UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Trabajo Fin de Máster del
Master en Ingeniería y Ciencia de Datos

Optimización de redes neuronales convolucionales para la clasificación de imágenes hiperespectrales

ANDRÉS ABELARDO GARCÍA ROQUÉ

Dirigido por: JUAN MARIO HAUT HURTADO

RAFAEL PASTOR VARGAS

Curso: 2021-2022: 1ª Convocatoria

Agradecimientos

A mis amigos, sin los cuales este trabajo podría haberse acabado hace un año.

Resumen

Este proyecto abarca el estudio de las redes neuronales convolucionales aplicadas a imágenes hiperespectrales con el objetivo final de optimizar su funcionamiento, reduciendo el consumo de memoria y de electricidad para poder emplearlas a bordo de micro satélites de observación de la Tierra. Para ello se realizará un estudio del funcionamiento de estas redes y del estado del arte de la tecnología. Posteriormente se hará una revisión de la literatura para buscar métodos adecuados de optimización de este tipo de redes, tratando de reducir tanto el ancho como la profundidad de la red sin disminuir significativamente su precisión, eliminando cálculos innecesarios y por lo tanto reduciendo su consumo. Por último se aplicarán dichos métodos y se evaluarán los resultados analizando la viabilidad de uso.

Abstract

This project covers the study of convolutional neural networks applied to hyperspectral images with the ultimate goal of optimizing their performance, reducing memory and power consumption in order to use them on board Earth observation micro-satellites. To achieve this goal the principal component analysis method is applied to the convolutional layers in order to detect redundant filters and eliminate them. The obtained results are very positive, returning imrpoved architectures for the networks, saving a lot of unnecessary computations, memory and energy consumption without significantly affecting the precission of the networks.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción general y objetivos

1.1 Motivación y objetivos

La utilización de satélites pequeños (SmallSats) está siendo tendencia en las recientes misiones de observación remota de la tierra, ya que permite reducir drásticamente el coste de la misión y la complejidad del hardware. Tradicionalmente, los dispositivos FPGA (field-programmable gate array) están siendo utilizados a bordos gracias a su buen compromiso entre rendimiento y consumo de energía, pero generalmente requieren un esfuerzo significativo desde el punto de vista de diseño y programabilidad, lo que eventualmente puede limitar su capacidad práctica. En este sentido y gracias a los recientes avances en IoT, han surgido nuevas y atractivas plataformas con gran capacidad de procesamiento paralelo como son NVIDIA Xavier, TX2, etc. En este trabajo fin de máster se pretende hacer una revisión y adaptación de diferentes algoritmos de procesamiento de imágenes teledetectadas analizando su capacidad computacional, consumo energético y su posible integración en plataformas satelitales. Estas técnicas serán evaluadas sobre sobre imágenes reales obtenidas por satélite.

El objetivo general del trabajo consiste en la optimización de algoritmos de procesamiento de imágenes teledetectadas, tanto para reducir el consumo de memoria como para reducir el consumo eléctrico. Para ello, se han definido los siguientes objetivos específicos:

- Estudio de documentación y del software disponible en la literatura de teledetección para comprender las particularidades de las imágenes teledetectadas.
- Dearrollar códigos de procesamiento de imágenes teledetectadas en el framework seleccionado.
- Análisis del estado del arte de algoritmos de pruning y posibles alternativas y su aplicación a imágenes remotas.
- Proveer posibles mejoras en el procesamiento centradas en el consumo energético y de memoria.

1.2 Estructura de la memoria

La memoria de este proyecto se estructura en los siguientes capítulos:

1. Introducción general y objetivos: Comentario de la propuesta general del proyecto y los objetivos de desarrollo planteados para su realización.

2. Analisis del estado del arte: Análisis y explicación del estado del arte de las tecnologías empleadas. Particularmente de las redes neuronales convolucionales y las imágenes hiperespectrales. Adicionalmente se realiza el estudio de la aplicación de algoritmos de prunning a las redes neuronales y su viabilidad junto con el método de Principal Component Analysis aplicado a los filtros convolucionales.

3. Desarrollo: Explicación del trabajo realizado junto con el código creado y su adaptación para el uso con imágenes hiperespectrales.

4. Resultados: Explicación de los experimentos realizados, exposición y análisis de los resultados obtenidos.

5. Conclusiones y trabajos futuros: Resumen de las conclusiones obtenidas tras la realización del proyecto y propuesta de aplicaciones y futuros desarrollos para mejorar el proyecto.

Capítulo 2

Estado del arte

En este capítulo se estudia el estado del arte de las tecnologías empleadas en este trabajo. Principalmente se analizan las redes neuronales convolucionales, las imágenes hiperespectrales y la clasificación de estas imágenes mediante las redes estudiadas. Posteriormente se estudian los algoritmos de pruning para su aplicación en redes neuronales y el Principal Component Analysis para su aplicación en redes neuronales convolucionales.

2.1 Redes neuronales convolucionales

Las redes neuronales convolucionales (CNN por sus siglas en inglés) son un tipo de red neuronal comunmente empleada para la clasificación de imágenes aunque también son utilizadas para discursos o señales de audio.

Las CNN inicialmente se inspiraran en procesos biológicos replicando las neuronas de una corteza visual. Son una variación de los perceptrones multicapa muy útiles para problemas de visión artificial. Su primera aparición es en 1980 en el artículo "Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position" Fukushima (1980).

En 1988 se fueron mejoradas en el artículo "Gradient-based learning applied to document recognition" LeCun et al. (1988) donde se propone su uso para el reconocimiento de documentos escritos. En el año 2011 fueron adaptadas para su ejecución en unidades de procesamiento gráfico Ciresan et al. (2011), lo cual aceleró enormemente el entrenamiento y permitió extender su uso enormemente.

Estas redes se componen componen principalmente de tres partes:

- Capas convolucionales
- Capas de agrupación (Pooling Layers)
- Capas Fully Connected (FC)

Las dos primeras partes se introducen en el artículo Fukushima (1980)y son la base de cualquier red neuronal convolucional, aunque no es imprescindible la existencia de capas de agrupación dentro de la red para su funcionamiento.

2.1.1 Capas convolucionales

Las redes neuronales clásicas (MLP por sus siglas en inglés) para clasificar una imagen necesitan aplanar dicha imagen en un vector de tamaño Npixels x 1 y pasarla a una MLP para su clasificación. Este método es viable en los casos en los que se trate de imágenes de pocos pixeles y únicamente dos dimensiones con escala de grises (introducir referencia a MNIST). En la vida real estas condiciones son casi inexistentes y en su lugar se analizan imagenes mucho más grandes con múltiples canales para representar los colores (por ejemplo RGB). Para solucionar estos problemas se emplean las CNN.

Dentro de una capa convolucional de una CNN se capturan dependencias espacio-temporales para mejorar la clasificación mediante la aplicación de filtros convolucionales. Estos filtros consisten en la extracción de propiedades esenciales de la imagen, por ejemplo detección de bordes verticales u horizontales, colores, inclinación, etc... que aporten información útil para su clasificación. Según va aumentando la profundidad de la red aumenta la complejidad de las propiedades extraídas por los filtros. Una capa convolucional requiere de varias componentes, los cuales son datos de entrada, filtros y un mapa de activaciones como se muestra en la imagen 2.1.

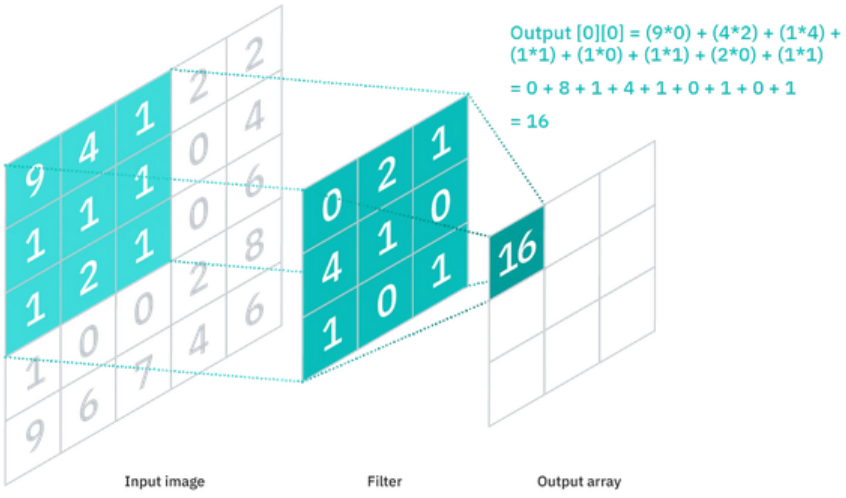

Figura 2.1: Componenetes de una capa convolucional IBM (2020)

Tras la aplicación de la convolución se obtiene el mapa de activaciones que se puede tratar como una capa parcialmente conexa, reduciendo en gran medida el número de parámetros ajustables en la red y así disminuyendo los cálculos necesarios para entrenarla.

Los filtros se aplican sobre la ímagen analizada con un método de ventana deslizante. Se

establece un tamaño de kernel para crear un filtro que irá avanzando sobre la imagen (pixel a pixel u otro valor que se determine, denominado stride). El área del filtro debe ser un número impar de manera que quede un píxel central (como se puede ver en la figura 2.3) y se genere un filtro de dimensión KxK, siendo K el ancho y alto del filtro (típicamente K=3, K=5, K=7...). A diferencia de las capas FC de un MLP, las capas convolucionales son muy versátiles. Una capa convolucional puede aceptar filtros de una, dos y tres dimensiones. Al especificar el área de un filtro también se puede especificar su profundidad creando un filtro de dimensiones KxKxq, siendo q la profundidad del filtro, de manera que afecte a múltiples canales a la vez. Por ejemplo en el caso de imágenes múltiples canales se puede crear un filtro cúbico para aplicarlo a varios canales de manera simultánea como se muestra en la figura 2.3 (derecha).

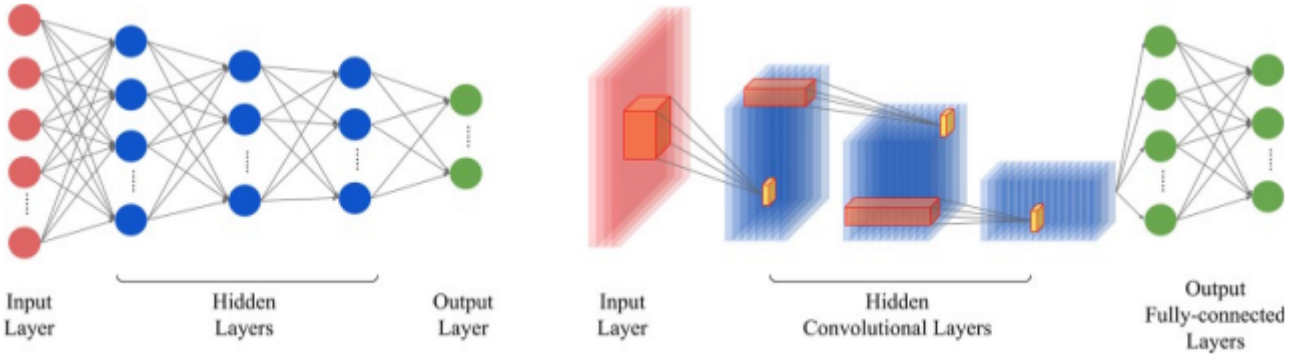

Figura 2.2: Comparación entre un MLP (izquierda) y un arquitectura convolucional de una red neuronal profunda (derecha) Paoletti et al. (2019)

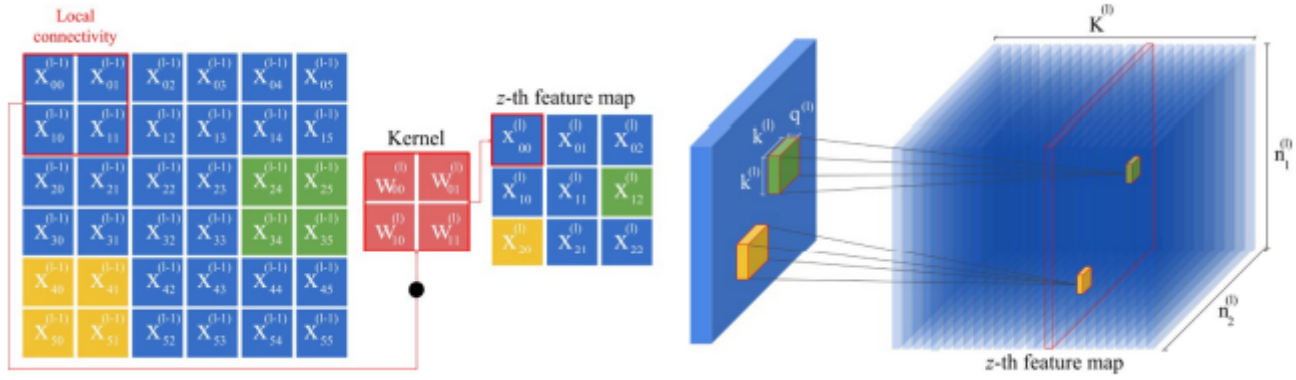

Figura 2.3: Representación gráfica de un filtro convolucional de dos dimensiones (izquierda) y de tres dimensiones (derecha) Paoletti et al. (2019)

En el caso de que los filtros excedan los bordes de la imagen se emplea una técnica conocida como Zero Padding. Esta técnica rellena los bordes externos de la imagen con ceros de manera que genera una imagen de mayor tamaño que sí es válida para las dimensiones de los filtros como se muestra en la imagen 2.3.

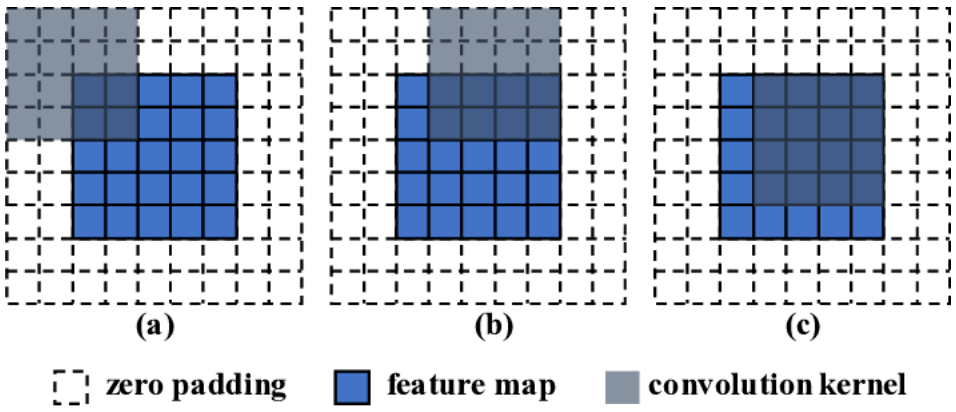

Figura 2.4: Aplicación de Zero Padding a una imagen Xu et al. (2020)

Despues de cada operación de convolución, una CNN aplica una transformación ReLU (o la que se considere más adecuada para el modelo) introduciendo así no linearidad al modelo de manera que pueda aprender representaciones no lineares de la estructura de datos.

2.1.2 Capas de agrupación

Las capas de agrupación se encargan de reducir el número de parámetros de entrada. De manera similar a las capas convolucionales, las capas de agrupación aplican un filtro que recorre la imagen de entrada para realizar una agrupación de los valores contenidos en el filtro a una nueva imagen de dimensión reducida. Hay dos técnicas principales de agrupación ilustradas en la figura 2.5:

- Max pooling: El filtro selecciona el valor más alto de la imagen y lo devuelve como salida. Es la técnica que se emplea con mayor frecuencia. Fue introducida inicialmente en 1990 por Yamaguchi (1990)

- Average pooling: El filtro computa la media de los valores obtenidos y la devuelve como salida.

Esta técnica de agrupación genera una gran pérdida de información pero se compensa por sus beneficios dentro de una CNN. Entre ellos destacan principalmente la reducción de complejidad y aumento de la eficiencia pero también limita el riesgo de sobreajuste en los datos de entrenamiento.

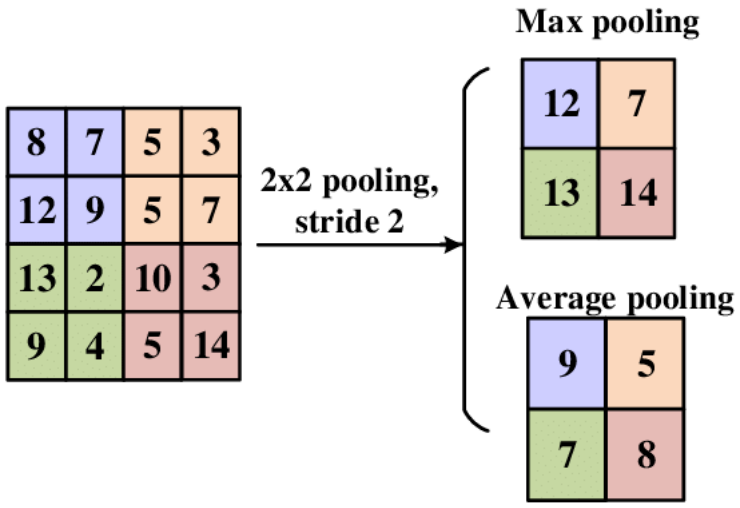

Figura 2.5: Aplicación de Max Pooling y Average Pooling Yingge et al. (2020)

2.1.3 Capa Fully Connected

Estas son las capas que se encargan de la tarea de clasificación igual que en un MLP clásico. Estas capas suelen emplear funciones Softmax como activación para devolver valores entre 0 y 1 como una probabilidad.

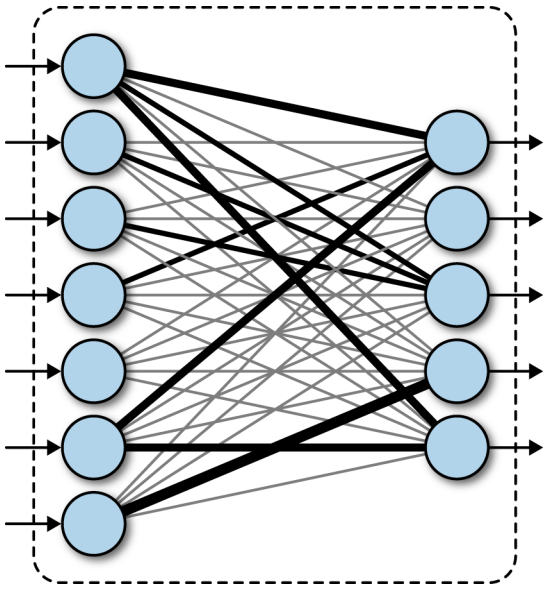

Figura 2.6: Representacion de una capa FC Ramsundar and Zadeh (2018)

2.2 Imagenes hiperespectrales

Las imágenes hiperespectrales (HSI por sus siglas en inglés) consisten en una colección de cientos de imágenes en distintas longitudes de onda para el mismo área. El ojo humano única-

mente percibe los colores mediante tres receptores distintos: Rojo, verde y azul. Las imágenes hiperespectrales miden el espectro de luz continuo para cada píxel no solamente en el espectro visible si no también en valores infra-rojos.

Las imágenes se agrupan en los llamados hipercubos, los cuales capturan los datos en dos dimensiones para representar la escena y una tercera que abarca el contenido espectral. Aparte de las imágenes hiperespectrales también existen las imágenes multiespectrales, las cuales tienen un número n de bandas (normalmente entre 3 y 10) pero no necesariamente distribuidas continuamente en una longitud de onda y contienen un número inferior de bandas que las imágenes hiperespectrales como se puede ver en la figura 2.7.

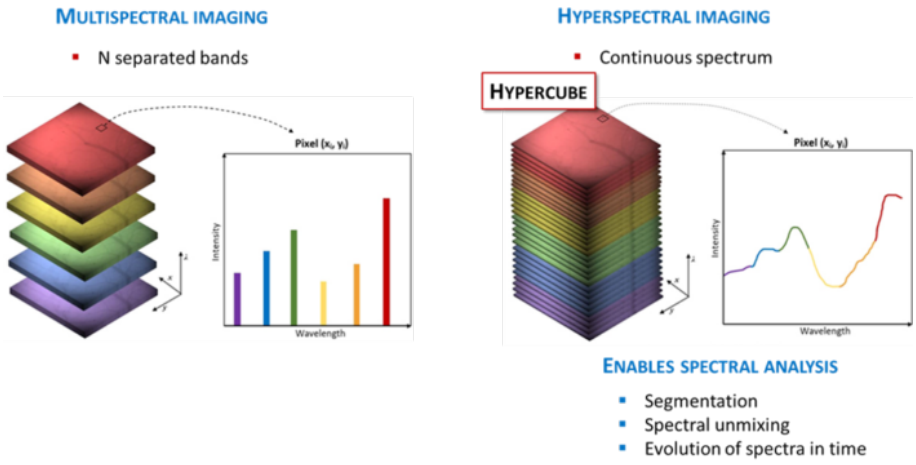

Figura 2.7: Diferencia entre una imagen multiespectral (izquierda) y una imagen hiperespectral (derecha)

Los sensores hiperespectrales que captan este tipo de imágenes y los algoritmos de procesamiento de estas son extremadamente útiles para diversos campos, entre ellos la astronomía, agricultura, imágenes biomédicas, vigilancia o explotaciones naturales Chilton (2020). Ciertos materiales tienen huellas únicas en el espectro electromagnético y una vez identificados se pueden emplear estas huellas para encontrar nuevas fuentes de este material a través de estas imágenes.

Los hipercubos se obtienen de sensores montados en satélites como el EO-1 de la NASA NASA (2000). Estos hipercubos son especialmente útiles para proyectos de observación de la tierra en los que se pretende identificar grandes áreas terrestres sin la necesidad de realizar desde tierra la captura de imágenes.

La principal desventaja de estas imágenes es el gran volumen de memoria que ocupan puesto que contienen cantidades inmensas de información comparadas con una imagen RGB típica. Para estandarizar su manejo, en 2019 se aprobó un estandar de compresión de estas imágenes mediante dos métodos, uno sin pérdidas de información ESA (2019) y otro con pérdidas reducidas Metaespectral (2021).

2.3 Redes neuronales convolucionales aplicadas a imagenes hiperespectrales

Existen varios modelos de redes neuronales profundas empleados para la clasificación de imágenes hiperespectrales: Autoencoders, Deep belief networks, recurrent neural networks y redes neuronales convolucionales. En este apartado únicamente se estudian la redes neuronales convolucionales aplicadas a la clasificación de imágenes hiperespectrales ya que es el modelo empleado para el desarrollo del trabajo.

A diferencia de los modelos mencionados anteriormente (autoencoders, Deep belief networks y recurrent neural networks) cuya base para la arquitectura son las capas FC, en las redes neuronales convolucionales la capa convolucional (CONV) es la unidad estructural básica. Estas redes integran cualidades espectrales en la información espacio-contextual de las imágenes hiperespectrales de una manera más eficiente que los otros métodos mencionados. La gran flexibilidad que proporcionan sobre la dimensionalidad de las capas y la capacidad de realizar suposiciones sobre la imágenes han convertido a las CNN uno de los modelos más populares y exitosos para la clasificación de imágenes hiperespectrales. La arquitectura de una CNN se compone de dos partes diferenciadas que se pueden interpretar como dos redes. Estas dos redes se entrenan en conjunto para optimizar todos los pesos de la CNN.

- La primera red consiste en la extracción de características. De esto se encargan los filtros convolucionales que además aprenden representaciones a alto nivel de las imágenes.

- La segunda red consiste en una serie de capas FC que computan la tarea de clasificación.

2.3.1 Modelos espectrales para el análisis de imágenes hiperespectrales

Los modelos espectrales consideran los pertenecientes a cada capa como los datos de entrada. Se puede considerar un número reducido de capas si este es muy elevado. Para obtener las capas más relevantes se aplica Principal Component Analysis u otro método de reducción de datos. A estos datos se le aplican filtros unidimensionales (CNN1D) en cada capa convolucional.

2.3.2 Modelos espaciales para el análisis de imágenes hiperespectrales

Los modelos espaciales únicamente consideran la información obtenida del cubo de datos de imágenes hiperespectrales. En estos casos también se emplea el Principal Component Anañysis para reducir el número de dimensiones con las que se va a trabajar. Normalmente se emplean

arquitecturas con filtros bidimensionales (CNN2D) para procesar la información espacial y posteriormente se obtienen los píxeles vecinos al pixel que se quiere clasificar con una dimensión dxd.

2.3.3 Modelos espectro-espaciales para el análisis de imágenes hiperespectrales

Estos modelos tienen en cuenta tanto las propiedades espectrales como las espaciales del cubo de datos de imágenes hiperespectrales. Gracias a la flexibilidad de las CNN existen varias estrategias y arquitecturas para realizar el procesado espacio-espectral.

- La arquitectura CNN1D puede ser empleada reagrupando la información espacial y concatenándola con las propiedades espectrales.(fig 2.8)
- La arquitectura CNN2D puede realizar el procesado espectro-espacial de diferentes maneras. Una de ellas es directamente proporcionar al modelo de regiones tridimensionales de tamaño d x d x Ncanales donde Ncanales puede ser el número de canales contenidos en el cubo de datos o un número concreto de canales obtenidos tras un PCA u otro método de reducción de datos. (fig 2.9)
- Adicionalmente a los modelos CNN1D y CNN2D existe el modelo CNN3D para la clasificación espectro-espacial.donde los filtros adoptan una forma tetradimensional de la forma k x k x k x q capaces de extraer cualidades espectro-espaciales de alto nivel de una manera natural. (fig 2.10)

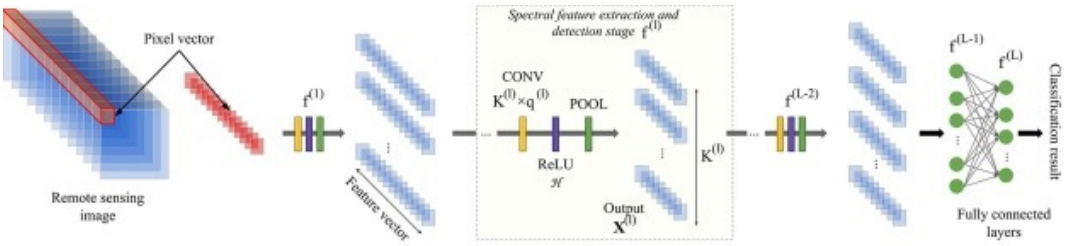

Figura 2.8: Arquitectura CNN1D Paoletti et al. (2019)

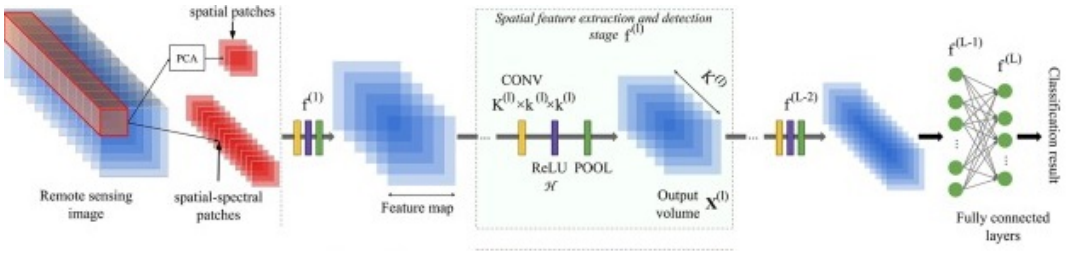

Figura 2.9: Arquitectura CNN2D Paoletti et al. (2019)

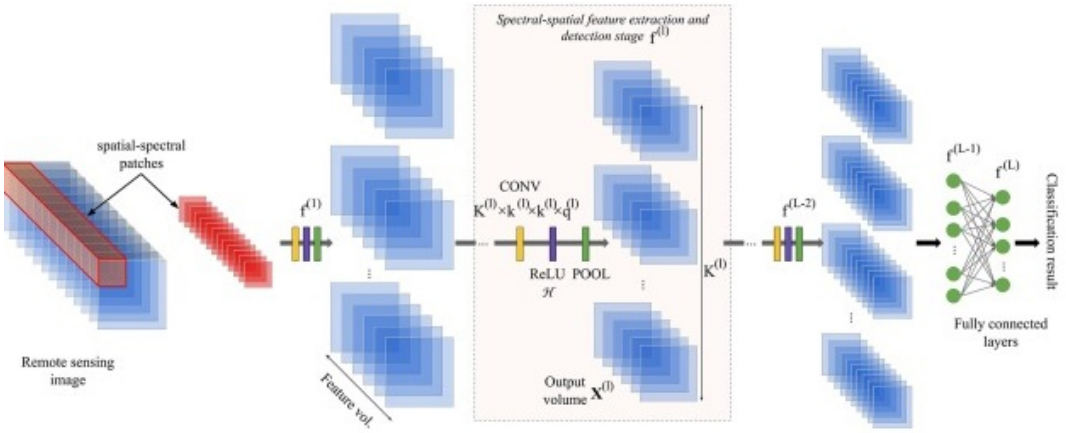

Figura 2.10: Arquitectura CNN3D Paoletti et al. (2019)

2.4 Pruning en redes neuronales

Los algoritmos de pruning "poda" en inglés eliminan las neuronas o los pesos innecesarios os menos relevantes para la red Blalock et al. (2020). De esta manera se busca obtener la red más sencilla posible sin afectar (o afectando lo mínimo posible) a la precisión de esta. La eliminación de pesos es más común ya que resulta más fácil de realizar sin afectar a los resultados de la red, pero la estructura resultante es más complicada de operar al no estar todos los pesos conectados a todas las neuronas. La poda de nodos enteros de la red, aunque es más complicada, permite una ejecución densa de la red lo cual es más fácilmente ejecutable a nivel hardware, la desventaja es que la eliminación de un nodo puede afectar en mayor medida a la precisión de la red.

Para seleccionar qué nodos o pesos podar existen varios métodos, el objetivo final es eliminar aquellos que sean menos importantes. En la figura 2.11 se muestra una visualización del resultado de aplicar un algoritmo de pruning a una red neuronal.

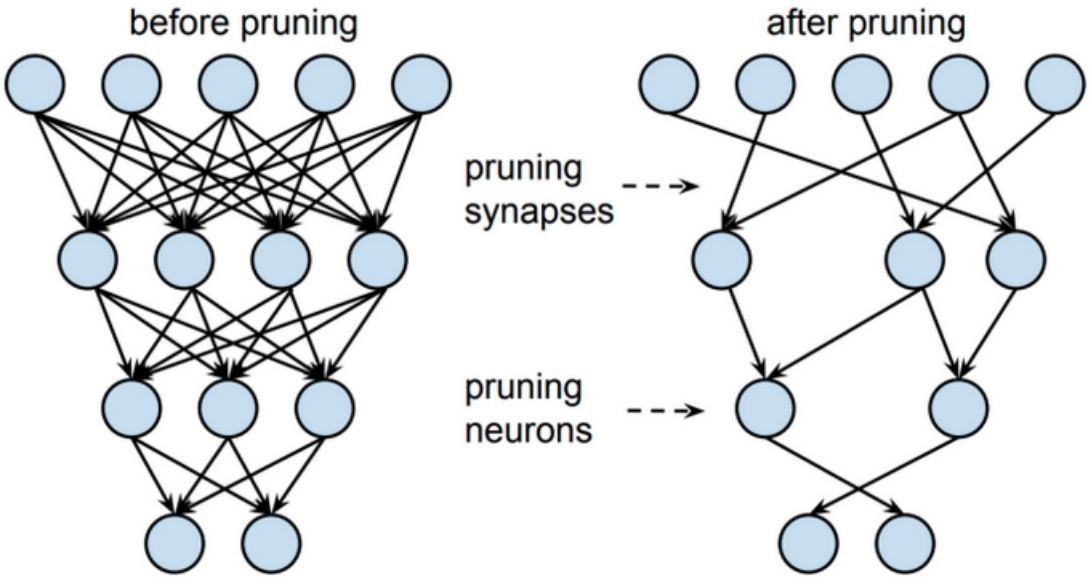

Figura 2.11: Visualización de pruning

El método más básico para realizar pruning a una red neuronal es el conocido como “oracle pruning”. Este método elimina las neuronas una por una y reentrena la red para analizar hasta que punto afecta dicha neurona a la precisión de la red. Es fácil de ver que es un método extremadamente costoso computacionalmente y no muy viable en casos reales en los que no se dispone de una capacidad de cómputo inmensa, por lo que se deben analizar otros métodos.Una de las maneras más simples de elegir que nodos o pesos podar se basa en los valores de los pesos, en los casos en los que los pesos sean muy próximos a cero se puede redondear este valor a cero, lo cual es equivalente a eliminar el peso sin temor a afectar de manera considerable a la red. De una forma similar se pueden analizar las neuronas estudiando sus activaciones al procesar los datos. Si estos valores son muy próximos a cero de manera consistente también se puede considerar que dicha neurona no es de especial relevancia para la red y se podrá eliminar con pocas repercusiones. Existen diversos métodos para realizar pruning a redes neuronales, varios se analizan en Molchanov et al. (2017) y Li et al. (2017) siendo aplicados a redes neuronales convolucionales.

Una de las principales desventajas de los algoritmos de pruning es la constante necesidad de reentrenar la red para evaluar la precisión perdida al aplicar la poda como se puede ver en la figura 2.12. Además su uso se vuelve más efectivo cuanto más profunda sea la red, por lo que no genera mucha mejora en redes menos profundas y más anchas, como es el caso de las redes aplicadas a la clasificación de imágenes hiperespectrales.

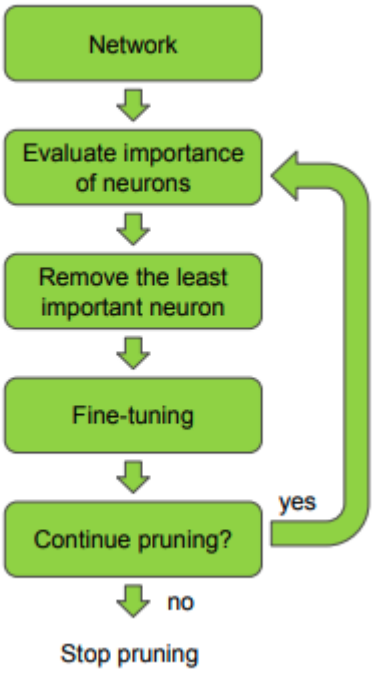

Figura 2.12: Flujo de pruning

2.5 Análisis de componentes principales

Principal component analysis es una técnica de reducción de dimensionalidad comúnmente aplicada a conjuntos grandes de datos que contienen un número elevado de variables, para reducir este número en la medida de lo posible afectando en la menor medida a la cantidad de información que proporcionan los datos. De esta forma se describe el conjunto de datos mediante nuevas variables no correlacionadas. Una cantidad de datos reducida reduce los costes computacionales y de entrenamiento al tener que estudiarse menos variables, y también facilita el análisis y la visualización de los datos. Fue inventado en 1901 por Karl Pearson Pearson (1901) como un análogo del teorema de ejes principales. Posteriormente fue desarrollado de manera independiente y nombrado por Harold Hotelling en 1933 Hotelling (1933).

PCA se utiliza en el análisis exploratorio de datos y en la elaboración de modelos predictivos. Se suele utilizar para la reducción de la dimensionalidad proyectando cada punto de datos solo en los primeros componentes principales para obtener datos de menor dimensión, preservando al mismo tiempo la mayor parte posible de la variación de los datos. El primer componente principal puede definirse de forma equivalente como una dirección que maximiza la varianza de los datos proyectados. El i-ésimo componente principal puede tomarse como una dirección ortogonal a los primeros i-1 componentes principales que maximiza la varianza de los datos proyectados.

PCA se define como una transformación lineal ortogonal que transforma los datos en un nuevo sistema de coordenadas de manera que la mayor varianza por alguna proyección escalar de los datos viene a situarse en la primera coordenada (denominada primer componente principal), la segunda mayor varianza en la segunda coordenada, y así sucesivamente Jolliffe (2002).

PCA puede ser empleado para reducir la cantidad de neuronas en una red neuronal y, en el caso de este trabajo, la cantidad de filtros convolucionales en una red neuronal convolucional. En el artículo Garg et al. (2019) se estudia la aplicación de PCA a redes neuronales convolucionales proporcionando dos principales ventajas:

1. Elimina la necesidad de reentrenar la red en cada iteración, requiriendo un único reentrenamiento al final del análisis.

2. Ofrece mejores resultados en redes menos profundas y más anchas a diferencia de los algoritmos de pruning.

Es por estas dos ventajas que se ha elegido este método para su estudio y desarrollo. En el capítulo 3 de este documento se analiza con más detalle el proceso de análisis y su aplicación a redes neuronales convolucionales.

Capítulo 3

Desarrollo

El desarrollo de este proyecto se basa principalmente en el artículo "A low effort approach to structured CNN design using PCA" Garg et al. (2019). En este artículo se propone la idea de emplear PCA para optimizar la arquitectura de una red ya entrenada, reduciendo el número de parámetros y el coste computacional y de memoria.

La principal ventaja de este método frente a otros es que únicamente requiere un reentrenamiento de la red al finalizar el análisis frente al reentrenamiento iterativo de los algoritmos de *pruning*, lo cual permite realizar la optimización de una manera mucho menos costosa. El objetivo final del método es conservar el mínimo número de filtros en la red que expliquen el 99,9% de la varianza en cada capa.

Este trabajo se realiza en combinación del artículo "Deep learning classifiers for hyperspectral imaging: A review" Paoletti et al. (2019). En este artículo se analizan las actuales técnicas para la clasificación de imágenes hiperespectrales, proporcionando una compilación de códigos y métodos disponibles en https//github.com/mhaut/hyperspectral_deeplearning_review que se emplearán para la evaluación y estudio de los resultados.

3.1 Proceso de análisis y optimización

En la figura 3.1 se muestra la diferencia de un método de optimización mediante pruning (arriba) y el proceso propuesto mediante el uso de PCA (abajo). Como se ve la principal ventaja consiste en la eliminación de los dos bucles, reduciéndose en gran medida los costes computacionales de la optimización. Adicionalmente se puede reducir el porcentaje de varianza deseado (99,9% por defecto) para satisfacer requisitos más estrictos de coste computacional o eléctrico en el sistema en que se pretende ejecutar el modelo.

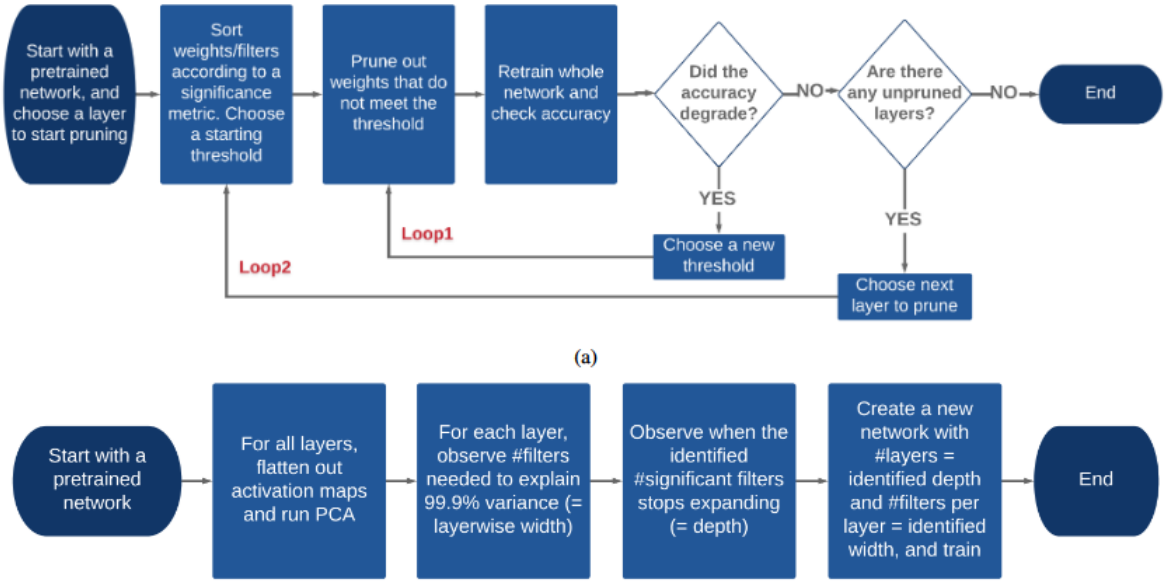

Figura 3.1: Proceso de optimización mediante pruning (arriba) y proceso de optimización mediante PCA (abajo) Garg et al. (2019)

En la figura 3.2 se muestra la visualización de los filtros convolucionales de una capa de la red preentrenada AlexNet. En esta imagen se puede apreciar la redundancia existente entre los filtros lo cual abre la posibilidad de eliminar parte de ellos sin afectar de manera significativa a la precisión de la red. El objetivo principal consiste en detectar que filtros están altamente correlacionados y que potencialmente detectan las mismas propiedades en la imagen para poder eliminarlos. De esta forma la optimización de la red se hace analizando el conjunto de los filtros de una capa, a diferencia de los algoritmos de pruning que realizan un análisis filtro a filtro.

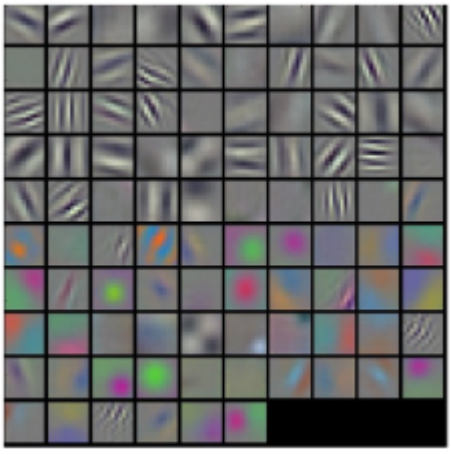

Figura 3.2: Visualización de filtros convolucionales de AlexNet Garg et al. (2019)

Para detectar la redundancia entre los filtros se emplean las activaciones que generan dichos filtros al procesar las imágenes. La entrada estándar para el algoritmo de PCA es una matriz de

dos dimensiones en la cual cada fila representa una nueva muestra y cada columna corresponde a una propiedad del filtro para cada una de las muestras. El valor de las propiedades de un filtro es la salida de la convolución de dicho filtro aplicado a una entrada como se muestra en la figura 3.3. Por lo tanto un valor en la matriz de PCA en la posición [1i,j] es el resultado de la activación de la entrada i por el filtro j. La misma entrada es procesada por todos los filtros creando una fila entera de propiedades en la matriz de PCA para dicha entrada. Hay tantas de estas entradas como pixeles hay en en el mapa de activación de la capa correspondiente. Este aplanamiento del mapa de activación tras la convolución da muchas muestras para los filtros pertenecientes a la capa. Si hay activaciones correlacionadas en la matriz generada con las activaciones aplanadas esto implica que hay hay filtros que generan resultados redundantes.

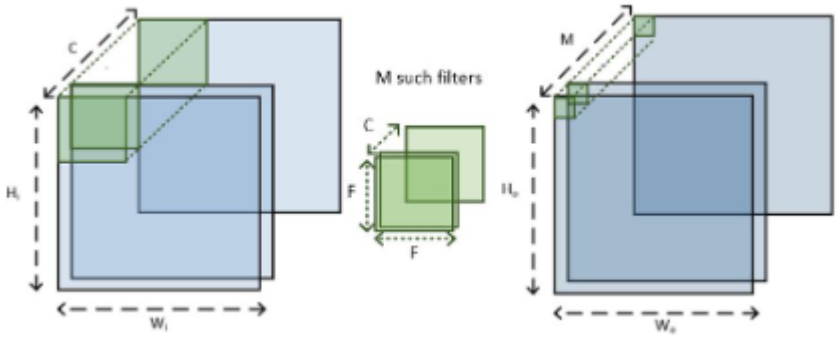

Figura 3.3: La salida de la convolución de un filtro se puede interpretar como el valor de las propiedades de dicho filtro. Los valores en verde generan una muestra para PCA Garg et al. (2019)

A continuación se muestra el pseudocódigo proporcionado en el artículo Garg et al. (2019)

Listado de código 3.1: Pseudocódigo

```
1  function FLATTEN(num_batches, layer)
2     for batch = 1 to num_batches do
3         Perform a forward pass
4         act_layer <- activations[layer]. size: N*H*W*C
5         reshape act_layer into [N*H*W,C]
6         for sample in act_layer do
7             act_flatten.append(sample)
8         end for
9     end for
10    return act_flatten
11 end function
12
13 function RUN_PCA(threshold, layer)
```

```
14      num_batches <- d(100 * C/(H * W * N ))e
15      act_flatten <- FLATTEN(num_batches, layer)
16      perform PCA on act_flatten, C num_components
17      var_cumul <- cumulative sum of explained_var_ratio
18      pca_graph <- plot var_cumul against #filters
19      SL <- #components with var_cumul<threshold
20      return SL
21  end function
22
23  function MAIN(threshold)
24      for all layer in layers do
25          SL <- RUN_PCA(threshold, layer)
26          S.append(SL)
27      end for
28      new_net <- [S[0]]
29      for i <- 1 to num_layers do
30          if S[i]>S[i - 1] then
31              new_net.append(S[i])
32          else
33              break
34          end if
35      end for
36      new config: # layers <- len(new_net)
37      each 'layers # filters <- SL
38      randomly initialize a new network with new config
39      train new network . Only training iteration
40  end function
```

En la figura 3.4 se muestra una visualización del algoritmo especificado en el pseudocódigo anterior junto a un ejemplo de la curva de PCA de resultados, la cual indica el punto en el que se alcanza el 99,9% de varianza entre los filtros de una capa.

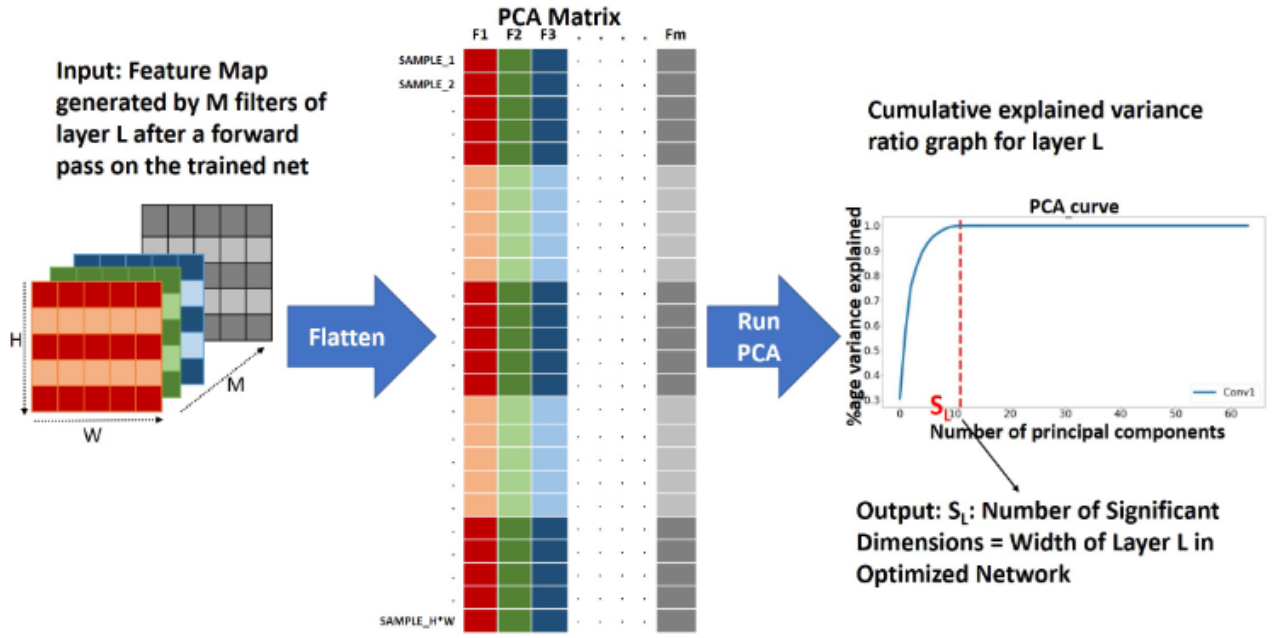

Figura 3.4: Visualización del algoritmo Garg et al. (2019)

3.2 Implementación del método

La implementación del método se ha realizado en el lenguaje de programación Python por la gran cantidad de librerías y frameworks disponibles para el desarrollo de técnicas de aprendizaje automático. Particularmente se ha empleado Keras y Tensorflow por ser las mas empleadas a lo largo de la formación en el Master de Ingeniería y Ciencia de Datos de la UNED.

El método se ha implementado en una clase de python dividida en dos métodos principales y una función principal.

3.2.1 Definición de la clase

A continuación se muestra la definición inicial de la clase "optimize" y su función ___init___ que inicializa sus parámetros.

Listado de código 3.2: Definición de la clase y función ___init___

```
1 class optimize:
2     def __init__(self):
3         self.bsize = 100
4         self.model = tf.keras.models.load_model("nombre_modelo.h5")
5         #alternativamente si es un modelo almacenado localmente
6         #self.model = load_model("nombre_modelo.h5")
7         self.dataset = read_data()
```

```
8
9           predicted_x = self.model.predict(self.dataset['x_test'])
10          residuals   = np.argmax(predicted_x,1)==np.argmax(self.
               dataset['y_test'],1)
11          print("Accuracy inicial: ", sum(residuals)/len(residuals))
```

3.2.2 Función flatten

A continuación se muestra el código de la función "flatten" que realiza el aplanamiento del mapa de activaciones generado por los filtros convolucionales para pasarlo como input al PCA.

Recibe como entrada el número de *batches* de datos sobre los que se va a operar y la capa de la red.

Listado de código 3.3: Definición de la función flatten

```
1  def flatten(self, num_batches, layer):
2      maxbatches = self.dataset['x_train'].shape[0]//self.bsize-1
3      random_positions = range(maxbatches)[:num_batches]
4      feat_ext = tf.keras.Model(self.model.input, self.model.layers[
          layer].output)
5      for idbatch in random_positions:
6          images = self.dataset['x_train'][self.bsize*idbatch:self.
              bsize*idbatch+self.bsize]
7          act_layer = feat_ext.predict(images)
8          act_layer = act_layer.reshape(-1, act_layer.shape[-1])
9          try:
10             act_flatten = np.vstack((act_flatten, act_layer))
11         except:
12             act_flatten = act_layer
13     return act_flatten
```

3.2.3 Función run_PCA

A continuación se muestra el código de la función "run_PCA", la cual realiza el análisis de PCA y el cálculo de la varianza entre los filtros para conservar únicamente el minimo necesario que explique la varianza determinada por el parámetroe de entrada "threshold".

Recibe como entrada la capa de la red y el threshold que determinará la varianza mínima a conservar entre los filtros.

Listado de código 3.4: Definición de la función run_PCA

```
def run_PCA(self, layer, threshold=0.999):
    feat_ext = tf.keras.Model(self.model.input, self.model.layers[
       layer].output)

    images = self.dataset['x_train'][:self.bsize]
    feat = feat_ext.predict(images)
    N, H, W, C = feat.shape
    num_batches = math.ceil(100*C/(H*W*N))
    act_flatten = self.flatten(num_batches, layer)

    pca = PCA(n_components=C).fit(act_flatten)
    a_trans = pca.transform(act_flatten)

    print('Explained variance ratio:',pca.explained_variance_ratio_
       )
    plt.plot(np.cumsum(pca.explained_variance_ratio_))
    optimal_num_filters=np.sum(np.cumsum(pca.
       explained_variance_ratio_)<threshold)
    plt.axvline(x=optimal_num_filters)
    plt.savefig('path_to_directory'+ str(layer) +'.jpeg')  #saves
       the PCA figure.
    print('Porcentaje de varianza que se quiere retener',threshold)
    print('El numero de filtros requerido para explicar dicha
       varianza es',optimal_num_filters)
    print('El porcentaje de filtros que explican la varianza es', (
       optimal_num_filters/C)*100,"%")
    print('TERMINA UNA ITERACION')
    print('-------------------------------------------------')
    return optimal_num_filters
```

3.2.4 Función main

A continuación se muestra el código de la función principal “main” que llama a las funciones previamente. Esta función distingue entre las capas convolucionales de la red de manera que únicamente estas sean pasadas a las funciones para su análisis. Itera sobre una lista de varianzas para los distintos experimentos, pero en un caso de uso típico únicamente se proporcionaría un único valor de varianza dentro de la lista, realizando así un único análisis conservando la varianza

deseada.

Listado de código 3.5: Definición de la función main

```
#precisiones = [0.999, 0.998, 0.997, 0.996, 0.995, 0.99, 0.95]
precisiones = [0.999]
resultados = []
for precision in precisiones:
    print("NUEVO ANALISIS")
    print("--------------------------")
    print("Precision empleada :", precision)
    opt_net = optimize()     S = []
    for i in range(len(opt_net.model.layers)):
        if 'conv' in opt_net.model.layers[i].name:
            S_L = opt_net.run_PCA(i, threshold=precision)
            S.append(S_L)
    print("modelo filtrado pesos", S)
    S_final = [S[i] for i in range(1, len(S)) if S[i] > S[i-1]]
    print("modelo filtrado capas", S_final)
    resultados.append(precision)
    resultados.append(S)
    print(resultados)
```

3.2.5 Función read_data

Para la carga y preparación de los datos de imágenes hiperespectrales se define una función que llama al método “load_data()” disponible en [referencia código] el cual prepara los datos para ser introducidos a los distintos métodos. Esta función devuelve un diccionario indicando los conjuntos de entrenamiento y validación así como el número de clases que contienen los datos. No recibe parámetros de entrada pero dentro de la función se deben modificar los valores para indicar a la función “load_data()” que parámetros emplear. Adicionalmente se ha modificado la precisión de los datos a float32 y así poder realizar el entrenamiento con el conjunto completo de datos ya que previamente podían producirse errores de reserva de memoria por el gran tamaño de los datos.

Listado de código 3.6: Definición de la función read_data

```
def read_data():
    #pixels, labels, num_class = loadData("UP", num_components=40,
       preprocessing="standard")
```

```
3     pixels, labels, num_class = loadData("IP", preprocessing="
         standard")
4     pixels, labels = createImageCubes(pixels, labels, windowSize=9,
          removeZeroLabels = False)
5     pixels = pixels[labels!=0]
6     labels = labels[labels!=0] - 1
7     x_train, x_test, y_train, y_test = split_data(pixels, labels,
         0.15, rand_state=None)
8     x_test  = x_test[..., np.newaxis]
9     x_train = x_train[..., np.newaxis]
10    inputhsape = x_train.shape[1:]
11    num_classes=16
12    x_train = x_train.astype('float32')
13    x_test  = x_test.astype('float32')
14    dataset = {'x_train':x_train, 'x_test':x_test, 'y_train':
         y_train, 'y_test':y_test, 'num_classes':num_classes}
15    return dataset
```

3.3 Modelos y redes empleadas

Como se ha mencionado previamente, los modelos empleados para las pruebas del método se han obtenido principalmente Paoletti et al. (2019) disponibles en https//github.com/mhaut/hyperspectral_
De los modelos disponibles se han empleado principalmente la redes neuronales convolucionales bidimensionales (cnn2d). Estas redes están definidas en https//github.com/mhaut/hyperspectral_deeplearı
y se componen de dos capas convolucionales. Adicionalmente se han empleado redes con 3 capas convolucionales para evaluar los resultados.

3.3.1 CNN2D con dos capas

Esta red contiene dos capas convolucionales con 50 filtros en la primera capa y 100 en la segunda seguido por una capa densa de 100 neuronas. El tamaño de los filtros convolucionales es de 5x5. Después de cada una de las capas convolucionales existe una capa de activación cuya función de activación es la función relu.

Tras la segunda capa de activación se define una capa de agrupación de tamaño 2x2. Posteriormente se define una capa densa con 100 neuronas y una capa de activación con función de activación *relu*. Por último se define una última capa densa cuyo tamaño será el número de de clases a clasificar en los datos y una función de activación *softmax*.

En el cuadro 3.1 se muestra la recopilación de esta arquitectura, habiéndola extraído de una red entrenada con un conjunto de datos con 16 clases, por lo que la última capa densa cuenta con 16 neuronas.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 15, 15, 50)	50050
activation (Activation)	(None, 15, 15, 50)	0
conv2d_1 (Conv2D)	(None, 11, 11, 100)	125100
activation_1 (Activation)	(None, 11, 11, 100)	0
max_pooling2d (MaxPooling2D)	(None, 5, 5, 100)	0
flatten (Flatten)	(None, 2500)	0
dense (Dense)	(None, 100)	250100
activation_2 (Activation)	(None, 100)	0
dense_1 (Dense)	(None, 16)	1616

Cuadro 3.1: Summary de la red cnn2d de 2 capas

La figura 3.5 muestra una representación visual aproximada de la red neuronal convolucional definida.

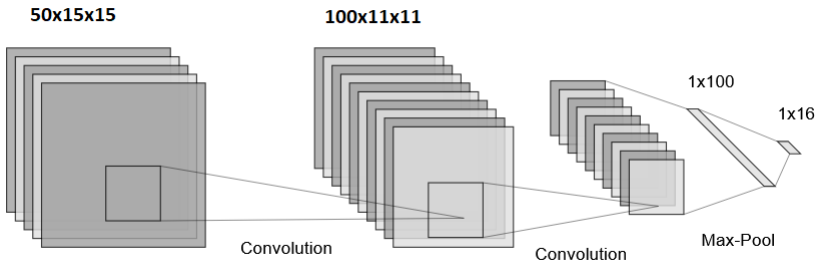

Figura 3.5:

3.3.2 CNN2D con tres capas

La siguiente red definida contiene tres capas convolucionales con 32 filtros en la primera capa, 64 en la segunda y 128 en la tercera, seguido de una capa densa de 128 neuronas. El tamaño de los filtros convolucionales es 3x3. Esta red es inicialmente más estrecha que la anterior pero en la última capa acaba contiene más filtros convolucionales que al estar en una capa más profunda extraen cualidades más específicas de los datos.

Al igual que en la red del apartado 3.3.1, tras cada capa convolucional hay una capa de activación cuya función de activación es *relu*. Esta red no cuenta con capas de agrupación. Tras las capas convolucionales se define una capa densa con 128 neuronas y función de activación *relu*. Por último se define una capa densa al igual que en la red anterior cuya cantidad de neuronas será igual al número de clases que se quieran clasificar en los datos.

En el cuadro 3.2 se muestra la recopilación de esta arquitectura, habiéndola extraído de una red entrenada con un conjunto de datos con 16 clases, por lo que la última capa densa cuenta con 16 neuronas.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 7, 7, 32)	11552
activation (Activation)	(None, 7, 7, 32)	0
conv2d_1 (Conv2D)	(None, 5, 5, 64)	18496
activation_1 (Activation)	(None, 5, 5, 64)	0
conv2d_2 (Conv2D)	(None, 3, 3, 128)	73856
flatten (Flatten)	(None, 1152)	0
dense (Dense)	(None, 128)	147584
activation_2 (Activation)	(None, 128)	0
dense_1 (Dense)	(None, 9)	1161

Cuadro 3.2: Summary de la red cnn2d de 3 capas

La figura 3.6 muestra una representación visual aproximada de la red neuronal convolucional definida.

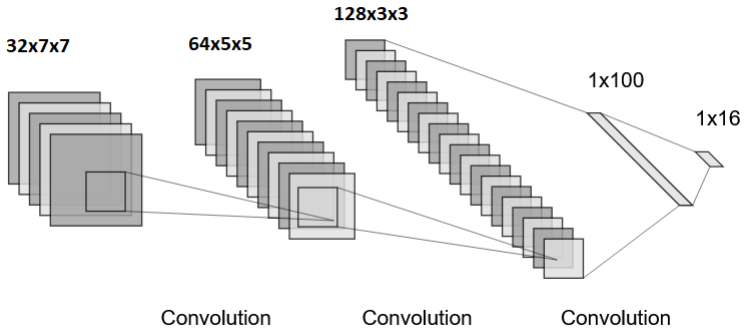

Figura 3.6:

Capítulo 4

Experimentos y resultados

En este capitulo se explican los experimentos realizados para la evaluación de la viabilidad del método y el análisis de los resultados arrojados por dichos experimentos. Previamente se ha verificado la correcta implementación del método realizando una comparativa de resultados con los mostrados en Paoletti et al. (2019) aplicándolo a las redes seleccionadas en el artículo. Los resultados obtenidos son muy similares (no idénticos dada la variabilidad inherente a las reden neuronales) y se confirma el funcionamiento para poder aplicarlo a las redes neuronales convolucionales disponibles en https//github.com/mhaut/hyperspectral_deeplearning_review.

4.1 Experimentos

Para la evaluación de la viabilidad del método se han definido el siguiente experimento,

- Aplicación del método a arquitectura cnn de dos capas (3.3.1) y tres capas (3.3.2) conservando la misma semilla de datos y reduciendo el porcentaje de varianza a conservar en cada iteración [0.999, 0.998, 0.997, 0.996 0.995, 0.99, 0.95]
- Reentrenamiento de las redes con las nuevas arquitecturas y análisis de las precisiones obtenidas.
- Comparativa entre descenso de precisión y descenso de coste computacional.
- Replicación del experimento con distintos conjuntos de datos para evaluar la consistencia (falta de capacidad de computo).

La ejecución de los experimentos se ha realizado de manera conjunta entre un entorno de ejecución remota de Google Colab para el entrenamiento de las redes (debido al alto coste computacional) y un ordenador personal para la aplicación del método de optimización y el análisis de resultados. El método de optimización desarrollado requiere una capacidad de cómputo relativamente baja comparada con el entrenamiento de las redes neuronales convolucionales y no

requiere de GPU para acelerar el entrenamiento, sin embargo se necesita una cantidad elevada de memoria RAM para el manejo del conjunto de datos.

Todos los experimentos se realizan empleando los parámetros por defecto que vienen definidos en los métodos analizados a excepción de los siguientes:

- Dataset: Se debe seleccionar uno de los datasets disponibles para el entrenamiento y evaluación.
- Verbose_train: Amplía los logs que se muestran por pantalla al realizar el entrenamiento de las redes.
- Spatialsize: Indica el tamaño de ventana que se empleará alrededor del píxel que se va a clasificar en la imagen. Por defecto tiene un valor de 19 pero se ha demostrado que este es un valor muy elevado y actualmente se emplean valores menores para la clasificación de la imagen. El valor elegido para el análisis es 11 para la red cnn2d de dos capas y 9 para la red cnn2d de tres capas.
- Random_state: Inicializa la semilla aleatoria para dividir el conjunto de datos. Por defecto no tiene valor. El valor elegido es 42. Seleccionando un valor fijo para la semilla podemos asegurar que el entrenamiento y el análisis de todas las redes se realizan con los mismos conjuntos de datos.

A continuación se muestra un ejemplo del comando necesario para ejecutar el entrenamiento de una CNN2D con los valores definidos previamente y con el dataset Indian Pines (IP):

- python3 cnn2d_ampliada.py --dataset IP --verbosetrain --spatialsize 9 --random_state 42

4.2 Resultados

4.2.1 CNN2D dos capas

A continuación se muestran los resultados del experimento sobre una red neuronal convolucional bidimensional con la arquitectura inicial definida en el apartado 3.3.1 de este documento.

Inicialmente la red tiene los siguientes valores:

- Precisión: 95,13%
- Parámetros: 386866
- Tamaño de ventana: 11
- Porcentaje de entrenamiento: 15%

% Varianza	Capas convolucionales	% Precisión	Nº parámetros	% Parámetros
100	[50, 100]	95,12	386866	100
99,9	[26, 41]	93,37	162533	42,01
99,8	[21, 31]	93,68	126143	32,61
99,7	[18, 26]	94,16	106060	27,42
99,6	[16, 23]	92,13	93255	24,11
99,5	[15, 21]	93,29	86727	22,42
99	[12, 15]	91,63	67743	17,51
95	[7, 5]	83,30	38103	9,85

Cuadro 4.1: Resultados de la aplicación de PCA sobre una cnn2d de dos capas

En la tabla 4.1 se muestran los resultados de la aplicación del algoritmo de optimización mediante PCA a la red neuronal convolucional definida en el apartado 3.3.1 de este documento. El experimento realizado conservando el 99,9% de varianza en los filtros muestra un descenso del 57.99% en el número de parámetros entrenables de la red perdiendo a penas un 1.75% de precisión en la red. Este experimento se muestra en la figura 4.1. Ambas capas de la red muestran una trayectoria muy vertical seguida de una curva muy cerrada, esto indica que son propensas a poder perder filtros convolucionales sin afectar en gran medida a la precisión de la red, como se demuestra por los resultados obtenidos.

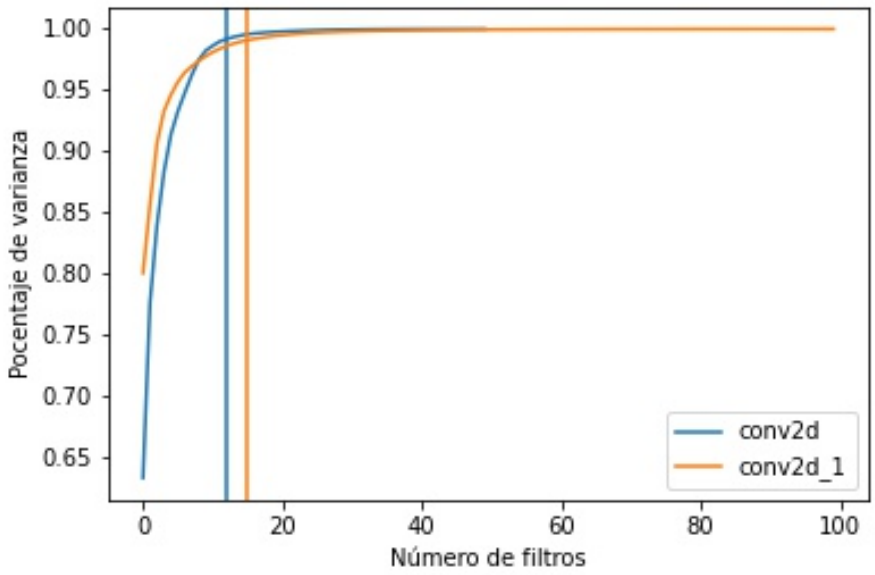

Figura 4.1: Curva PCA obtenida de la red 3.3.1 conservando una varianza del 99,9% en los filtros

En las figuras 4.2 y 4.3 se muestran los resultados de la aplicación a cada capa (primera y segunda respectivamente) del método a la red neuronal con un descenso progresivo del porcentaje de varianza a conservar entre los filtros convolucionales. Entre el 99,9% de varianza y el 99,5% se

aprecia un estrechamiento entre los huecos que separan cada línea a pesar de tener un descenso de la varianza equitativo. Esto es debido a la aproximación al comienzo de la curva donde el descenso empieza a ser mas pronunciado, indicando que cada filtro que se elimine producirá un mayor descenso en el porcentaje de varianza que conservarán entre ellos.

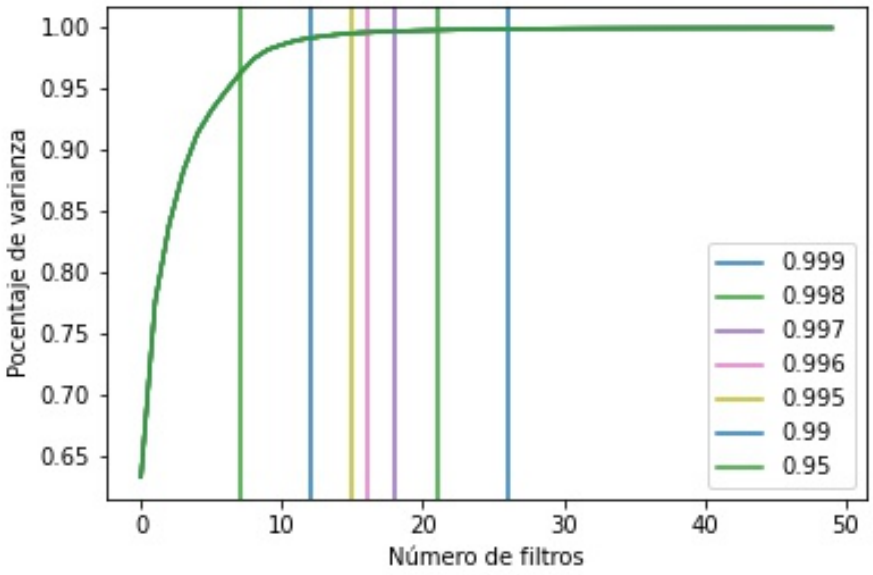

Figura 4.2: Resultados de optimización de la primera capa convolucional para la red definida en el apartado 3.3.1 con los resultados de la tabla 4.1

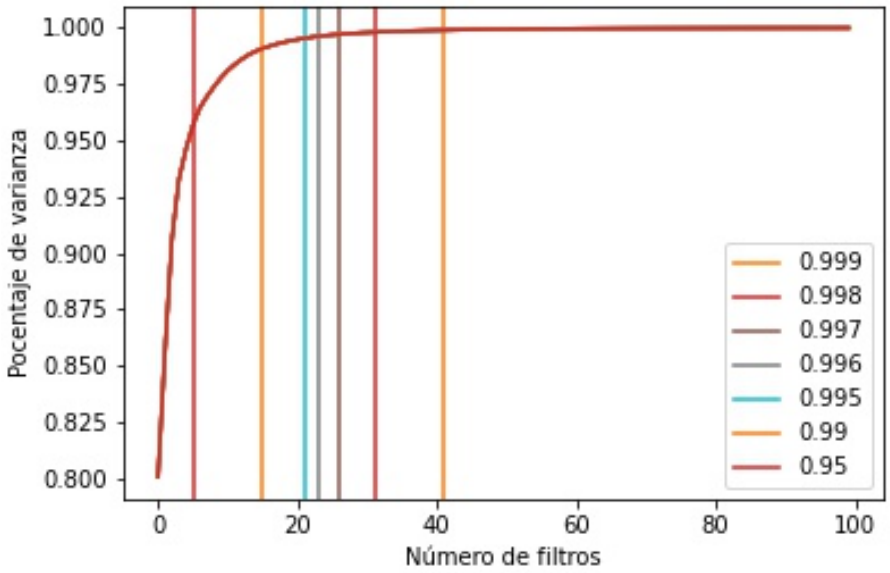

Figura 4.3: Resultados de optimización de la segunda capa convolucional para la red definida en el apartado 3.3.1 con los resultados de la tabla 4.1

En la figura 4.4 se muestra el porcentaje de parámetros entrenables frente al porcentaje de varianza conservada entre los filtros convolucionales. Se aprecia un enorme descenso entre el 100% de varianza (100% de los parámetros) y el 99.9% (42.01% de los parámetros) lo cual significa que es un ahorro enorme de coste computacional comparado con el sacrificio de precisión del 1,75%. Esto junto a los análisis anteriores indican que la red inicialmente no tenía una arquitectura muy eficiente y es fácilmente mejorable en su rendimiento sin hacer un gran sacrificio de precisión.

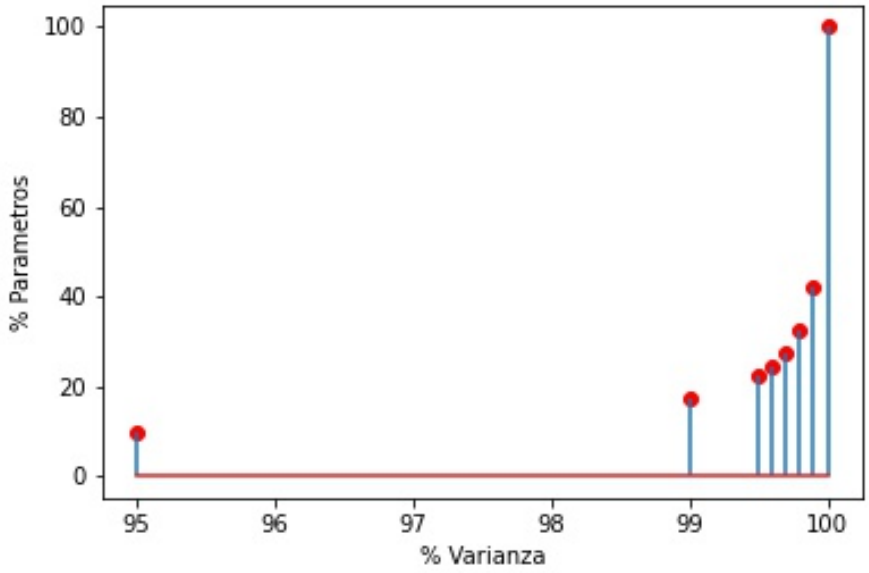

Figura 4.4: Porcentaje de parámetros frente a porcentaje de varianza conservado para la red definida en el apartado 3.3.1 con los resultados de la tabla 4.1

4.2.2 CNN2D tres capas

A continuación se muestran los resultados del experimento sobre una red neuronal convolucional bidimensional con la arquitectura inicial definida en el apartado 3.3.2 de este documento.

Inicialmente la red tiene los siguientes valores:

- Precisión: 92,65%
- Parámetros: 299632
- Tamaño de ventana: 9
- Porcentaje de entrenamiento: 15%

% Varianza	Capas convolucionales	% Precisión	Nº parámetros	% Parámetros
100	[32, 64, 128]	92,65	299632	100
99,9	[25, 52, 113]	91,38	242142	80,81
99,8	[21, 46, 103]	91,68	210154	70,14
99,7	[18, 41, 96]	90,21	187405	62,55
99,6	[16, 38, 90]	90,06	171068	57,10
99,5	[15, 35, 85]	90,77	158747	52,98
99	[11, 27, 68]	87,59	119631	39,93
95	[6, 11, 27]	69,70	47407	15,70

Cuadro 4.2: Resultados de la aplicación de PCA sobre una cnn2d de tres capas

En la tabla 4.2 se muestran los resultados de la aplicación del algoritmo de optimización mediante PCA a la red neuronal convolucional definida en el apartado 3.3.2 de este documento. Para el experimento en el que se conserva un 99,9% de la varianza entre los filtros se aprecia un descenso en la precisión del 1,27% para un descenso del 19,19% en el número de parámetros entrenables de la red. Esto significa que los filtros convolucionales en cada capa son mucho más relevantes para mantener un porcentaje alto de varianza entre ellos. Se puede visualizar en la figura 4.5 para la capa convolucional "conv2d_2" en la que se aprecia una curva mucho más abierta con un ascenso menos vertical comparado con las otras dos capas.

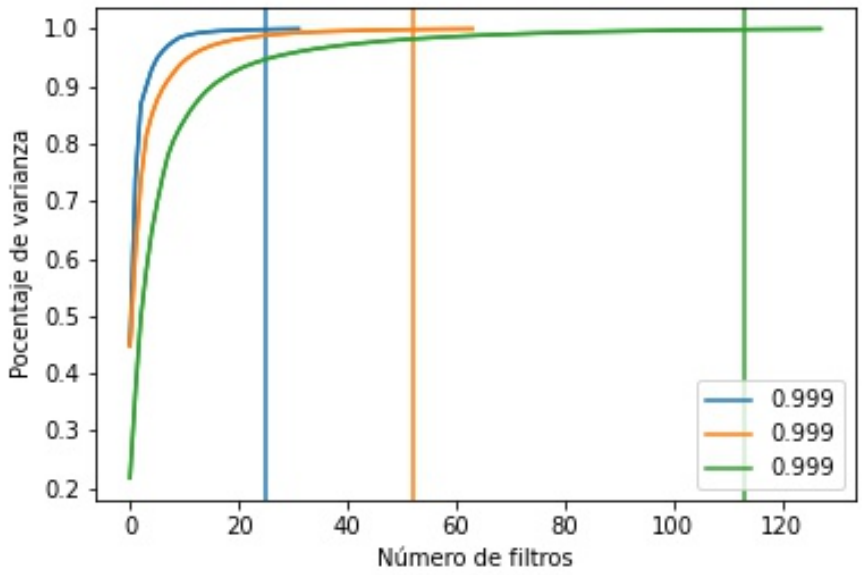

Figura 4.5: Curva PCA obtenida de la red 3.3.2 conservando una varianza del 99,9% en los filtros

En las figuras 4.6, 4.7 y 4.8 se muestran las curvas de PCA individuales para cada capa. Como se ha mencionado previamente la capa "conv2d_2" tiene la curva más abierta y es la comenzará antes a producir grandes descensos en el porcentaje de varianza de los filtros de la

capa si se eliminan suficientes filtros. Por el contrario la capa “conv2d” tiene una curva mucho más vertical con un giro muy pronunciado lo cual la hace menos susceptible de de generar grandes descensos en la varianza de los filtros de su capa. Esto se debe a que la capa “conv2d” es la primera capa de la red y sus filtros obtienen propiedades de la imagen más generalizadas y fácilmente adquiribles por las de otros filtros. Por el contrario, la capa “conv2d_2” es la última capa de la red y obtiene las propiedades de menor nivel, extrayendo cualidades más precisas. Esto significa que la eliminación de un filtro en la última capa afecta más a la red ya que el resto de los filtros tienen mayor dificultad en extraer las cualidades que extraía el filtro eliminado.

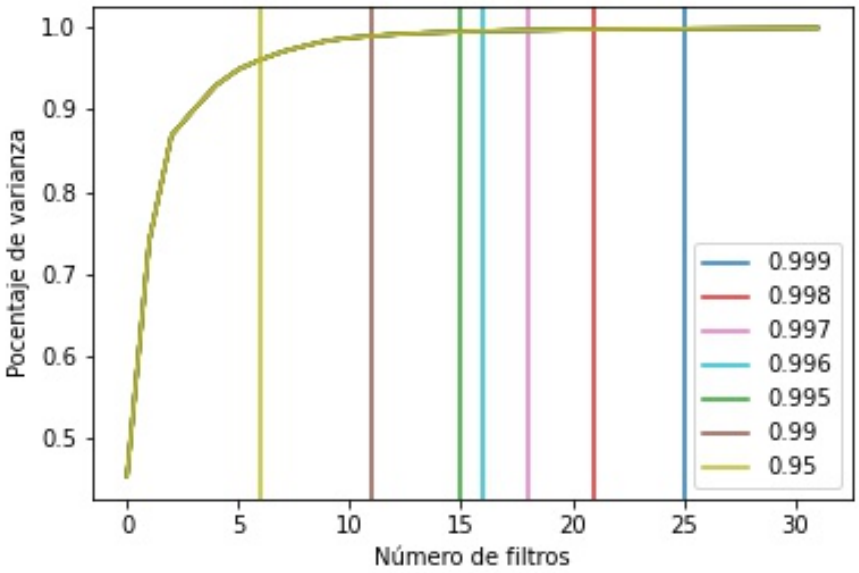

Figura 4.6: Resultados de optimización de la primera capa convolucional para la red definida en el apartado 3.3.2 con los resultados de la tabla 4.2

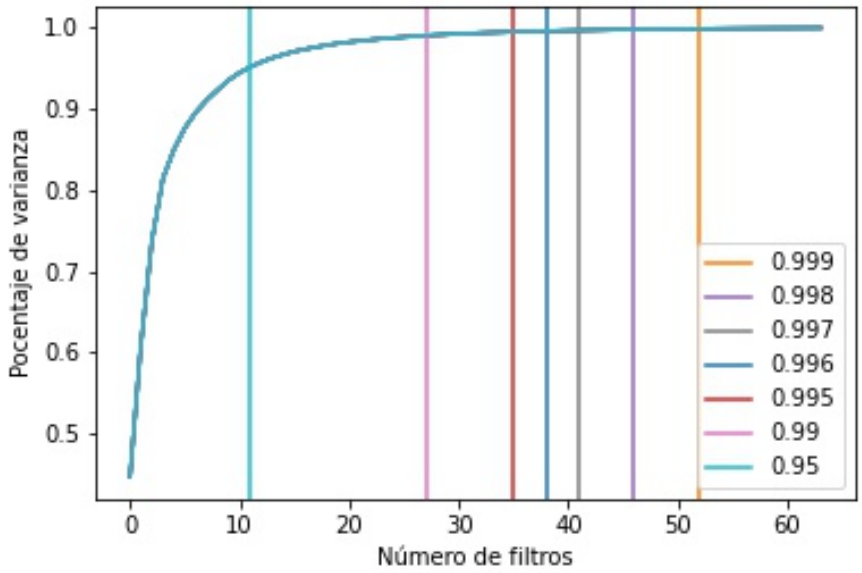

Figura 4.7: Resultados de optimización de la segunda capa convolucional para la red definida en el apartado 3.3.2 con los resultados de la tabla 4.2

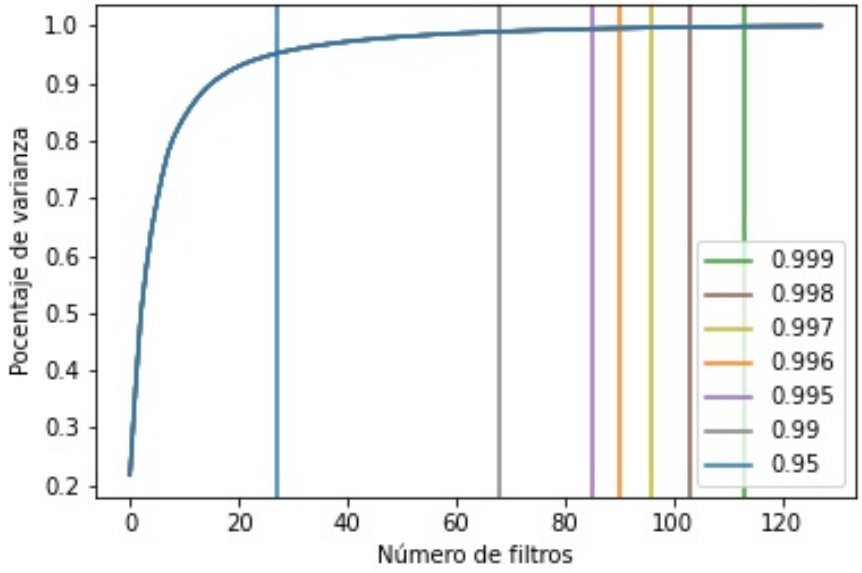

Figura 4.8: Resultados de optimización de la tercera capa convolucional para la red definida en el apartado 3.3.2 con los resultados de la tabla 4.2

En la figura 4.9 se muestra la curva de regresión cuadrática calculada con los valores de varianza frente a precisión de la tabla 4.2. En esta curva se muestra el descenso esperado de la precisión frente al porcentaje de varianza conservado entre los filtros de una capa. Debido a la concentración de la mayoría de los valores en puntos muy cercanos al 100% de varianza y

con una distancia muy pequeña entre los puntos no se puede esperar una curva fiable. Como experimento se ha aplicado el método de PCA a la red cnn2d de tres capas conservando un 70% de varianza entre los filtros, ya que según la curva este debería ser un valor muy próximo a 0 en la precisión de la red. Tras la aplicación del método a la red se obtiene que debe tener una composición de 1 filtro en la primera capa, dos filtros en la segunda y 6 filtros en la tercera ([1, 2, 6]. Aplicando esta arquitectura a la red y reentrenándola se obtiene una precisión de 33.91969290380917%.

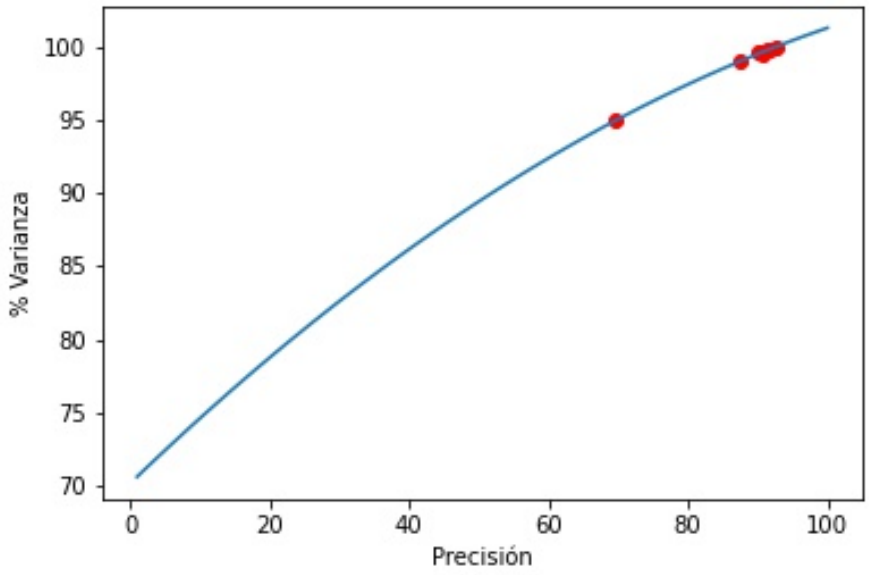

Figura 4.9: Visualización de la curva de regresión cuadrática para la red definida en el apartado 3.3.2 con los resultados de la tabla 4.2

Con el valor obtenido de precisión con el 70% de varianza en los filtros se recalcula la curva de regresión cuadrática que se muestra en la figura 4.10. De la misma forma que previamente se observaba una aproximación al 0% de precisión alrededor del 70% de varianza, ahora se observa cercano al 30% de varianza. Se aplica el método a la red conservando un 30% de varianza para comprobar la configuración devuelta. El resultado es 0 filtros en las dos primeras capas y un filtro en la tercera ([0, 0, 1]). No es necesario reentrenar la red para saber que va a obtener valores de precisión muy bajos al contar únicamente con una capa y un filtro convolucional. Esta curva muestra una aproximación más real a la progresión de la precisión de la red frente al descenso del porcentaje de varianza conservado entre los filtros.

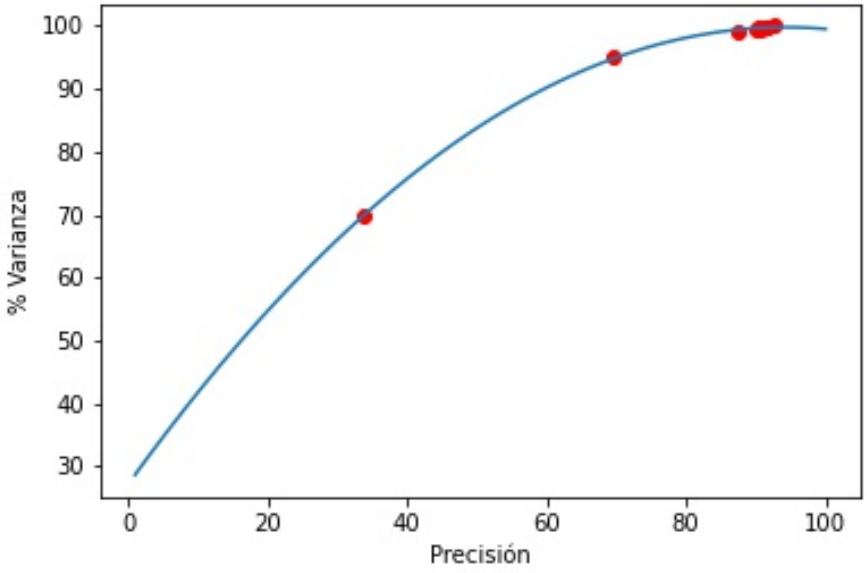

Figura 4.10: Visualización de la curva de regresión cuadrática para la red definida en el apartado 3.3.2 con los resultados de la tabla 4.2 ampliada con el cálculo del 70% de varianza en los filtros de la red

Capítulo 5

Conclusiones y trabajos futuros

5.1 Conclusiones

Las conclusiones del trabajo son muy positivas en el aspecto de la posibilidad de optimizar el rendimiento de redes neuronales convolucionales sin un coste elevado para la precisión y sin un gran esfuerzo para el cálculo de esta arquitectura optimizada a diferencia de los algoritmos de pruning con mayor coste computacional.

La aplicación del método y la extracción de las curvas de PCA facilitan mucho la identificación de redes poco optimizadas con un gran número de filtros poco relevantes. Tambien facilita visualizar cuales son las capas más propensas a aportar mayor precisión. Además de detectar de manera eficaz la redundancia entre los filtros de las primeras capas convolucionales, lo cual suele ser mas costoso para los algoritmos de pruning y en el caso de las redes para la clasificación de imágenes hiperespectrales la arquitectura suele ser bastante ancha pero poco profunda, por lo que su aplicación para estas redes es especialmente relevante.

Debido a la falta del hardware necesario no se han podido realizar más experimentos con diversos datasets o con más configuraciones de redes neuronales ya que requería demasiado tiempo el entrenamiento de tantas redes neuronales para su evaluación y los conjuntos de datos tienen un tamaño muy elevado.

5.2 Trabajos futuros

La principal ruta que se debe seguir para trabajos futuros es la adaptación del método para su uso en redes neuronales convolucionales con filtros cúbicos (cnn3d). El aplanamiento de estos filtros para formar la matriz que recibe el algoritmo de PCA como entrada es distinto, pero no se requieren mayores modificaciones en el método presentado. La aplicación del método a estas redes puede significar un ahorro mucho más grande en memoria y coste computacional ya que la eliminación de un filtro cúbico proporciona mucho mas ahorro que si se elimina un filtro

cuadrado.

Para optimizar más las redes se propone el uso de precisión mixta proporcionado por Tensorflow. Esta precisión mixta altera la precisión de los datos entre float16 y float32 para optimizar el consumo de memoria cuando no se requiera tanta precisión. No se ha podido realizar la experimentación con la precisión mixta ya que requiere de GPUs más modernas y potentes que las proporcionadas por Google Colab. Para su uso basta con aplicar las siguientes dos líneas de código al comienzo de la definición de la red neuronal:

- policy = mixed_precision.Policy('mixed_float16')
- mixed_precision.set_global_policy(policy)

Capítulo 6

Estimación y presupuestos

6.1 Estimación

A continuación se muestra la estimación inicial de las horas dedicadas al desarrollo del proyecto

- Estudio de documentación y del software disponible en la literatura de teledetección para comprender las particularidades de las imágenes teledetectadas: 20 horas
- Estudio de los frameworks de desarrollo de arquitecturas neuronales con el fin de elegir el más adecuado para la tarea a realizar: 20 horas
- Desarrollar códigos de procesamiento de imágenes teledetectadas en el framework seleccionado: 90 horas
- Análisis del estado del arte de algoritmos de pruning y su aplicación a imágenes remotas: 50 horas
- Proveer posibles mejoras en el procesamiento centradas en el consumo energético, redes multievaluadas: 40 horas
- Realización de la memoria: 80 horas

El consumo real de horas no se ha registrado detalladamente, pero se puede afirmar con seguridad que el estudio de la literatura tanto para imágenes teledetectadas como de los algoritmos de pruning aplicados a redes neuronales neuronales convolucionales ha sido muchísimo más elevado, del orden de 3 o 4 veces más de lo planificado. Especialmente la búsqueda de artículos que mostrasen potencial para su aplicación a las redes neuronales convolucionales que se pretendían reducir.

6.2 Presupuestos

El coste en horas ha sido elevado ya que el proyecto se ha prolongado durante medio año. El principal consumo de horas ha sido el estudio y revisión de la literatura que en las fases iniciales de desarrollo consumió mucho más de lo esperado.

No ha habido coste en el estudio de la literatura ya que todos los artículos consultados estaban disponibles de forma libre.

No ha habido coste en licencias software al emplear software libre en la totalidad del proyecto.

No ha habido coste en el consumo de hardware al haberse realizado los entrenamientos en el entorno de desarrollo gratuito de Google Colab. Aunque este entorno era gratuito ha demostrado ser insuficiente para la realización completa del proyecto. Las tarjetas gráficas empleadas no eran lo bastante modernas para poder emplear la precisión mixta proporcionada por Tensorflow como se menciona en el apartado de trabajos futuros de este documento. Adicionalmente no se disponía de suficiente memoria RAM para realizar los experimentos con otros datasets más pesados conservando todos los elementos de dichos datasets y así manteniendo la consistencia con los demás experimentos. Por último el entrenamiento de las redes con datasets más pesados era excesivamente lento y no se podía programar y dejar corriendo constantemente ya que Google Colab expulsa a los usuarios que realizan cálculos prolongados sin supervisión.

Aunque no ha habido costes adicionales en el desarrollo del proyecto se requiere de sistemas hardware potentes y con plena disposición para poder realizar el trabajo completo.

Bibliografía

Blalock, D., Ortiz, J. J. G., Frankle, J., and Guttag, J. (2020). What is the state of neural network pruning?

Chilton, A. (2020). The working principle and key applications of infrared sensors.

Ciresan, D., Meier, U., Masci, J., Gambardella, L. M., and Schmidhuber, J. (2011). Flexible, high performance convolutional neural networks for image classification.

ESA (2019). Data reduction and compression session.

Fukushima, K. (1980). Neocognitron: A self-organizing neural network modelfor a mechanism of pattern recognitionunaffected by shift in position.

Garg, I., Panda, P., and Roy, K. (2019). A low effort approach to structured cnn design using pca.

Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components.

IBM (2020). Convolutional neural networks. *IBM Cloud Education*.

Jolliffe, I. T. (2002). Principal component analysis.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1988). Gradient-based learning applied to document recognition.

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Gra, H. P. (2017). Pruning filters for efficient convnets.

Metaespectral (2021). Metaspectral - multispectral and hyperspectral data management.

Molchanov, P., Tyree, S., Karras, T., Aila, T., and Kautz, J. (2017). Pruning convolutional neural networksfor resource efficient inference.

NASA (2000). Earth observing 1 (eo-1).

Paoletti, M., Haut, J. M., Plaza, J., and Plaza, A. (2019). Deep learning classifiers for hyperspectral imaging: A review. *ISPRS Journal of Photogrammetry and Remote Sensing*.

Pearson, K. (1901). On lines and planes of closest fit to systems of points in space.

Ramsundar, B. and Zadeh, R. B. (2018). *TensorFlow for Deep Learning*. O'Reilly Media, Inc.

Xu, R., Wang, X., Chen, K., Zhou, B., and Loy, C. C. (2020). Positional encoding as spatial inductive bias in gans.

Yamaguchi, Kouichi; Sakamoto, K. A. T. F. Y. (1990). A neural network for speaker-independent isolated word recognition.

Yingge, H., Ali, I., and Lee, K.-Y. (2020). Deep neural networks on chip - a survey.