j^{(L-1)}} \cdot h_k^{(L-2)}
\end{aligned}$$

donde $W_{:,j}^{(i)}$ es el vector columna correspondiente a la $j$-ésima columna de $W^{(i)}$. Las ecuaciones del error imputado a una neurona 2.2 nos permiten reescribir las anteriores igualdades en función de los errores imputados a esa neurona.

$$\delta^{(L-1)} = \delta^{(L)} \cdot W^{(L)} \cdot \frac{\partial h^{(L-1)}}{\partial z^{(L-1)}} \text{ ,en particular, } \delta_j^{(L-1)} = \delta^{(L)} \cdot W_{:,j}^{(L)} \cdot \frac{\partial h_j^{(L-1)}}{\partial z_j^{(L-1)}}$$

Aplicando este procedimiento recursivamente obtenemos el algoritmo de propagación hacia atrás o *backpropagation*, que detallamos a continuación,

---

**Algoritmo 2.2** (**Propagación hacia atrás**)

---

*Calculamos* $\delta^{(L)} = (\delta_1^{(L)}, \ldots, \delta_n^{(L)})$ *donde*

$$\delta^{(L)} \leftarrow \frac{\partial \mathcal{L}}{\partial h^{(L)}} \cdot \frac{\partial h^{(L)}}{\partial z^{(L)}}$$

*Calculamos los errores asociados a los parámetros de la última capa:*

$$\frac{\partial \mathcal{L}}{\partial W_{j,k}^{(L)}} \leftarrow \delta_j^{(L)} \cdot h_k^{(L-1)}, \qquad \frac{\partial \mathcal{L}}{\partial b_j^{(L)}} \leftarrow \delta_j^{(L)}, \forall j \in \{1, \ldots, n\}$$

***para*** *cada* **i** *desde* **L** *hasta* **2***:*

* ***Propagamos*** *el error a la capa anterior:*

$$\delta^{(i-1)} \leftarrow \delta^{(i)} \cdot W^{(i)} \cdot \frac{\partial h^{(i-1)}}{\partial z^{(i-1)}}$$

* ***Obtenemos*** *la derivada en cada parámetro de la capa:*

$$\frac{\partial \mathcal{L}}{\partial W_{j,k}^{(i-1)}} \leftarrow \delta_j^{(i-1)} \cdot h_k^{(i-2)}, \qquad \frac{\partial \mathcal{L}}{\partial b_j^{(i-1)}} \leftarrow \delta_j^{(i-1)}$$

***fin*** *para*

---

En este algoritmo solo hay que tener en cuenta que cuando nos referimos a $f_k^{(0)}$ estamos haciendo referencia a $x_k$ para todo $k \in \{1, \ldots, m\}$.

Debido a que disponemos de una cantidad considerable de instancias, utilizaremos una estrategia por lotes para entrenar los modelos. Este procedimiento consiste en agrupar las instancias de entrenamiento en lotes de un menor tamaño que el número total de instancias. En vez de calcular el error y propagarlo para cada instancia, calculamos el error de todas las instancias de cada lote, para seguidamente propagarlo y actualizar los parámetros.

Los detalles de este caso y el resto de capas son presentados en el Anexo C.

## 2.7. Evaluación

El problema central en aprendizaje automático es el desempeño de nuestros modelos cuando los datos de entrada son nuevos y no han sido utilizados durante el entrenamiento. El comportamiento de un modelo en este conjunto de datos nuevos determina su capacidad de generalización. Generalmente se estima el error de generalización como el error de evaluación, esto es, el error del modelo en un conjunto de muestras que no se han usado para el entrenamiento de dicho modelo. Por ello, muchos de los esfuerzos de la modelización se centran en extrapolar información relativa de una parte del espacio de entrada a otras no observadas.

### 2.7.1. Precisión

Como métrica principal vamos a utilizar la precisión, la cual calculamos como

$$\frac{\text{Número de muestras clasificadas correctamente}}{\text{Número total de predicciones realizadas}}$$

donde para los modelos que nosotros hemos construido establecemos como categoría predicha aquella categoría que tiene mayor probabilidad de ocurrencia según la predicción hecha. Cuando nos referimos a la precisión en porcentaje es el mismo valor que el anterior multiplicado por 100.

### 2.7.2. F Valor

**Definición 2.15.** *Se llama $F\beta - valor$ a la siguiente media armónica entre la precisión y la sensibilidad,*

$$(1+\beta^2) \quad \frac{Precisión \cdot Sensibilidad}{\beta^2 Precisión + Sensibilidad}$$

La precisión de un clasificador de múltiples categorías se calcula como el número de verdaderos positivos entre el número total de positivos predichos. Por otro lado, la sensibilidad se calcula como el cociente de verdaderos positivos por la suma de verdaderos positivos y falsos negativos.

Es apropiado utilizar la precisión como métrica de evaluación cuando buscamos minimizar los falsos positivos mientras que la sensibilidad es ideal para cuando queremos minimizar los falsos negativos. Esto hace que considerar la métrica anterior sea una aproximación interesante cuando queremos combinar ambos valores en una única métrica. El caso de uso más habitual es $\beta = 1$, que proporciona una métrica en la que da el mismo peso a la precisión que a la sensibilidad. Esta es una métrica muy empleada cuando tratamos conjuntos de datos no balanceados (He and Ma, 2013).

Para la evaluación del modelo necesitaremos medidas complementarias a la precisión, como los mapas de calor o las matrices de correlación, ya que si nos fijáramos únicamente en la precisión, el desbalanceo de los datos puede sesgar ese parámetro de validación obteniendo valores altos por clasificar la clase más frecuente. Esto es tratado en más profundidad en el apartado 2.9.2.

Por ejemplo, hemos calculado la precisión de nuestros modelos de aprendizaje profundo teniendo en cuenta las 3 categorías más probables predichas a la que nos referiremos como *3-Precisión*. Esta métrica es bastante relevante debido a que en la realidad, en la región de $100m^2$ que corresponde a un píxel, se encuentran distintos hábitats de vegetación y el indicado en el muestreo es aquel dominante, por lo que considerar el segundo y tercer hábitat más probable y obtener un aumento significativo de la precisión indica que el modelo es capaz de predecir la presencia de un hábitat de manera robusta.

## 2.8. Optimización

Cuando hablamos de optimización en el ámbito del aprendizaje profundo nos referimos a la búsqueda del mínimo global sobre la superficie correspondiente al coste, esto es, el valor de la función de pérdida. En esta superficie hay múltiples mínimos locales. Aunque el objetivo es encontrar el mínimo global, cuando utilizamos redes neuronales profundas basta con encontrar mínimos con un error suficientemente pequeño que sean menos costosos computacionalmente.

### 2.8.1. Función de pérdida

En la sección 2.3 introdujimos las funciones de pérdida como herramienta para medir el desempeño de nuestros modelos. En nuestro caso nos encontramos ante un problema de clasificación multicategoría. Para un espacio de salida como el nuestro, donde queremos predecir una categoría entre $h$ posibles, tenemos la categoría a la que corresponde cada instancia de entrenamiento y evaluación, que viene representada por $y \in \{0,1\}^h$ donde todas las coordenadas son 0 a excepción de una. La posición de dicho 1 indica la categoría a la que pertenece.

**Definición 2.16.** *Sea $y_{pred} = f(x) \in [0,1]^h$ la predicción del modelo $f$ hecha para $\mathcal{X}$ y $\mathcal{Y} = \{0,1\}^h$, la función de pérdida de entropía cruzada categórica $\mathcal{L}$ que evalúa el desempeño del modelo $f$, es*

$$\begin{aligned} \mathcal{L} : \mathcal{A} \times \mathcal{Y} &\to \mathbb{R} \\ (y_{pred}, y) &\mapsto \sum_{i=1}^{h} y_i \cdot \log\left(y_{pred}(x)\right) \end{aligned}$$

Esta es la función adecuada para nuestros modelos debido a que nuestras capa de salida tiene la función de activación *Softmax* (Gómez, 2018). Por la Definición 2.8, el espacio de acción $\mathcal{A}$ de nuestro modelo va a corresponder con las probabilidades de pertenencia a cada categoría. El valor de esta función será menor cuanto mayor sea la probabilidad predicha para la categoría correcta, tomando valor 0 si se predice que la categoría correcta tiene probabilidad 1.

### 2.8.2. Optimizador Adam

En esta subsección establecemos el algoritmo de optimización utilizado en nuestras redes neuronales. Este algoritmo de optimización es el que establece la estrategia

a seguir para actualizar cada parámetro una vez que hemos calculado el coste asociado a él con el algoritmo de propagación hacia atrás. Es muy común utilizar algoritmo basados en descenso del gradiente para la optimización de los parámetros.

Estos algoritmos tratan de minimizar $\widehat{R}(f, \theta)$ actualizando los parámetros en dirección contraria al gradiente del riesgo empírico de nuestro modelo $f$ sobre el conjunto de entrenamiento $D_{entre}$, esto es, $\bigtriangledown_\theta \widehat{R}(f, \theta)$. Un parámetro común en todos estos algoritmos es el ratio de aprendizaje, $\eta$ que representa el tamaño de los pasos que tomamos, para actualizar los parámetros en función del gradiente anterior. Los ratios de aprendizaje más pequeños implican mayor tiempo de aprendizaje ya que necesitaran más pasos para alcanzar una solución óptima. Además tiene el inconveniente de que es más probable que converja a un mínimo local y no sea capaz de explorar otras combinaciones de parámetro con mínimos locales mejores. En cambio, con ratios de aprendizaje grandes tendremos el problema de que será menos probable que converja.

Una aproximación podría haber sido calcular los gradientes para todas las muestras y actualizar entonces los parámetros. Esta aproximación nos garantiza la convergencia al mínimo global en superficies convexas y mínimos locales para no convexas (Cetin et al., 1993), pero es ineficiente computacionalmente ya que estará calculado valores muy similares cuando tengamos muestras muy parecidas antes de actualizar $\theta$. Por otro lado, si actualizamos los pesos $\theta$ para cada muestra obtenemos un algoritmo mucho más rápido que la opción anterior. En cambio, al estar actualizando $\theta$ frecuentemente, provocará actualizaciones con mayor varianza y que por lo tanto causara cambios mayores el los valores de $\widehat{R}(f; \theta)$.

Como se explica en sección 2.3, vamos a utilizar un entrenamiento por lotes, que es una aproximación intermedia entre las dos anteriores, en la cual actualizamos $\theta$ para cada lote. Se puede considerar como una generalización de los dos algoritmos anteriores, considerando estos como los casos en los que consideramos los lotes de tamaño 1 y de tamaño el total del conjunto de datos de entrenamiento. Con esta aproximación reducimos la varianza en las actualizaciones de $\theta$ y podemos aprovechar las librerías de aprendizaje profundo de referencia que disponen de métodos muy eficientes para las operaciones matriciales necesarias para el cálculo de los gradientes sobre lotes, (Ruder, 2016). En Bottou (1991), se prueba que cuando no actualizamos los pesos globalmente también converge, incluso con funciones no diferenciables en todo el espacio, así como escapar de mínimos locales y encontrar mejores opciones. Para el entrenamiento dividimos el conjunto de entrenamiento $D_{entre}$ en lotes de un

tamaño similar para cada época[2] y ejecutamos el algoritmo anterior sobre todos los lotes.

Una vez consideradas el descenso del gradiente por lotes surgen varios desafíos como el hecho de que el ratio de aprendizaje es el mismo para todos los pesos y esto puede afectar a conjuntos de datos dispersos con distintas frecuencias de apariencia de sus características.

En nuestro caso, utilizaremos uno de los algoritmos estado del arte en aprendizaje profundo llamado Adam (de sus siglas en ingles *Adaptative Moment estimation*), que fue introducido en Kingma and Ba (2014) y que describimos a continuación.

---

**Algoritmo 2.3** (**Algoritmo de optimización Adam**)

---

ENTRADA: *Se toman los valores por defecto:* $\eta = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$
* ***Inicializamos*** *los valores que utilizaremos*

$$m \leftarrow 0, \qquad v \leftarrow 0, \qquad t \leftarrow 0$$

***para*** *cada época:*

* ***Tomamos*** *cíclicamente un lote* $L \subseteq D_{entre}$*:* $t \leftarrow t + 1$

***para*** *cada parámetro entrenable* $\theta$ *de nuestro red neuronal:*

* ***Ejecutamos*** *el algoritmo 2.2 para tener el coste asociado a* $\theta$*.*

$$g_\theta \leftarrow \bigtriangledown_\theta \widehat{R}_n(f, \theta) \tag{2.3}$$

* ***Actualizamos*** *y* ***corregimos*** *las estimaciones de la media y varianza.*

$$\begin{aligned} m_\theta &\leftarrow \beta_1 m_\theta + (1 - \beta_1) g_\theta \\ v_\theta &\leftarrow \beta_2 v_\theta + (1 - \beta_2) g_\theta^2 \\ \widehat{m_\theta} &\leftarrow \frac{m_\theta}{1 - \beta_1^t}, \qquad \widehat{v_\theta} \leftarrow \frac{v_\theta}{1 - \beta_2^t} \end{aligned} \tag{2.4}$$

* ***Actualizamos*** *el parámetro* $\theta$*.*

$$\theta \leftarrow \theta - \eta \frac{\widehat{m_\theta}}{\sqrt{\widehat{v_\theta}} + \epsilon} \tag{2.5}$$

***fin*** *para*

***fin*** *para*

[2]El número de épocas hace referencia al número de iteraciones que recorremos el conjunto de entrenamiento.

Este método implementa ratios de aprendizaje adaptativos para cada parámetro, para solucionar todos los problemas derivados de utilizar ratios constantes, los cuales se demuestran en Kuan and Hornik (1991). A lo largo del entrenamiento va guardando unas medias exponencialmente decrecientes de los gradientes y sus cuadrados, $m$ y $v$, calculadas en 2.4. Estos valores son estimaciones del primer y segundo momento estándar de los gradientes (la media y la varianza). Destacar que estos valores tienen un sesgo hacia el 0 ya que son iniciados con 0's. Por ello, se realiza una corrección de estos sesgos donde $\beta_1^t$ y $\beta_2^t$ son las potencias $t$-ésimas. Finalmente, actualizamos cada parámetro $\theta$ siguiendo la regla descrita en 2.5.

## 2.9. Regularización

Se conocen como técnicas de regularización al conjunto de todas las técnicas diseñadas para que los modelos no solo tengan un buen resultados en los datos de entrenamiento sino también en los de evaluación. Esto es importante debido a que el mínimo alcanzado durante el entrenamiento puede variar cuando consideremos nuevas muestras, de hecho, es lo más habitual.

En el aprendizaje profundo, la regularización está basada mayoritariamente en en la regularización de estimadores, lo cual se consigue incrementando el sesgo a cambio de reducir la varianza. Diremos que una de estas técnicas es eficiente cuando reduzca significativamente la varianza a cambio de un pequeño incremento del sesgo.

### 2.9.1. Tamaño de los lotes

Las distintas piezas del hardware obtienen mejores tiempos de ejecución para las operaciones con matrices de cierto tamaño. En los casos de las tarjetas gráficas, las cuales se usan para el entrenamiento de redes neuronales, tienen mejores tiempos de ejecución para tamaños potencias de 2. En la literatura científica se ha visto que tamaños de lotes pequeños tienen un efecto regularizador, Wilson and Martinez (2003), lo cual se puede deber al ruido añadido en el proceso de ajuste de los parámetros. Por otro lado, tamaños mayores son mejores para la paralelización del entrenamiento (Devarakonda et al., 2017).

En nuestro caso hemos utilizado lotes de 32 muestras, siguiendo los resultados obtenidos en Masters and Luschi (2018).

### 2.9.2. Técnicas de aumento de datos

Uno de los problemas más frecuentes en los modelos de aprendizaje automático es el uso de datos no balanceados para el entrenamiento de nuestro modelo. Esto es importante debido a que tener una etiqueta EUNIS más presente que el resto puede provocar un sesgo de nuestro modelo hacia dicha categoría. Este comportamiento es tratado por falacia de la frecuencia base. Si un gran porcentaje de las muestras con las que entrenamos al modelo corresponden a una única categoría, mejoras pequeñas en la precisión para clasificar dicha categoría producen reducciones significativas en la función de pérdida ya que es una suma de todas las muestras.

Un caso extremo es cuando entrenamos un modelo para detectar transacciones fraudulentas. En nuestros datos de entrada podemos tener 1000 transacciones de las cuales solo 1 es fraudulenta. Si entrenamos a un modelo es probable que acabemos teniendo un modelo que prediga siempre las transacciones como verdaderas y tenga como resultado una precisión de 99.9 %, lo que podría parecer como un gran modelo. En cambio, si tiene interés detectar las categorías menos presentes, en este caso, las transacciones fraudulentas. Por ello necesitamos hacer modificaciones en nuestro modelo.

Una aproximación puede ser modificar la función de pérdida para penalizar los falsos negativos. Sin embargo, nosotros optaremos por balancear los datos. Esto se puede conseguir por varias vías: aumentando el número de muestras de las categorías menos representadas, reduciendo el número de muestras de las categorías más representadas o combinando ambas. En nuestro problema, dado que los datos son muy limitados, no vamos a eliminar ninguna muestra y por tanto, optamos por la primera opción, el uso de técnicas de aumento de datos.

Esta es una de las técnicas más usadas dentro del aprendizaje profundo debido a que la mejor manera para que un modelo generalice mejor es entrenarlo con más datos. Aunque en la vida real la cantidad de datos que tenemos es limitada, siempre podemos crear nuevas instancias. Por eso, incluso cuando tratamos con un conjunto de datos balanceado, empleamos estas técnicas ya que un aumento de datos se asocia con mayor poder de generalización. En Bishop (1995b) se muestra que la inclusión de ruido en los datos de entrada es similar a otros métodos de regularización más utilizados como la regularización de Tikhonov.

En nuestro caso, hemos añadido hasta 2 nuevas instancias con ruido por cada muestra real para las categorías con menos presencia. Esto lo hemos hecho multiplicando cada valor $x$ de dicha instancia por una variable aleatoria con distribución normal $X \sim N(1, 0.05)$. Tras añadir estas dos muestras hemos reducido parte del

des-balanceo de los datos, pero los datos siguen sin estar balanceados completamente. Esto es porque el número de muestras estaba entre 100 y 5000 y por lo tanto, para balancear los datos deberíamos aumentar $\times 50$ alguna de las muestras. Al fin y al cabo, como lo que estamos haciendo es copias de las muestras presentes con un poco de ruido, hemos limitado este número de nuevas muestras a 2 para evitar añadir demasiado ruido y perjudicar el desempeño de nuestro modelo. Las mejoras por añadir ruido van siendo menores cada vez y el coste computacional de tratar con más muestras sigue aumentando.

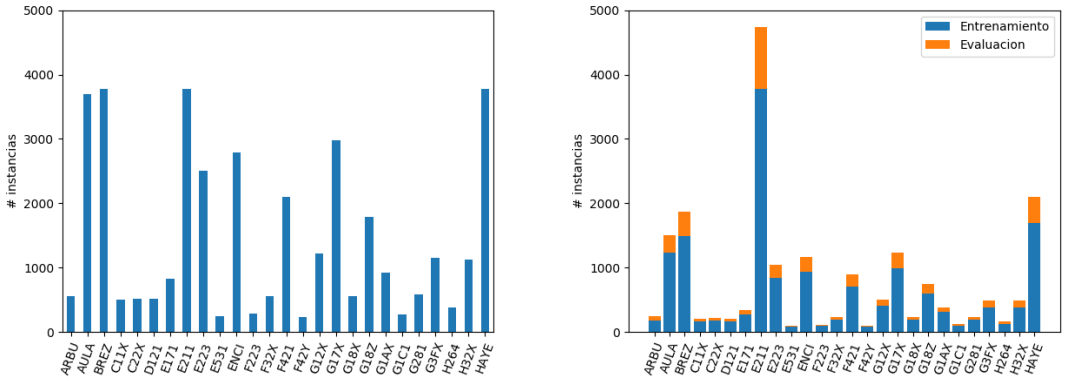


Figura 2.9: A la izquierda, distribución de categorías del conjunto de entrenamiento tras añadir muestras con ruido. A la derecha, distribución de los conjuntos de entrenamiento y evaluación.

Esta técnica es particularmente efectiva cuando tratamos con imágenes, que es nuestro caso. Hay varias maneras de crear nuevas muestras de entrenamiento. Las más comunes suelen ser rotaciones, simetrías o zooms en las imágenes. En nuestro caso no utilizaremos ninguna de estas debido a que afectan a características propias de nuestras instancias de entrenamiento. Por ejemplo, no podemos rotar las imágenes porque hay ciertos tipos de vegetación cuya aparición puede depender de su orientación geográfica. Para ilustrar esta problemática imaginémonos que queremos clasificar imágenes según la letra que muestran. No tendría sentido girar las imágenes pues estaríamos alterando características propias de las letras. Por ejemplo, la letra "b" cuando la aplicamos una simetría respecto al eje vertical pasa a ser "d".

La técnica que utilizaremos nosotros es incluir ruido en las muestras, lo que se utiliza en la mayoría de problemas desde su primera aparición en Sietsma and Dow (1991). Esta es una gran técnica para solventar los problemas de robustez al ruido que pueden tener las redes neuronales (Tang and Eliasmith, 2010). En Bishop (1995a,b) se muestra que esta técnica tiene un efecto similar a imponer términos de

penalización en la función de pérdida, aproximación muy utilizada tanto en redes neuronales como en regresiones logísticas y lineales.

Recordar que para el caso de la CNN hemos dispuesto los datos de entrada como imágenes con 108 bandas mientras en el caso 3CNN eran imágenes con 12 bandas pero las imágenes tomaban 9 posibles valores en la dimensión temporal, por lo que la cantidad de información utilizada por ambos modelos es la misma. Para el conjunto de entrenamiento se extrae aleatoriamente un 80 % del total y el de evaluación se forma con las muestras restantes. Esta partición será la misma en todos los modelos estudiados a no ser que se especifique lo contrario. En la figura 2.10 vemos que la partición mantiene su distribución espacial. En la figura 2.9 podemos ver que la distribución de categorías se mantiene en la partición y además podemos ver el balanceo que sufren los datos durante el entrenamiento.

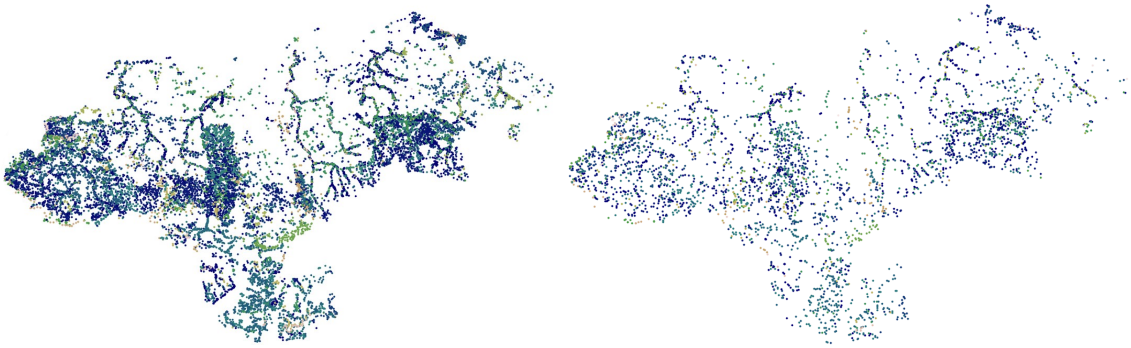

Figura 2.10: Mapas de la distribución espacial de los EUNIS en los conjuntos de entrenamiento (a la izquierda) y evaluación(a la derecha). Sigue la misma leyenda que en figura 2.1.

### 2.9.3. Capa apagado de neuronas

Esta técnica de regularización consiste en una capa que podemos añadir a nuestros modelos (Srivastava et al., 2014). Apareció como una técnica sin coste computacional que simula el entrenamiento y evaluación en modelos de agregación de modelos. Esta capa no tiene ningún efecto cuando no ejecutamos el modelo para ajustar los parámetros. Consiste en apagar una selección distinta de las neuronas en cada lote del algoritmo 2.3. Esto se consigue multiplicando por 0 la salida de un subconjunto aleatorio de las neuronas.

Por eso, el coste computacional es $O(n)$ por muestra e iteración, donde $n$ es el número de neuronas de la capa anterior, puesto que no tiene ningún parámetro y

solo debe multiplicar por 0 un parámetro con probabilidad $p$. Esto sucede cuando estamos entrenando la red. Cuando no queremos ajustar los parámetros de la red no es necesario que apaguemos ninguna neurona. En cambio, debemos multiplicar por $p$ los pesos de las neuronas de las capas en las que hemos estado apagando neuronas, con el fin de escalar dichos pesos. Con esto nos aseguramos de que el valor esperado de cada neurona es el mismo tanto si esta entrenando como si no.

Cuando empaquetamos varios modelos, estos son independientes y se entrenan en diferentes particiones del conjunto de entrenamiento. Sin embargo, en el caso de utilizar esta capa, simulamos modelos que comparte algunos parámetros y no se puede considerar que lleguen a entrenarse pues en cada iteración cambia el modelo considerado. Mientras que el primer método no es escalable para muchos modelos, el segundo consigue hacerlo con un número exponencialmente mayor que el primero.

Siguiendo la arquitectura de *GoogleLeNet*, la cual se ha tomado como arquitectura de referencia para el tratamiento de imágenes tras su presentación en Szegedy et al. (2015), cuando ganó el concurso de reconocimiento de imágenes ILSVRC 2014, hemos utilizado 0.40 como valor de $p$.

### 2.9.4. Otros

En la construcción de redes neuronales es muy común tratar con dos problemas conocidos como la explosión y el desvanecimiento del gradiente, en los cuales no profundizaremos aquí pero que se han tenido en cuenta en la construcción de la red. Estos problemas se previenen con el uso de algoritmos de optimización como *Adam* y evitando inicializar los parámetros a 0. Nosotros hemos seguido la aproximación por defecto utilizada por todos las librerías de aprendizaje profundo, llamada inicialización de *Xavier* distribuida uniformemente, que fue presentada en Glorot and Bengio (2010).

# CAPÍTULO 3

# Resultados y análisis

## 3.1. Modelización

Para la modelización de este problema hemos seguido una aproximación a nivel de píxeles, o *pixelwise*. El problema de esta implementación es que no tiene en cuenta el contexto espacial. Para solucionar este inconveniente hemos decidido seleccionar para la clasificación de cada píxel los valores del cuadrado de tamaño 11x11 píxeles cuyo centro corresponde al que queremos clasificar. Este proceso supone un incremento considerable en el coste computacional para clasificar la vegetación de toda Cantabria, pero es lo que nos permite modelar el contexto espacial de la vegetación utilizando CNNs.

## 3.2. Modelos empleados

Si bien en la presente memoria describimos en detalle los resultados obtenidos con CNNs, como se ha comentado en secciones anteriores, uno de los objetivos iniciales fue evaluar las técnicas clásicas de aprendizaje automático como paso previo a la aplicación de las CNNs. Como arquitectura de referencia con las técnicas de aprendizaje automático clásicas hemos considerado una combinación de entrenar 26 modelos que predecían la probabilidad de ocurrencia de cada hábitat y después pasamos la salida de los 26 modelos como entrada a un RF, que predecía cual era el hábitat con mayor probabilidad de ocurrencia. Para los modelos en cada hábitat hemos utilizado 2 alternativas: SVM y regresión logística. Estos modelos fueron optimizados en una fase anterior del proyecto cuando solo había 12 categorías y para este caso, simplemente se ha extendido el número de categorías sin modificar los

modelos. Los resultados en los mejores casos ronda 64 % de precisión en el conjunto de entrenamiento y 44 % en el de evaluación.

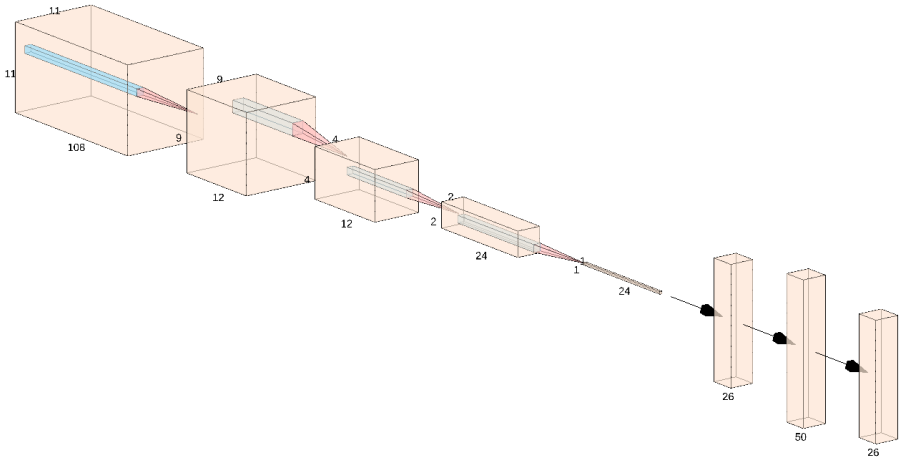


Figura 3.1: Esquema de nuestro modelo de red neuronal con convoluciones espaciales. Cada bloque representa las dimensiones de los canales y el número de ellos. El primer bloque corresponde con los datos de entrada, que son 108 canales de tamaño $11 \times 11$.

Para la construcción de la CNN hemos tomado como referencia la arquitectura *CaffeNet-5* presentada en Jia et al. (2014), adaptando los tamaños de los filtros en función del tamaño de entrada de nuestras imágenes. Esta arquitectura se considera de referencia para las tareas de clasificación de imágenes (Suzuki et al., 2016).

Hemos incorporado las capas de apagado, cuyo uso se ha visto que mejora el desempeño de las redes para reducir el sobreajuste a los datos de entrenamiento, tanto en redes con un gran número de parámetros como con conjuntos de entrenamiento muy limitados (Hinton et al., 2012; Dahl et al., 2013). Por el mismo motivo, hemos introducido las capas de normalización por lotes, ya que controlando el *Internal Covariate Shift* (visto en sección 2.5.4) evitamos sobreajustar en el entrenamiento.

Para el diseño de 3-CNN convertimos las capas convolucionales 2D y las capas de reducción 2D en capas de convolución y reducción 3D. En este caso los datos de entrada tendrán las dimensiones $11 \times 11 \times 9$ con 12 canales de salida, donde las dimensiones espaciales se reducen de la misma manera que en la CNN, mientras que la dimensión temporal sólo se reduce hasta 7. Es decir, los canales de salida tras los dos bloques de convoluciones 3D son de la forma $1 \times 1 \times 7$.

Dado que el número de canales de salida del segundo bloque es 24, tras la capa de aplanamiento tenemos un vector de dimensión 168, a diferencia del caso con convoluciones 2D, donde tras la capa de aplanamiento obtenemos un vector de dimensión 24.

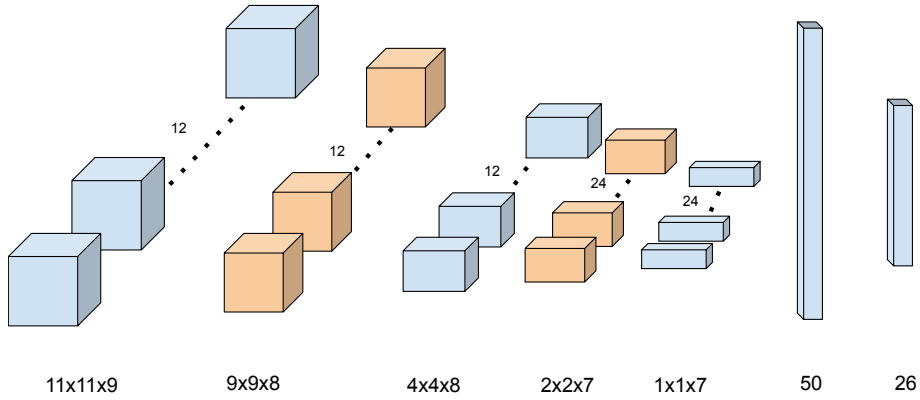


Figura 3.2: Esquema de nuestro modelo de red neuronal con convoluciones espaciales y temporales, donde cada bloque representa un canal a diferencia del esquema de CNN, donde los bloques representaban los canales a lo largo de una dimensión.

**Arquitectura CNN**:

- Capa de normalización por lotes
- Capa convolución 2D: 12 filtros de tamaño $3 \times 3$ y función de activación ReLU
- Capa de reducción 2D con factores de escalado $(2, 2)$
- Capa de apagado de neuronas, con probabilidad de apagado 0.4
- Capa convolución 2D: 12 filtros de tamaño $3 \times 3$ y función de activación ReLU
- Capa de reducción 2D con factores de escalado (2, 2)
- Capa de apagado de neuronas, con probabilidad de apagado 0.4
- Capa de normalización por lotes
- Capa de aplanamiento
- Capa densa de 50 neuronas con función de activación ReLU
- Capa de apagado de neuronas, con probabilidad de apagado 0.4
- Capa densa de 26 neuronas con función de activación Softmax

**Arquitectura 3-CNN**:

- Capa de normalización por lotes
- Capa convolución 3D: 12 filtros de tamaño $3 \times 3 \times 2$ y función de activación ReLU
- Capa de reducción 3D con factores de escalado (2, 2, 1)
- Capa de apagado de neuronas, con probabilidad de apagado 0.4
- Capa convolución 3D: 12 filtros de tamaño $3 \times 3 \times 2$ y función de activación ReLU
- Capa de reducción 3D con factores de escalado (2, 2, 1)
- Capa de apagado de neuronas, con probabilidad de apagado 0.4
- Capa de normalización por lotes
- Capa de aplanamiento
- Capa densa de 50 neuronas con función de activación ReLU
- Capa de apagado de neuronas, con probabilidad de apagado 0.4
- Capa densa de 26 neuronas con función de activación Softmax

Debido a la baja resolución temporal solo hemos optado por filtros de tamaño 2 en la dimensión temporal. Por eso, el tamaño de los datos de entradas es reducido de tamaño 9 a 7 en la dimensión temporal y los patrones temporales aprendidos serán de bajo nivel, es decir, no aprenderá patrones temporales complejos.

En virtud de las arquitecturas descritas anteriormente, la 3-CNN tiene un total de 17,660 parámetros mientras que la CNN posee un total de 17,300. El número de parámetros sirve como un indicador de referencia para la complejidad de una red ya que representa el poder de modelización que tiene una red, esto es, el número de distintas combinaciones que puede tomar. Sin embargo, hay que tener cuidado al utilizar este valor como métrica de complejidad pues pierde todo su sentido cuando cambiamos de problema o el tipo de arquitectura utilizada.

En ambas arquitecturas la mayoría de parámetros ajustables corresponden con el primer módulo de ambas, que engloba los bloques de convoluciones tanto 3D en la primera como 2D en la segunda. Esto tiene sentido pues con estas redes estamos buscando explorar el potencial de modelización de las relaciones espacio-temporales y espaciales.

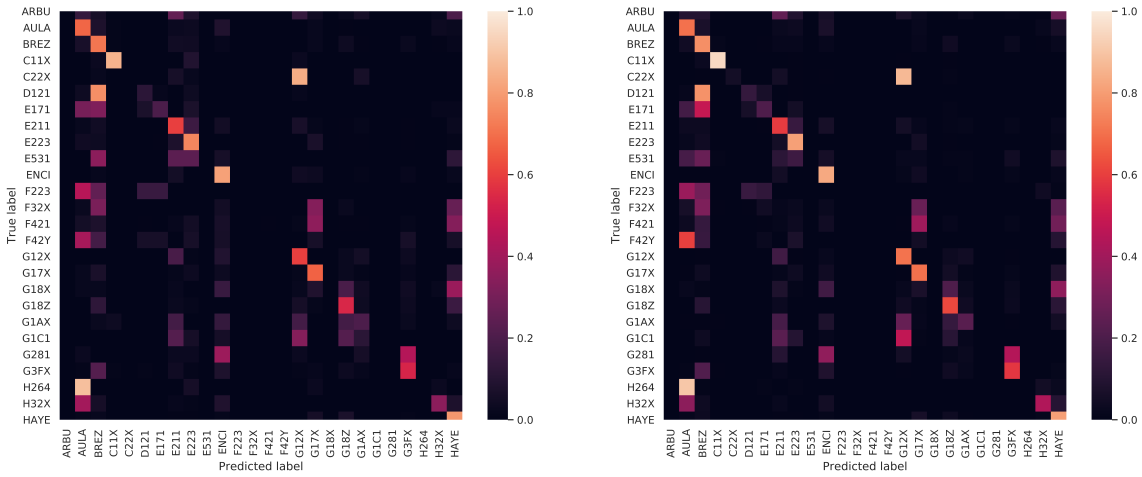


Figura 3.3: Matrices de correlación de CNN sobre el conjunto de entrenamiento en la parte izquierda (un 80 % de los puntos muestreados) y sobre el conjunto de evaluación(el 20 % restante) en la parte derecha.

La red CNN ha obtenido un 58.31 % de precisión en entrenamiento y 55.83 % en evaluación, mientras que la 3-CNN ha obtenido 59.06 % y 56.26 % respectivamente. Si bien la precisión es mejor en el caso de la 3-CNNs, la validación del modelo debe considerar otros parámetros/dimensiones de verificación. De estos datos cabe destacar la gran capacidad de generalización de la red, ya que los resultados del modelo

cuando se evalúa sobre muestras independientes es comparable a los resultados en entrenamiento. Por eso decimos que ninguna de las redes sobreajusta a los datos de entrenamiento.

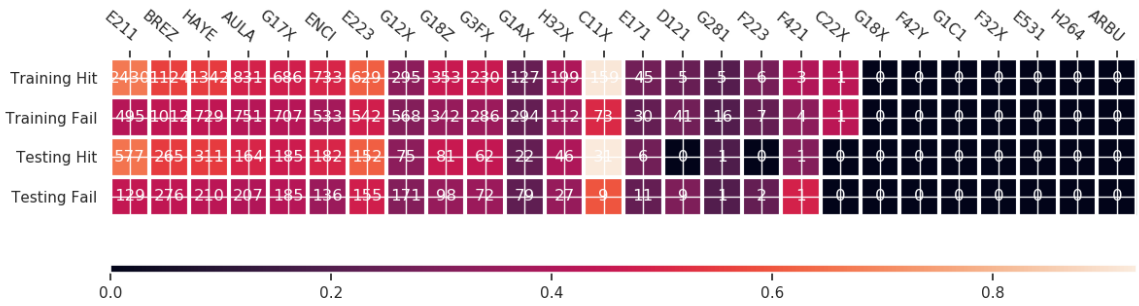


Figura 3.4: Mapa de calor de las predicciones de CNN sobre todos los puntos del conjunto de entrenamiento y de evaluación agrupados por su hábitat EUNIS. El color representa la probabilidad media de los puntos predichos con dicha categoría, y el número corresponde con las ocurrencias de dichas predicciones.

Las figuras 3.4 y 3.6 muestran los resultados de clasificación agrupados por hábitat EUNIS para las muestras de entrenamiento y evaluación, para ambos modelos. Comparando dichas figuras observamos que, además de una mejor precisión, la red 3-CNN es capaz de reconocer un mayor número de categorías durante el entrenamiento como queda reflejado por las celdas con valores no nulos de ambos mapas de calor. Observando la figura 3.7 vemos que los ratios entre el número de aciertos y fallos es uniforme a lo largo de la mayoría de categorías EUNIS en ambos modelos, salvo la categoría C11X que tiene un valor muy elevado y las muestras menos predichas por nuestro modelo. Esto nos indica que las técnicas de balanceo han sido eficaces ya que las redes no se han especializado en diferenciar las 2 o 3 categorías que son más frecuentes.

Los valores probabilísticos obtenidas concuerdan en gran medida con los ratios de acierto. Por ejemplo, para la categoría C11X, la probabilidad media cuando es la categoría predicha está muy cerca de 1.0, observando el color de sus casillas de acierto y el ratio de aciertos y fallos es cercano al 100 %. Consultando el Anexo B observamos que se corresponde con regiones con agua estancada permanentemente. La presencia de agua se detecta fácilmente a través de la banda NDVI, por lo que tiene sentido que si una categoría corresponde a regiones con agua se identifique fácilmente en nuestro modelo por la presencia de esta banda en los datos de entrada.

Las siguientes categorías con un ratio de acierto por fallos más alto son las categorías C22X, H32X y E211. La categoría C22X corresponde a flujos de agua como

ríos. Tiene sentido que identifique bien esta categoría al igual que en la categoría C11X por la presencia de la banda NDVI. En cuanto a la categoría H32X, se corresponde con acantilados, que se discriminan bien porque, con las convoluciones sobre la dimensión espacial, un acantilado supone un cambio drástico en la imagen, lo cual se corresponde con características de bajo nivel que son más fáciles de ser aprendidas por nuestra red.

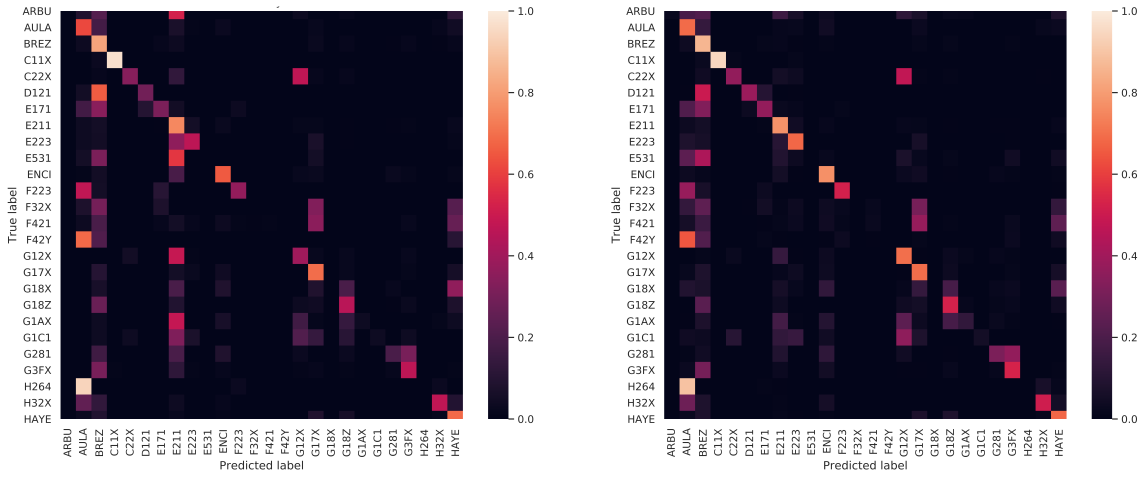


Figura 3.5: Matrices de correlación del modelo 3-CNN sobre el conjunto de entrenamiento y evaluación.

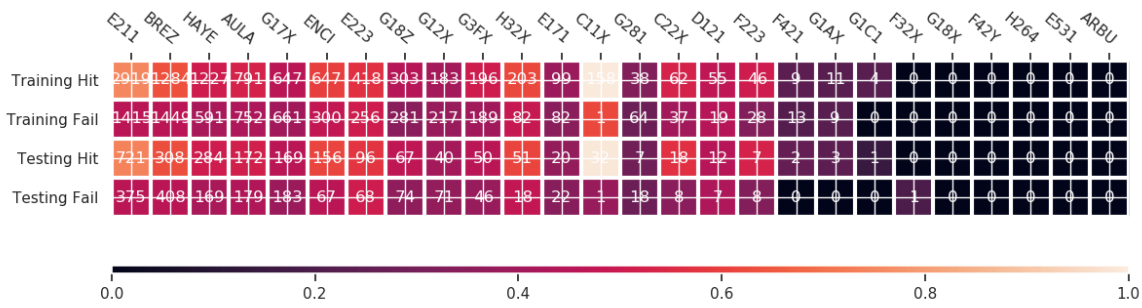


Figura 3.6: Mapa de calor de las predicciones de 3-CNN sobre todos los puntos del conjunto de entrenamiento y de evaluación agrupados por su hábitat EUNIS. El color representa la probabilidad media de los puntos predichos con dicha categoría, y el número corresponde con las ocurrencias de dichas predicciones.

El alto ratio de acierto en el caso de E211 posiblemente se deba a un sesgo del modelo para aprender más esta categoría pues es la más presente. Esto indica que

todavía hay margen de mejora en el balanceo de los datos para conseguir un modelo con mayor poder de generalización.

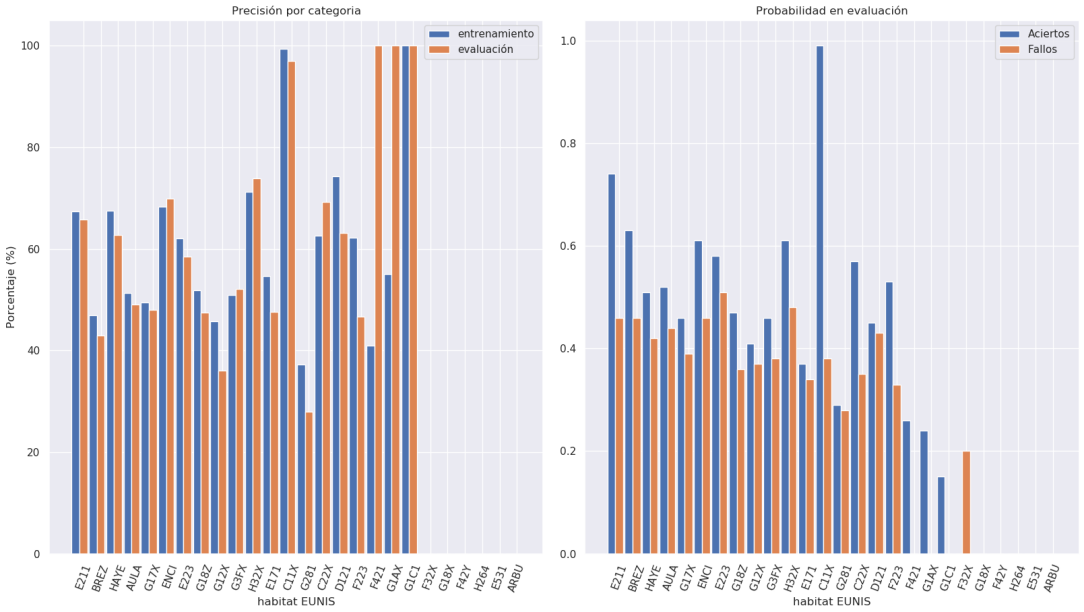


Figura 3.7: Ratios de acierto y fallo del modelo 3CNN desglosado por categoría EUNIS en las gráficas de la izquierda, mientras que en la derecha se muestran las probabilidades predichas de la categoría más probable diferenciando los aciertos y los fallos, todo ello diferenciando entre los conjuntos de entrenamiento y evaluación.

Por otro lado hay categorías como F223 o G1C1 que tienen un alto ratio de acierto, como se ve en la figura 3.7, a pesar de que el número de predicciones hechas sobre este conjunto es muy inferior al número de muestras presentes. Una posible explicación de este comportamiento es que el modelo predice una categoría similar a no ser que observe algún patrón o relación característica de este. Estas categorías corresponden con aquellas que no es capaz de encontrar la CNN.

Ahora comparemos las matrices de correlación de los modelos CNN y 3-CNN (ver en figuras 3.3 y 3.5). Lo primero que destaca de la imagen es que ambos modelos predicen los H264 y H32X como arbustos calcicolas, estos son los arbustos que crecen en un suelo calcáreo. La confusión entonces se debe a que la categoría H264 representa suelos calcáreos y la categoría H32X, acantilados interiores, que presentan una caracterización muy similar al suelo calcáreo.

Comparando las matrices de ambos modelos, observamos que la 3-CNN es capaz de aprender correctamente más categorías tanto en entrenamiento como en evaluación que la CNN (esto se ve, observando la cantidad de píxeles con valores claros en la diagonal). Similarmente, si nos fijamos en las columnas con valores más claros,

podemos observar que la red 3-CNN es capaz de predecir más categorías que la CNN, aunque observamos que en ambos casos todavía hay un sesgo de predicción hacia las categorías más frecuentes, como por ejemplo E211, AULA, BREZ, HAYE, G17X o ENCI.

Un detalle bastante relevante a cerca del valor añadido de considerar la variable temporal es para el caso en que predecimos los cursos de agua permanentes sin mareas (C22X) como llanuras de inundación ribereña mixta (G12X). Se aprecia una diferencia muy significativa en el color de estos píxeles de las matrices de CNN y 3-CNN tanto en entrenamiento como evaluación. Al considerar la variable temporal tiene mucho sentido que este error sea menos frecuente porque las llanuras de inundación no tienen presencia de agua durante todo el año, en cambio en los cursos de agua son permanentes. Luego estas dos categorías van a evolucionar a lo largo del tiempo de una manera diferente, por lo que considerar la variabilidad temporal de las bandas mejora el rendimiento del modelo en este caso.

Observando el gráfico derecho de la figura 3.7, podemos apreciar que la probabilidad media de la categoría predicha es considerablemente mayor en los casos en los que acierta que en los que falla, reflejando la capacidad de discriminación del modelo. Este resultado expresa la potencialidad de nuestro modelo para discernir entre hábitats de vegetación.

Los resultados obtenidos para la *3-Precisión* son 81.70 % y 80.30 % para la CNN, y 83.21 % y 80.65 % para la 3-CNN, en los conjuntos de entrenamiento y evaluación respectivamente. El aumento de precisión es significativo, reflejando el poder predictivo de nuestros modelos tanto cuando consideramos la variabilidad temporal como cuando no.

## 3.3. Generación de Mapa de Vegetación

Como consecuencia de todos los resultados obtenidos anteriormente se ha utilizado la 3-CNN como modelo para generar el mapa de vegetación. Una vez visto que la red no tiene problemas de sobreajuste y es capaz de generalizar para nuevas muestras independientes, hemos re-entrenado la misma 3-CNN para todos las muestras disponibles. En este caso, hemos entrenado durante 3000 épocas, obteniendo los resultados mostrados en las figuras D.1 y D.2.

Para la generación de este mapa se han considerado más de 88 millones de píxeles no nulos. Para la predicción de todos estos píxeles se ha desplegado una instancia en la nube de *DigitalOcean* de 16 CPUs (*Intel Xeon Broadwell*, 2.6 GHz) y 32 GB de memoria RAM, donde hemos separado todo el mapa en chunks o subregiones de

$250 \times 250$ píxeles y se han lanzado 8 hilos de ejecución. De esta manera ha permitido realizar las predicciones en varias regiones al mismo tiempo y reducir el tiempo total de creación del mapa de más de 2 semanas a 4 días.

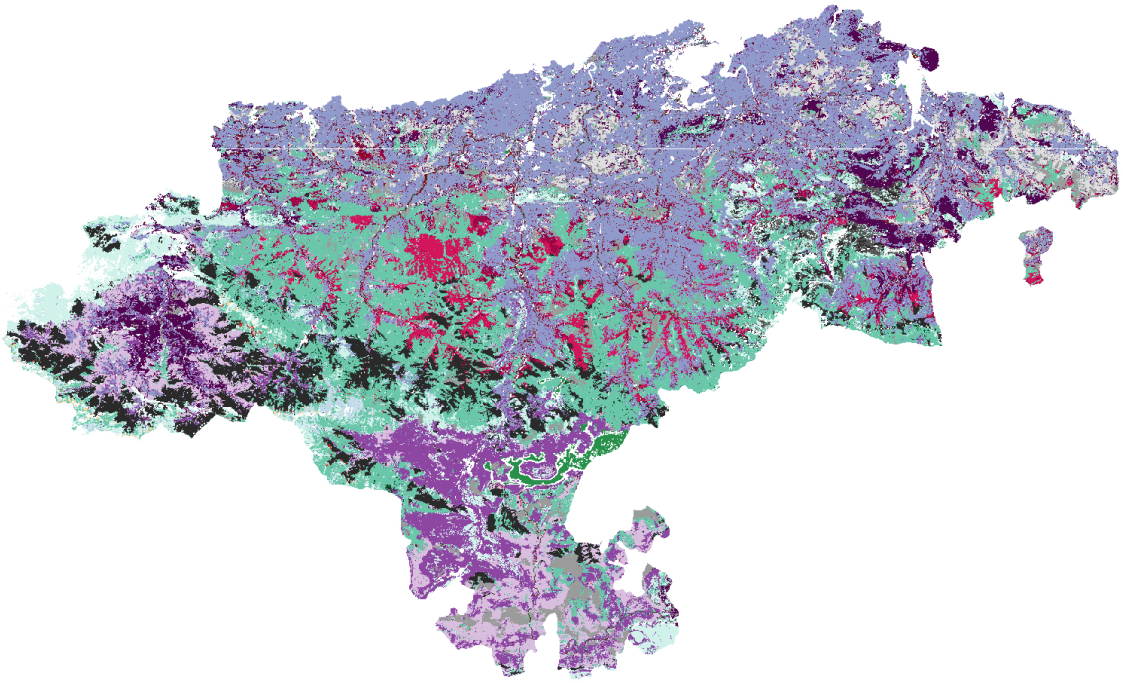

Figura 3.8: Mapa de vegetación de la Comunidad Autónoma de Cantabria a partir del modelo 3CNN con una resolución espacial de 10 metros. Cada píxel es coloreado en función de la categoría EUNIS con mayor probabilidad. La leyenda del mapa corresponde con la tabla en el Anexo B.

Tras observar el mapa, destacar que las zonas blancas corresponden con valores sin predicción que se deben a la ausencia de valor en ese píxel de alguno de las bandas utilizadas. En particular, existe una linea blanca que atraviesa prácticamente toda la región horizontalmente. Esta franja corresponde con puntos para los que no hay predicción debido a que una de las bandas de satélite (en concreto, la banda $B5$ correspondiente al 20 Agosto de 2017) tenía esa misma linea de valores ausentes. En este caso, simplemente hemos eliminado los puntos correspondientes del modelo dando lugar a esa línea horizontal en el mapa anterior. Otra aproximación podía haber sido optar por interpolar dichos valores para tener un mapa de vegetación completo, aunque hemos optado por la generación de dicho mapa sin predicciones en estos píxeles.

De la generación del mapa, lo primero que debemos fijarnos es en la capacidad

de generar un mapa coherente, lo cual se ha conseguido. Una posibilidad hubiera sido que el mapa tuviera mucho ruido, es decir, que los píxeles consecutivos tomaran valores muy distintos produciendo una imagen muy pixelada. Este efecto se consigue suprimir utilizando las convoluciones para extraer patrones espaciales.

De modo similar, nos podemos fijar en la distribución de los colores, los cuales representan categorías EUNIS. Se aprecia rápidamente que la distribución de los colores, y por tanto de los hábitats de vegetación, varía de norte a sur y de este a oeste, reflejando la diversidad de la CCAA, con diferencias climáticas y orográficas notables que dan lugar a diferencias en la vegetación de, por ejemplo, zonas costeras respecto al sur de Cantabria, los Picos de Europa o el Valle de Liébana.

Además, como es un mapa de muy alta resolución espacial, se puede hacer zoom en zonas conocidas, como en mi caso el Parque de la Naturaleza de Cabárceno o Peña Cabarga, y observar que el mapa es capaz de capturar los cambios de hábitats que se aprecian visualmente.

Otro indicación de que el mapa resultante es bueno es la precisión considerando los 3 EUNIS más probables, como se ha indicado antes, donde se alcanza un 83.89 %.

Analizando figuras D.1 y D.2, observamos que la categoría C11X, sigue teniendo un alto porcentaje de aciertos. Por otro lado la categoría F32X tiene un ratio de aciertos del 100 % ya que solo es predicha una vez.

## 3.4. *Modelando el* LiDAR

Finalmente hemos probado el uso del *LiDAR*, introducido en 2.1, para la que se han combinado las dos arquitecturas anteriores. Para cada instancia de entrenamiento tenemos, por un lado, el conjunto de valores en 9 espacios temporales de las 12 mismas bandas que antes (ver Anexo A) en una región de $11 \times 11$ píxeles y, por otro lado, los valores del *LiDAR* en dicha región. El *LiDAR* no se incluye como banda para 3-CNN porque es un sensor del que solo disponemos datos cada 5 años, por su alto coste de adquisición. Por este mismo motivo, también es importante evaluar el valor añadido que esta banda pueda aportar para los mapas de vegetación.

Para esta arquitectura hemos combinado la parte inicial de ambas arquitecturas hasta la capa de aplanamiento y después hemos concatenado los dos vectores resultantes. Se concatenan un vector de tamaño 24, que corresponde con la salida del módulo de convoluciones 2D sobre el canal del *LiDAR* y otro vector de tamaño 168, que corresponde con el módulo con convoluciones 3D. Con el vector resultante de tamaño 192 utilizamos el conjunto de capas densa y de apagado de neuronas común a las dos arquitecturas. Esta red tiene un total de 22112 parámetros ajustables don-

de 7836 son del modulo de convoluciones 3D, 3200 del módulo de convoluciones 2D sobre el *LiDAR* y 11076 de las capas densas.

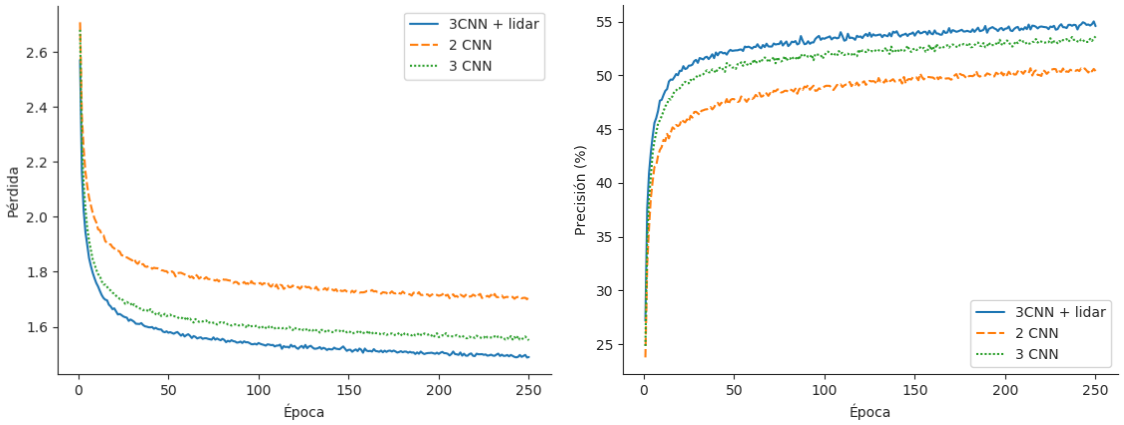


Figura 3.9: Progreso de la precisión y función de pérdida, entropía categórica cruzada, durante el entrenamiento de las distintas arquitecturas.

La figura 3.9 muestra la evolución de la función de pérdida definida en la sección 2.8.1 y la precisión de acierto, las cuales son inversamente proporcionales. Destacar que existe una diferencia entre los valores de la precisión de la gráfica y los valores finales. Esto se debe a que la precisión durante el entrenamiento es la media entre todos los lotes con todas las instancias de entrenamiento con ruido incluido. Además, este cálculo se realiza durante el entrenamiento por lo que afecta a la existencia de las capas de normalización por lotes y de apagado de neuronas, ya que hemos visto que estas tienen distinto comportamiento dependiendo de si están en fase de entrenamiento o de evaluación.

El *LiDAR* da una información de gran importancia pues permite discriminar fácilmente entre arbustos y arboles, simplemente por la altura de los mismos. La introducción de este canal en la arquitectura mejora las resultados tanto durante entrenamiento como en evaluación de las anteriores opciones, aunque parece que sufre un poco más de sobreajuste pues su precisión en el conjunto de entrenamiento es 61.78 % y en el de evaluación 58.39 %. Si para cada píxel consideramos las tres categorías más probables, obtenemos unas precisiones de 85.49 % y 83.42 %. Por último, destacar que esta arquitectura también es capaz de reconocer más categorías durante el entrenamiento y evaluación.

En la figura 3.10 vemos los valores del *LiDAR* desglosado por categorías donde se muestran el mínimo (en rojo), el máximo (en verde) y la media (en azul). La zona sombreada corresponde a la media sumándole y restándole una desviación estándar. Además, observamos la presencia de ruido. Por ejemplo, todas las categorías tienen

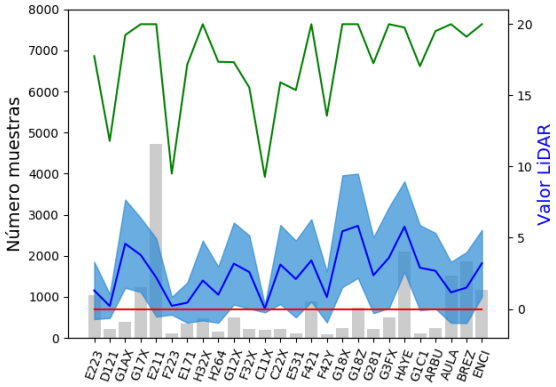


Figura 3.10: Valores del *LiDAR* desglosado por categorías.

alguna muestra con valor 0 o máximos mayores de $10m$. Esto se debe a varios factores desde la heterogeneidad en cada píxel hasta la presencia de bandadas de aves.

Cuando nos referíamos a que el *LiDAR* era útil para discriminar ciertas categorías nos referíamos a que hay diferencias significativas entre categorías. Por ejemplo, la categoría C11X que correspondía con agua estancada tiene una altura media de 0, lo que tiene sentido con lo que representa la categoría. Lo mismo sucede con los pantanos de baja altitud. En cambio, las categorías G18X y G18Z hacen referencia a zonas con predominancia de arbolado y por ello muestran un valor medio superior.

Como último experimento de evaluación, hemos dividido el conjunto por las coordenadas geográficas (tanto por latitud como longitud) y hemos comparado las diferencias de entrenar la 3-CNN en ambos conjuntos. Los resultados obtenidos muestran que los modelos no tienen suficientes muestras para el entrenamiento ya que hemos pasado de una división 80-20 a una 50-50, por lo que no tiene sentido hacer comparaciones entre divisiones.

| | Precisión(%) | 3 Precisión(%) | F1 Valor | Pérdida | Tiempo(s) |
|---|---|---|---|---|---|
| CNN | 58.31 (55.83) | 81.70 (80.30) | 0.38 (0.34) | 1.45 (1.51) | 9 |
| 3-CNN | 59.06 (56.26) | 83.21 (80.65) | 0.38 (0.35) | 1.33 (1.41) | 47 |
| 3-CNN + *LiDAR* | 61.78 (58.39) | 85.49 (83.42) | 0.42 (0.38) | 1.23 (1.34) | 50 |
| N/S (entrenando en Sur) | 62.62 (46.29) | 87.60 (68.73) | 0.47 (0.18) | 1.15 (2.11) | 22 |
| N/S (entrenando en Norte) | 62.01 (42.25) | 85.68 (66.57) | 0.37 (0.21) | 1.22 (2.16) | 22 |
| E/O (entrenado en Oeste) | 58.93 (46.67) | 85.08 (68.88) | 0.44 (0.21) | 1.27 (2.10) | 22 |
| E/O (entrenado en Este) | 66.29 (43.02) | 88.85 (69.30) | 0.48 (0.22) | 1.09 (2.14) | 22 |

Tabla 3.1: Tabla de resultados de los modelos sobre el conjuntos de entrenamiento (y entre paréntesis de evaluación). El tiempo corresponde a cada época de entrenamiento. Los modelos N/S y E/O hacem referencia al modelo 3-CNN separando las muestras equitativamente de Norte a Sur y de Este a Oeste.

# CAPÍTULO 4

# Conclusiones y discusión

## 4.1. Conclusiones

En este trabajo hemos podido ver el valor añadido que tiene 3-CNN sobre CNN para la creación de mapas de vegetación, viendo como mejora la precisión del modelo así como la capacidad de generalización. En cambio hemos sufrido un crecimiento muy significativo de los costes de computación durante el entrenamiento.

La 3-CNN ha mostrado su potencial para modelar las relaciones espacio-temporales, donde ha conseguido, con la misma cantidad de parámetros ajustables de la CNN, obtener mejores resultados en la clasificación de la vegetación en la región de Cantabria así como una mejor capacidad de generalización sobre nuevas muestras no vistas anteriormente. Estos EUNIS son predichos con baja frecuencia.

Además, se ha visto que las técnicas de balanceado han tenido un efecto positivo para el entrenamiento del modelo aunque hay presentes sesgos hacia ciertas categorías durante el aprendizaje.

Finalmente, se ha generado un mapa de vegetación de la Comunidad Autónoma de Cantabria con la 3-CNN que puede ser utilizado como herramienta para la monitorización de la distribución de los hábitats de vegetación y la gestión de políticas territoriales medioambientales debido a su capacidad para discernir cambios en los hábitats EUNIS.

Una limitación significativa del estudio es la escasa resolución temporal de los datos siendo además la mayoría de las fechas en verano, por lo que las relaciones temporales más complejas que podemos capturar están limitadas en su mayoría a esta estación.

## 4.2. Trabajos futuros

La obtención de nuevos datos permite construir nuevos modelos más complejos que son capaces de capturar relaciones más complejas. Este aumento de los datos es más necesario sobre la dimensión temporal, que es la que tiene una menor resolución.

Por ello, en futuros trabajos, se plantea replicar este estudio en zonas con mayor disponibilidad de imágenes a lo largo del año e investigar arquitecturas similares que permitan extraer patrones temporales más complejos en datos con mayor resolución temporal con el objetivo final de obtener modelos que sean capaces de aprender a reconocer todas las categorías presentes en el conjunto de entrenamiento.

Estamos asumiendo que el hábitat predominante de cada píxel no cambia durante una ventana de 3 años y que la evolución de la vegetación sigue el mismo comportamiento cada año. Podemos relajar la primera asunción y desechar la segunda considerando imágenes del mismo año y realizando un muestreo de los puntos tanto al principio como al final.

Por otro lado, el bajo número de muestras limita la complejidad de la red. Además, nos impide eliminar muestras de entrenamiento de las categorías más frecuentes por lo que solo podemos balancear los datos añadiendo nuevas muestras. Dado que estas nuevas instancias son copias con ruido de otras muestras no podemos añadir un número ilimitado por lo que aun hay margen de mejora en este aspecto.

Sería interesante evaluar los efectos de las distintas técnicas de balanceo de datos, probando con estrategias de *undersampling* para reducir el número de muestras de las clases más frecuentes a la vez que aumentamos el número de las menos frecuente(lo que hemos hecho en este trabajo), así como estudiar el punto a partir del cual el modelo empieza a empeorar cuando añadimos muestras con ruido, esto es, encontrar el ratio óptimo de creación de muestras así como la cantidad óptima de ruido añadida.

La disponibilidad de más datos permitiría el uso de un conjunto de validación para la selección de la red. En futuros trabajos se plantea estudiar el sobreajuste que podemos estar haciendo al conjunto de evaluación, pues estamos utilizando el mismo conjunto para todos los modelos y estamos tomando el que funciona mejor sobre ambos conjuntos. Resulta interesante utilizar técnicas como *K-fold* para controlar el sobreajuste a los datos de entrenamiento y evaluación.

La arquitectura que combina convoluciones 3D sobre las bandas de Sentinel y convoluciones 2D sobre el *LiDAR*, muestra potencial de mejorar a la arquitectura 3-CNN. Como aspecto negativo, esta red sufre de un mayor sobreajuste con respecto

a la 3-CNN por lo que falta ajustar la complejidad de dicha red para evitar el sobreajuste a los datos de entrenamiento, ajustando el número de capas y/o neuronas de los módulos de convoluciones.

Se plantean dos alternativas para el desarrollo de mapas más precisos. La primera consiste en utilizar los valores del *LiDAR* para separar todas las muestras en dos grupos de tal manera que se puedan separar el mayor número de categorías (observando 3.10, 4 metros parece un buen valor) y una vez tenemos las muestras separadas entrenar dos modelos distintos, utilizando únicamente las categorías presentes en cada división. La segunda manera consiste en agrupar las categorías en grupos de 4 o 5 y entrenar a una red para predecir dichos grupos. Posteriormente, entrenaríamos 1 modelo por cada grupo que nos prediga la categoría de dicho píxel dentro de las contenidas en ese grupo.

Por último, dado que las hábitats muestreadas solo representan una parte de la superficie de Cantabria y nuestro modelo esta realizando predicciones para todos los píxeles de Cantabria, es interesante establecer un procedimiento para descartar algunas predicciones. Una posible alternativa es descartar los píxeles cuando la diferencia entre la probabilidad de ocurrencia de las 2 categorías más probables difiere menos de un 10 %.

Una segunda alternativa es establecer una cota de probabilidad para cada hábitat a partir de la cual no consideramos estas predicciones. Observando la distribución de probabilidad de las predicciones de cada hábitat, si sabemos que los hábitats considerados representan un 45 % de la superficie de Cantabria, podemos ignorar las predicciones con menor probabilidad de ocurrencia hasta que nos acerquemos al 45 % de presencia considerado. Para esta última opción es imprescindible tener un modelo entrenado sobre un conjunto de datos balanceados de tal manera que la precisión por categoría sea similar y evitemos así quitar muestras únicamente de la clase que predecimos peor.

# ANEXO A

## Bandas de satélite

A continuación se desglosa el detalle de las bandas de Sentinel empleadas como variable predictoras.

| Banda | Detalle | Res. Espacial |
|---|---|---|
| B2 | Azul | $10m$ |
| B3 | Verde | $10m$ |
| B4 | Rojo | $10m$ |
| B5 | Vegetation Red Edge 1 (Re1) | $20m$ |
| B6 | Vegetation Red Edge 2 (Re2) | $20m$ |
| B7 | Vegetation Red Edge 3 (Re3) | $20m$ |
| B8 | Infrarrojo Cercano (NIR) | $10m$ |
| B8A | Infrarrojo Cercano Estrecho (NIRn) | $20m$ |
| B11 | Infrarrojo de longitud de onda corta 1 (SWIR 1) | $20m$ |
| B12 | Infrarrojo de longitud de onda corta 2 (SWIR 2) | $20m$ |
| EVI | Índice de vegetación mejorado | $10m$ |
| NDVI | Índice de vegetación de diferencia normalizada | $10m$ |

Tabla A.1: Tabla de las bandas de Sentinel 2 utilizadas como variables predictoras de nuestros modelos

Los índices de vegetación anteriores se calculan de la siguiente manera:

$$NDVI = \frac{NIR - ROJO}{NIR + ROJO}, \quad EVI = 2.5 * \frac{NIR - Rojo}{NIR + 6 * Rojo - 7.5 * Azul + 1}$$

Las bandas espectrales con resolución de 20 m se han remuestreado a 10 mediante la aplicación del modelo de corrección atmosférica (DOS3) y corrigiendo la topografía utilizando el modelo digital de terreno (DEM).

A partir de estas variables predictoras hemos obtenido las imágenes en diferentes momentos. Estas fechas con las que hemos entrenado al modelo son:

- En 2016: 16 Junio
- En 2017: 21 Junio, 16 Julio y 20 Agosto
- En 2018: 17 Abril, 5 Agosto y 4 Octubre
- En 2019: 23 Marzo y 21 Junio

El número de muestras es mayor durante el verano por las limitaciones climáticas(por ejemplo, la gran presencia de nubes) que tiene la región de Cantabria.

# ANEXO B

## Categorías EUNIS

A continuación presentamos las categorías utilizadas para entrenar nuestros modelos. Estas categorías han surgido a partir de las categorías EUNIS utilizadas para describir hábitats de vegetación y se han seguido las recomendaciones establecidas por un equipo de botánicos y ecólogos del IHCantabria y la Universidad de Oviedo, para quedarnos con las especies más representativas de la región de Cantabria. Por ejemplo, se han usado los símbolos X, Y, Z para diferenciar categorías que se consideraban demasiado generales para el caso de estudio en Cantabria con la distribución de hábitats inicial.

En esta tabla se incluyen 5 categorías como resultado de agrupar varias categorías en 1. Por ejemplo, ARBU representa los arbustos calcicolas, juntando categorías F311 y F317. Del mismo modo, la categoría BREZ engloba a los brezales juntando las categorías F412, F42X y F42Z. Por otro lado, la categoría ENCI engloba a encinares y carrascales mediterráneos agrupando F52X, G21X y G21Y. Las dos últimas categorías agrupadas corresponden con HAYE y AULA, que representa a los hayedos la primera y a los lastonares y aulagares la segunda y siendo G162, G164 y G16X, y por otro lado, E126 y F74X, las agrupaciones de la primera y la segunda respectivamente.

Estas agrupaciones se realizaron con el mismo equipo de especialistas tras una fase en la que se planteo el mismo problema con 42 hábitats. En esta fase se vio que el modelo necesitaba una mayor cantidad de datos para poder ser capaz de clasificar la mayoría de hábitats por lo que se decidió reducir el número de hábitats a clasificar sin reducir el número de muestras.

| Color | Código | Hábitat de Vegetación |
|---|---|---|
| | ARBU | Calciphyte bushes |
| | AULA | Rugh grass and Gorses |
| | BREZ | Healths |
| | C11X | Permanent oligotrophic lakes, ponds and pools X |
| | C22X | Permanent non-tidal, fast, turbulent watercourses X |
| | D121 | Hyper oceanic low-altitude blanket bogs |
| | E171 | Swards |
| | E211 | Unbroken pastures |
| | E223 | Medio-European submontane hay meadows |
| | E531 | Sub-Atlantic Pteridium aquilinum fields |
| | ENCI | Holm/Kermes Oak forests |
| | F223 | Southern Palaearctic mountain dwarfscrub |
| | F32X | Submediterranean deciduous thickets X |
| | F421 | Sub-montane [Vaccinium]-[Calluna] heaths |
| | F42Y | Dry heaths |
| | G12X | Mixed riparian floodplain and gallery woodland X |
| | G17X | Thermophilous deciduous woodland X |
| | G18X | Acidophilous [Quercus] - dominated woodland X |
| | G18Z | Acidophilous [Quercus] - dominated woodland Z |
| | G1AX | Meso- and eutrophic oak, hornbeam, ash and related woodland X |
| | G1C1 | Highly artificial forestry plantations broad leaved deciduous |
| | G281 | Eucalyptus plantations |
| | G3FX | Native conifer plantations |
| | HAYE | Beech forests |
| | H264 | Oro-Cantabrian calcareous screes |
| | H32X | Basic and ultra-basic inland cliffs X |

Tabla B.1: Tabla de las categorías consideradas en nuestro modelo

# ANEXO C

# Profundizando en el entrenamiento

## C.1. Capa densa

Tratemos con detalle las fórmulas utilizadas. Sea $i \in \{1, \dots, L\}$, consideramos la función $\mathcal{L} : \mathbb{R}^{n_L} \to \mathbb{R}$,

$$\frac{\partial \mathcal{L}}{\partial f^{(L)}} = \left( \frac{\partial \mathcal{L}}{\partial f_1^{(L)}}, \frac{\partial \mathcal{L}}{\partial f_2^{(L)}}, \dots, \frac{\partial \mathcal{L}}{\partial f_n^{(L)}} \right)$$

a su vez tenemos que $f^{(i)} : \mathbb{R}^{n_i} \to \mathbb{R}^{n_i}$ con $f^{(i)} = \sigma^{(i)}(z^{(i)})$, $\sigma^{(i)} : \mathbb{R}^{n_1} \to \mathbb{R}^{n_1}$

$$\frac{\partial f^{(i)}}{\partial z^{(i)}} = \begin{pmatrix} \frac{\partial f_1^{(i)}}{\partial z_1^{(i)}} & \frac{\partial f_1^{(i)}}{\partial z_2^{(i)}} & \cdots & \frac{\partial f_1^{(i)}}{\partial z_{n_i}^{(i)}} \\ \frac{\partial f_2^{(i)}}{\partial z_1^{(i)}} & \frac{\partial f_2^{(i)}}{\partial z_2^{(i)}} & \cdots & \frac{\partial f_2^{(i)}}{\partial z_{n_i}^{(i)}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_{n_i}^{(i)}}{\partial z_1^{(i)}} & \frac{\partial f_{n_i}^{(i)}}{\partial z_2^{(i)}} & \cdots & \frac{\partial f_{n_i}^{(i)}}{\partial z_{n_i}^{(i)}} \end{pmatrix} = \begin{pmatrix} \frac{\partial \sigma_1^{(i)}}{\partial z_1^{(i)}} & 0 & \cdots & 0 \\ 0 & \frac{\partial \sigma_2^{(i)}}{\partial z_2^{(i)}} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{\partial \sigma_{n_i}^{(i)}}{\partial z_{n_i}^{(i)}} \end{pmatrix}$$

Ahora detallaremos las derivadas de las funciones de activación que usaremos para una capa $f^{(i)} = \sigma(z^{(i)})$ con $z \in \mathbb{R}^n$. En el caso de que $\sigma$ sea $ReLU$,

$$\frac{\partial \sigma_t}{\partial z_t} = \begin{cases} 1 & \text{si } 0 < z_t \\ 0 & \text{si } 0 > z_t \end{cases}, \text{para todo } t \in \{1, \dots, n\}$$

donde la derivada de la ReLU no esta definida en 0. En el caso de que $\sigma$ sea $Softmax$,

$$\frac{\partial \sigma_t}{\partial z_t} = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \left( 1 - \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \right), \text{para todo } t \in \{1, \dots, n\}$$

Volviendo a la notación para el caso general, considerando $z^{(i)} = W^{(i)} \cdot f^{(i-1)} + b^{(i)}$, con

$$z_r^{(i)} = \left( \sum_{s=1}^{s=m_i} W_{r,s}^{(i)} \cdot f_s^{(i-1)} \right) + b_j^{(i)} \Rightarrow \frac{\partial z_r^{(i)}}{\partial f_k^{(i-1)}} = W_{r,k}^{(i)}, \quad \forall r \in \{1, \dots, n_i\}$$

luego tomando $z^{(i)}$ en función de $f^{(i-1)}$ tenemos una aplicación de $\mathbb{R}^{n_i-1} = \mathbb{R}^{m_i}$ en $\mathbb{R}^{n_i}$, por lo que obtenemos la siguiente matriz al derivar,

$$\frac{\partial z^{(i)}}{\partial f^{(i-1)}} = \begin{pmatrix} \frac{\partial z_1^{(i)}}{\partial f_1^{(i-1)}} & \frac{\partial z_1^{(i)}}{\partial f_2^{(i-1)}} & \cdots & \frac{\partial z_1^{(i)}}{\partial f_{m_i}^{(i-1)}} \\ \frac{\partial z_2^{(i)}}{\partial f_1^{(i-1)}} & \frac{\partial z_2^{(i)}}{\partial f_2^{(i-1)}} & \cdots & \frac{\partial z_2^{(i)}}{\partial f_{m_i}^{(i-1)}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial z_{n_i}^{(i)}}{\partial f_1^{(i-1)}} & \frac{\partial z_{n_i}^{(i)}}{\partial f_2^{(i-1)}} & \cdots & \frac{\partial z_{n_i}^{(i)}}{\partial f_{m_i}^{(i-1)}} \end{pmatrix} = \begin{pmatrix} W_{1,1}^{(i)} & W_{1,2}^{(i)} & \cdots & W_{1,m_i}^{(i)} \\ W_{2,1}^{(i)} & W_{2,2}^{(i)} & \cdots & W_{2,m_i}^{(i)} \\ \vdots & \vdots & \ddots & \vdots \\ W_{n_i,1}^{(i)} & W_{n_i,2}^{(i)} & \cdots & W_{n_i,m_i}^{(i)} \end{pmatrix} = W^{(i)}$$

Por último, considerando $z^{(i)}$ en función de $W_{j,k}^{(i)}$ tenemos una aplicación de $\mathbb{R}$ en $\mathbb{R}^{n_i}$,

$$\frac{\partial z^{(i)}}{\partial W_{j,k}^{(i)}} = \begin{pmatrix} \frac{\partial z_1^{(i)}}{\partial W_{j,k}^{(i)}} \\ \frac{\partial z_2^{(i)}}{\partial W_{j,k}^{(i)}} \\ \vdots \\ \frac{\partial z_{n_i}^{(i)}}{\partial W_{j,k}^{(i)}} \end{pmatrix} = e_j \cdot f_k^{(i-1)}, \quad \frac{\partial z^{(i)}}{\partial b_j^{(i)}} = \begin{pmatrix} \frac{\partial z_1^{(i)}}{\partial b_j^{(i)}} \\ \frac{\partial z_2^{(i)}}{\partial b_j^{(i)}} \\ \vdots \\ \frac{\partial z_{n_i}^{(i)}}{\partial b_j^{(i)}} \end{pmatrix} = e_j$$

donde $e_j \in R^{n_i}$ es el vector de 0's con un único 1 en la $j$-ésima posición, ya que

$$\frac{\partial z_r^{(i)}}{\partial W_{j,k}^{(i)}} = \begin{cases} f_r^{(i-1)} & \text{si } r = j \\ 0 & \text{en otro caso} \end{cases} \quad y \quad \frac{\partial z_r^{(i)}}{\partial b_j^{(i)}} = \begin{cases} 1 & \text{si } r = j \\ 0 & \text{en otro caso} \end{cases}$$

Cabe recordar que cuando $i = 1$, tenemos que $z^{(1)} = W^{(1)} \cdot X + b^{(1)}$, donde $X \in \mathbb{R}^m = \mathbb{R}^{m_1}$ por lo que tomamos $f^{(0)} = X$.

## C.2. Capa Convolución

Para las capas de convolución nos limitaremos al caso 2D, donde cada neurona es la suma ponderada de los valores de los $i_x \cdot i_y \cdot d$ píxeles de las imágenes por los pesos del filtro que aplicamos para cada canal. El único matiz diferenciador con respecto a la capa densa es que aquí, para todas las neuronas que representan píxeles de una misma capa de salida, utilizamos los mismos pesos. Siguiendo la notación anterior, si tenemos $n$ filtros distintos con tamaño $i_x, i_y$ a lo largo de los eje $X$ e $Y$, donde cada filtro es un elemento de $\left(\mathcal{M}_{i_x,i_y}(\mathbb{R})\right)^d$, entonces cada filtro tiene $i_x \cdot i_y \cdot d$ parámetros más el sesgo y, por lo tanto, el número de parámetros para entrenar la red que corresponden a la capa de convolución es $(i_x \cdot i_y \cdot d + 1) \cdot n$.

A partir de aquí, la implementación de la capa de convolución para el algoritmo de propagación hacia atrás es inmediata. Consideramos que tenemos 1 neurona por cada píxel y banda de salida. Cuando aplicamos el algoritmo de propagación hacia

atrás, se calcula el gradiente de los pesos para todos los valores pero se añaden por capas de tal manera que se ajustan los pesos una única vez en función de dicha suma.

Para ver la implementación del algoritmo de propagación hacia atrás con una capa de convolución 2D vamos a tomar $p_x = p_y = 1$ y $r_x = r_y = 0$. Siguiendo la notación de la definición 2.11 y la primera subsección de este anexo, salvo que prescindimos del superíndice de los filtros que hacen referencia a la capa, así como de la función de activación cuyo funcionamiento es descrito para la capa densa y no varia en este caso. Consideremos que la capa $l$ es una capa de convolución con $n_q$ canales de entrada, y $n_t$ de salida. Cada canal viene representada por una matriz, por eso los datos de entrada o de salida de la red convolucional que corresponden con varios canales vienen representados por una tupla de matrices.

$$f_{i,j}^{(l,t)} = \sum_{s=1}^{n_q} (f^{(l-1,s)} * K^{(t,s)})(i,j) + b^{(l)} = \sum_{s=1}^{n_q} \sum_{m=1}^{k_y} \sum_{n=1}^{k_x} K_{m,n}^{(t,s)} \cdot f_{i+m-1,j+n-1}^{(l-1,s)} + b^{(l)}$$

siendo $K^{(t)} \in \mathcal{M}_{i_x,i_y}(\mathbb{R}) \times \overset{n_q)}{\cdots} \times \mathcal{M}_{i_x,i_y}(\mathbb{R})$ la $n_q$-tupla de filtros de la $l$-ésima capa que extrae la $t$-ésimo canal, donde en $K^{(t,q)} \in \mathcal{M}_{i_x,i_y}(\mathbb{R})$ donde en primer superíndice hace referencia al canal de salida que produce y el segundo a la de entrada, mientras que en $f^{(l-1,q)} \in \mathcal{M}_{l_x,l_y}(\mathbb{R})$ el primer superíndice indica la capa a la que pertenecen y el segundo al canal de entrada, y $b^{(l)} \in \mathbb{R}$ su sesgo,

$$\begin{aligned}
\delta_{x,y}^{(l-1,q)} &= \frac{\partial \mathcal{L}}{\partial f_{x,y}^{(l-1)}} = \sum_{r=1}^{n_t} \sum_{x'=1}^{n_x} \sum_{y'=1}^{n_y} \frac{\partial \mathcal{L}}{\partial f_{x',y'}^{(l,r)}} \frac{\partial f_{x',y'}^{(l,r)}}{\partial f_{x,y}^{(l-1,q)}} = \\
&= \sum_{r=1}^{n_t} \sum_{x'=1}^{n_x} \sum_{y'=1}^{n_y} \delta_{x',y'}^{(l,r)} \frac{\partial \left( \sum_{s=1}^{n_q} \sum_{m=1}^{k_y} \sum_{n=1}^{k_x} K_{m,n}^{(r,s)} \cdot f_{x'+m-1,y'+n-1}^{(l-1,s)} + b^{(l)} \right)}{\partial f_{x,y}^{(l-1,q)}} = \\
&= \sum_{r=1}^{n_t} \sum_{x'=\text{máx}\{x-i_x+1,1\}}^{x} \sum_{y'=\text{máx}\{y-i_y+1,1\}}^{y} \delta_{x',y'}^{(l,r)} \cdot K_{x+1-x',y+1-y'}^{(r,q)} = \\
&= \sum_{r=1}^{n_t} (\delta^{(l,r)} * rot180(K^{(r,q)}))(x - i_x, y - i_y)
\end{aligned}$$

donde $rot180()$ representa la matriz rotada 180 grados.

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial K_{x,y}^{(t,q)}} &= \sum_{r=1}^{n_t}\sum_{x'=1}^{n_x}\sum_{y'=1}^{n_y}\frac{\partial \mathcal{L}}{\partial f_{x',y'}^{(l,r)}}\frac{\partial f_{x',y'}^{(l,r)}}{\partial K_{x,y}^{(t,q)}} = \\ &= \sum_{r=1}^{n_t}\sum_{x'=1}^{n_x}\sum_{y'=1}^{n_y}\delta_{x',y'}^{(l,r)}\frac{\partial\left(\sum_{s=1}^{n_q}\sum_{m=1}^{k_y}\sum_{n=1}^{k_x}K_{m,n}^{(r,s)}\cdot f_{x'+m-1,y'+n-1}^{(l-1,s)}+b^{(l)}\right)}{\partial K_{x,y}^{(t,q)}} = \\ &= \sum_{x'=1}^{n_x}\sum_{y'=1}^{n_y}\delta_{x',y'}^{(l,t)}\cdot f^{(l-1,q)}(x'+x-1,y'+y-1) = \\ &= (f^{(l-1,q)} * \delta^{(l,t)})(x,y)\end{aligned}$$

Y por último, los errores asociados a cada sesgo son,

$$\frac{\partial \mathcal{L}}{\partial b^{(l)}} = \sum_{r=1}^{n_t}\sum_{x'=1}^{n_x}\sum_{y'=1}^{n_y}\delta_{x',y'}^{(l,r)}$$

### *Capa de reducción por máximos*

En cuanto a la capa de reducción por máximos no es necesario calcular el gradiente a través de la capa pues simplemente pasamos el gradiente a la neurona cuya salida fue el máximo y el resto de neuronas obtienen gradiente 0 cuando propagamos hacia atrás. Recordar que con el gradiente nos referimos al error imputado a cada neurona. Tampoco tenemos que calcular las derivadas respecto a los pesos, puesto que esta capa no tiene ningún peso.

### *Normalización por lotes*

Por otro lado, la implementación de el algoritmo hacia atrás 2.2 para la capa de normalización por lotes es directa ya que la transformación realizada por $f = NL_{\gamma,\beta}$ es diferenciable, siguiendo la notación descrita en el algoritmo 2.1, considerando que la $k$-ésima capa corresponde con una capa de normalización por lotes, $f^{(k)} = diag(\widehat{x})\gamma + \beta$ donde $diag(x)$ representa la matriz diagonal cuya diagonal es $x$.

Por otro lado, en la sección 2.6 presentamos la estrategia de entrenamiento por lotes. La capa de normalización por lotes toma como lotes de elementos aquellos lotes en los que se ha dividido $D_{entre}$. En este caso, cuando propagamos los errores lo hacemos calculando el error o coste asociado al lote como la media de la función de pérdida para todos los valores del lote. Utilizando la notación de dicha sección tenemos,

$$C = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\Phi(x^i), y^i), \qquad \frac{\partial C}{\partial f^{(L)}} = \sum_{i=1}^{m}\frac{\partial \mathcal{L}}{\partial f^{(L)}}\left(z^{(L)}\right)$$

donde cada uno de los sumandos se puede descomponer como se hacia en el algoritmo inicial.

A continuación, se detallan los valores de los gradientes necesarios para poder implementar el algoritmo de propagación hacia atrás con este tipo de capa donde los parámetros que ajustamos son

$$\mu = \frac{1}{m}\sum_{s=1}^{m} x^s, \quad \sigma^2 = \frac{1}{m}\sum_{s=1}^{m}(x^s-\mu)^2, \quad f(x^j) = \widehat{x}^j \odot \gamma + \beta, \quad \widehat{x}^j = \widehat{x}(x^j) = \frac{x^j-\mu}{\sigma^2+\epsilon}$$

$$\frac{\partial f^{(k)}}{\partial \beta_i}\left(x^j\right) = e_i, \qquad \frac{\partial f^{(k)}}{\partial \gamma_i}\left(x^j\right) = \widehat{x}^j \cdot e_i$$

así como,

$$\frac{\partial \mu}{\partial x}(x^j) = \begin{pmatrix} \frac{\partial \mu_1}{\partial x_1} & \cdots & \frac{\partial \mu_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial \mu_n}{\partial x_1} & \cdots & \frac{\partial \mu_n}{\partial x_n} \end{pmatrix} = \begin{pmatrix} \frac{1}{m} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{1}{m} \end{pmatrix} = diag\begin{pmatrix} \frac{1}{m} \\ \vdots \\ \frac{1}{m} \end{pmatrix}, \quad \frac{\partial f^{(k)}}{\partial \widehat{x}}(x^j) = diag\begin{pmatrix} \gamma \\ \vdots \\ \gamma \end{pmatrix}$$

$$\frac{\partial \sigma^2}{\partial x}(x^j) = \begin{pmatrix} \frac{\partial \sigma_1^2}{\partial x_1} & \cdots & \frac{\partial \sigma_1^2}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial \sigma_n^2}{\partial x_1} & \cdots & \frac{\partial \sigma_n^2}{\partial x_n} \end{pmatrix} = diag\begin{pmatrix} \frac{2(x_1^j-\mu_1)}{m}\left(1-\frac{1}{m}\right) \\ \vdots \\ \frac{2(x_n^j-\mu_n)}{m}\left(1-\frac{1}{m}\right) \end{pmatrix}, \quad \frac{\partial \widehat{x}^j}{\partial x}(x^j) = diag\begin{pmatrix} \frac{(x_1^j-\mu_1)}{2(\sigma_1^2+\epsilon)} \\ \vdots \\ \frac{(x_n^j-\mu_n)}{2(\sigma_n^2+\epsilon)} \end{pmatrix}$$

por lo que podemos propagar los errores en una capa de normalización por lotes $f^{(k)}$ con $j \in \{1,\ldots,m\}$

$$\frac{\partial f^{(k)}}{\partial x}(x^j) = \frac{\partial f^{(k)}}{\partial \widehat{x}}(\widehat{x}^j)\left[\frac{\partial \widehat{x}}{\partial x}(x^j) + \frac{\partial \widehat{x}}{\partial \mu}(\mu(x^j))\frac{\partial \mu}{\partial x}(x^j) + \frac{\partial \widehat{x}}{\partial \sigma^2}(\sigma^2(x^j))\frac{\partial \sigma^2}{\partial x}(x^j)\right]$$

donde se consideran $\mu$ y $\sigma^2$ como funciones de $x^j$ y además, por simplicidad en la notación hemos mantenido $x^j$ para los elementos del lote, pero que remarcar que no pertenecen a $D_{train}$ puesto que ya han sido transformados por $k-1$ capas. De esta ecuación solo nos falta conocer dos elementos que calcularemos a continuación.

$$\frac{\partial \widehat{x}}{\partial \mu}(\mu(x^j)) = diag\begin{pmatrix} \frac{-1}{\sqrt{\sigma_1^2+\epsilon}} \\ \frac{-1}{\sqrt{\sigma_2^2+\epsilon}} \\ \vdots \\ \frac{-1}{\sqrt{\sigma_n^2+\epsilon}} \end{pmatrix}, \qquad \frac{\partial \widehat{x}}{\partial \sigma^2}(\sigma^2(x^j)) = diag\begin{pmatrix} (x_1^j-\mu_1)\frac{-1}{2}(\sigma_1^2+\epsilon)^{\frac{-3}{2}} \\ \vdots \\ (x_n^j-\mu_n)\frac{-1}{2}(\sigma_n^2+\epsilon)^{\frac{-3}{2}} \end{pmatrix}$$

## C.3. Capa de aplanamiento

Por último, la capa de aplanamiento no posee ningún parámetro que tengamos que ajustar, por lo que no hay que calcular las derivadas respectos a los pesos de dicha capa. A la hora de propagar los errores, simplemente asignamos los errores a la neuronas de la capa anterior con la que se relacionan por la biyección.

# ANEXO D

## Más resultados

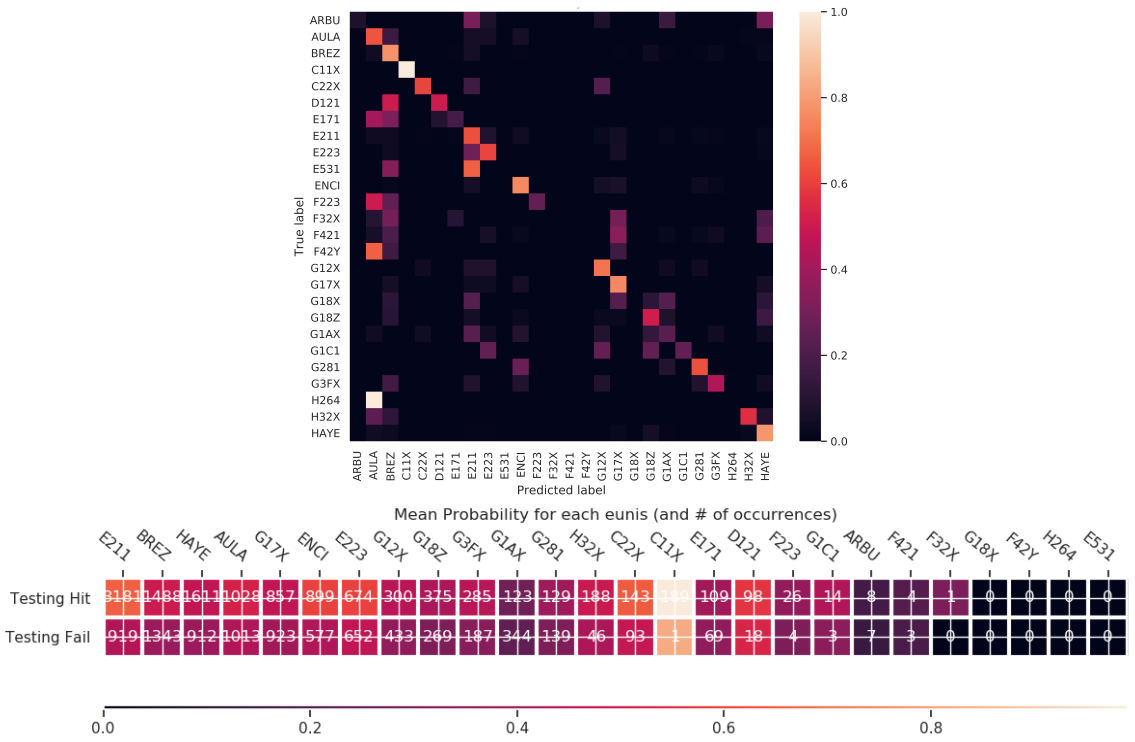


Figura D.1: Matriz de correlación y mapa de calor del modelo con 3-CNN entrenado sobre todos los puntos muestreados por los botánicos y evaluando las predicciones sobre estos mismos puntos.

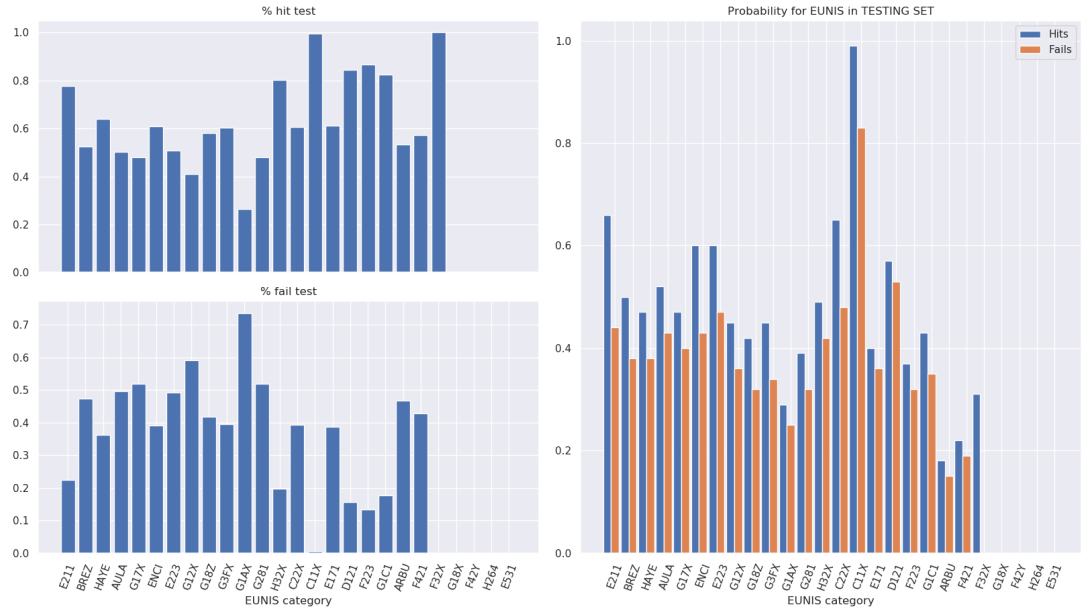


Figura D.2: Ratios de acierto y fallo del modelo 3-CNN desglosado por categoría EUNIS en las gráficas de la izquierda, mientras que en la derecha se muestra las probabilidades predichas de la categoría más probable diferenciando los aciertos y los fallos.

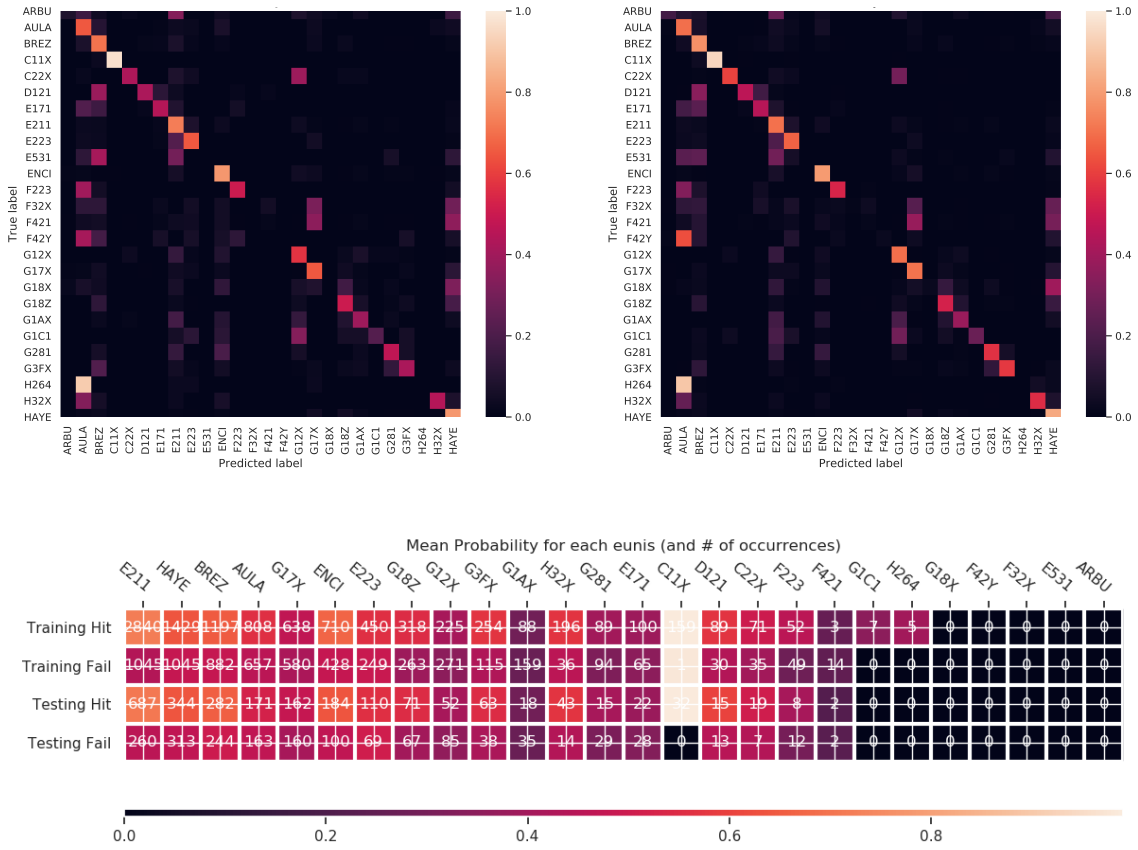


Figura D.3: Matrices de correlación y mapa de calor de la arquitectura que combina 3-CNN y CNN.

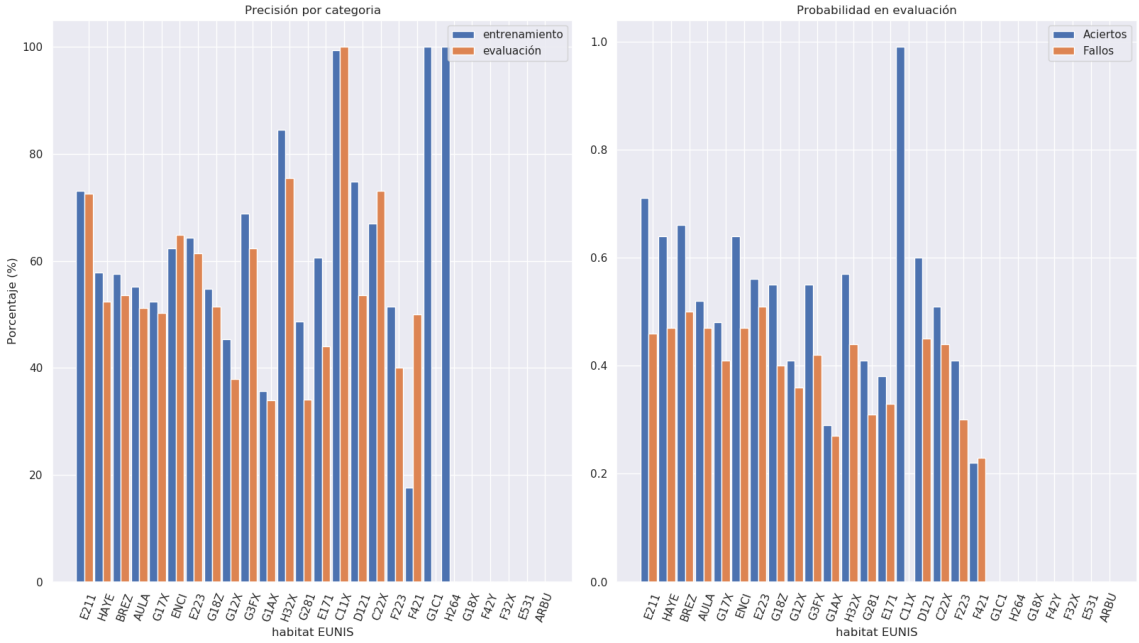


Figura D.4: Ratios de acierto y fallo (a la derecha) y precisión(a la izquierda) desglosado por hábitat EUNIS.

# Bibliografía


Bishop, C. M. (1995a). Regularization and complexity control in feed-forward networks.

Bishop, C. M. (1995b). Training with noise is equivalent to tikhonov regularization. 7(1):108–116. URL https://doi.org/10.1162/neco.1995.7.1.108.

Bottou, L. (1991). Stochastic gradient learning in neural networks. 91(8):12.

Campbell, J. B. and Wynne, R. H. (2011). *Introduction to remote sensing*. Guilford Press.

Ceinsys, T. L. (). Applications of satellite imagery & remote sensing data. urlhttp://https://www.ceinsys.com/blog/applications-of-satellite-imagery-remote-sensing-data/. Accedido 09-05-2020.

Cetin, B. C., Burdick, J. W., and Barhen, J. (1993). Global descent replaces gradient descent to avoid local minima problem in learning with artificial neural networks. In *IEEE International Conference on Neural Networks*, pp. 836–842. IEEE.

Chen, Y., Jiang, H., Li, C., Jia, X., and Ghamisi, P. (2016). Deep feature extraction and classification of hyperspectral images based on convolutional neural networks. 54(10):6232–6251.

Comission, E. (2017). Copernicus, europe eye's on earth.

Cover, T. M. (1965). Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. (3):326–334.

Dahl, G. E., Sainath, T. N., and Hinton, G. E. (2013). Improving deep neural networks for lvcsr using rectified linear units and dropout. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 8609–8613.

Davies, C. E., Moss, D., and Hill, M. O. (2004). Eunis habitat classification revised 2004. pp. 127–143.

Devarakonda, A., Naumov, M., and Garland, M. (2017). Adabatch: Adaptive batch sizes for training deep neural networks.

EUMETSAT (). Monitoring tropical cyclones from space. urlhttps://www.eumetsat.int/website/home/Images/ImageLibrary/DAT_2188665.html. Accedido 18-05-2020.

François, C. (2017). Deep learning with python.

Fuller, D. O. (2006). Tropical forest monitoring and remote sensing: A new era of transparency in forest governance? 27(1):15–29.

Garnot, V. S. F., Landrieu, L., Giordano, S., and Chehata, N. (2019). Time-space tradeoff in deep learning models for crop classification on satellite multi-spectral image time series. In *IGARSS 2019-2019 IEEE International Geoscience and Remote Sensing Symposium*, pp. 6247–6250. IEEE.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 315–323.

Gómez, R. (2018). Understanding categorical cross-entropy loss, binary cross-entropy loss, softmax loss, logistic loss, focal loss and all those confusing names.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

Hamida, A. B., Benoit, A., Lambert, P., and Amar, C. B. (2018). 3-d deep learning approach for remote sensing image classification. 56(8):4420–4434.

He, H. and Ma, Y. (2013). *Imbalanced learning: foundations, algorithms, and applications*. John Wiley & Sons.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors.

Hu, W., Huang, Y., Wei, L., Zhang, F., and Li, H. (2015). Deep convolutional neural networks for hyperspectral image classification. 2015.

Hughes, G. (1968). On the mean accuracy of statistical pattern recognizers. 14(1):55–63.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift.

Ji, S., Xu, W., Yang, M., and Yu, K. (2012). 3d convolutional neural networks for human action recognition. 35(1):221–231.

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pp. 675–678.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105.

Kuan, C.-M. and Hornik, K. (1991). Convergence of learning algorithms with constant learning rates. 2(5):484–489.

Kussul, N., Lavreniuk, M., Skakun, S., and Shelestov, A. (2017). Deep learning classification of land cover and crop types using remote sensing data. 14(5):778–782.

Li, Y., Zhang, H., and Shen, Q. (2017). Spectral–spatial classification of hyperspectral imagery with 3d convolutional neural network. 9(1):67.

Luo, P., Wang, X., Shao, W., and Peng, Z. (2019). Towards understanding regularization in batch normalization. In *International Conference on Learning Representations*. URL `https://openreview.net/forum?id=HJlLKjR9FQ`.

Ma, L., Liu, Y., Zhang, X., Ye, Y., Yin, G., and Johnson, B. A. (2019). Deep learning in remote sensing applications: A meta-analysis and review. 152:166–177.

Maggiori, E., Tarabalka, Y., Charpiat, G., and Alliez, P. (2016). Convolutional neural networks for large-scale remote-sensing image classification. 55(2):645–657.

Makantasis, K., Karantzalos, K., Doulamis, A., and Doulamis, N. (2015). Deep supervised learning for hyperspectral data classification through convolutional neural networks. In *2015 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pp. 4959–4962. IEEE.

Masters, D. and Luschi, C. (2018). Revisiting small batch training for deep neural networks.

Maturana, D. and Scherer, S. (2015). Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 922–928. IEEE.

Ruder, S. (2016). An overview of gradient descent optimization algorithms.

Sietsma, J. and Dow, R. J. (1991). Creating artificial neural networks that generalize. 4(1):67–79.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. 15(1):1929–1958.

Suzuki, S., Iida, N., Shouno, H., and Kido, S. (2016). Architecture design of deep convolutional neural network for diffuse lung disease using representation separation information. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, p. 387. The Steering Committee of The World Congress in Computer Science, Computer.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9.

Tang, Y. and Eliasmith, C. (2010). Deep networks for robust visual recognition. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pp. 1055–1062. Citeseer.

Wiesler, S. and Ney, H. (2011). A convergence analysis of log-linear training. In *Advances in Neural Information Processing Systems*, pp. 657–665.

Wilson, D. R. and Martinez, T. R. (2003). The general inefficiency of batch training for gradient descent learning. 16(10):1429–1451.

Zhu, X. X., Tuia, D., Mou, L., Xia, G.-S., Zhang, L., Xu, F., and Fraundorfer, F. (2017). Deep learning in remote sensing: A comprehensive review and list of resources. 5(4):8–36.