E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

Capas basadas en operadores OWA para Redes Neuronales Convolucionales

Máster Universitario en
Ingeniería Informática

Trabajo Fin de Máster

Alumna: Iris Dominguez Catena

Director: Mikel Galar Idoate

Codirector: Daniel Paternain Dallo

Pamplona, 17 de Febrero de 2020

upna Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Abstract

In this work we explore a novel way for increasing the capacity of Convolutional Neural Networks. Particularily, we propose a new technique for the generation of additional information based on the output of a convolutional block of a Convolutional Neural Network. We derive this information from the application of OWA operators at the channel level, and use this information to augment the input of the next layers in the network. We perform experiments to validate this new technique, testing how different parameters, namely the insertion point in the network, the ammount of learned OWA operators, and the channel sorting metric, affect the accuracy results of the network.

Keywords— Deep Learning, Neural Networks, Convolutional Neural Networks, Image Classification, OWA operators

Resumen

En este trabajo exploramos una nueva forma de ampliar la capacidad de las Redes Neuronales Convolucionales. En concreto, planteamos una nueva técnica para generar información adicional a partir de la salida de un bloque convolucional de una Red Neuronal Convolucional, empleando para ello operadores OWA a nivel de canal, y usando esta nueva información para ampliar la entrada de las siguientes capas de la red. Realizamos diversas pruebas con esta nueva técnica, comprobando como afectan diferentes parámetros a los resultados, incluyendo el punto de inserción de la nueva información, la cantidad de operadores OWA aplicados, o el tipo de métrica empleada para ordenar los canales de información original.

Palabras clave— Deep Learning, Redes Neuronales, Redes Neuronales Convolucionales, clasificación de imágenes, operadores OWA

Índice general

Capítulo 1

Introducción

1.1 Machine Learning

La Inteligencia Artificial (o *Artificial Learning*) es el campo de la informática dedicado al desarrollo de sistemas semi-autónomos, donde una máquina (un ordenador) realiza tareas sin disponer de instrucciones explícitas. En general, son sistemas bioinspirados, basados en la idea de la realización de tareas tras un aprendizaje.

Hoy en día, disponemos de ordenadores cada vez más potentes, pero limitados muchas veces por nuestra capacidad para diseñar algoritmos que hagan uso de esos recursos para procesar información. En algunos casos, como en los problemas de clasificación de imágenes, resulta difícil o casi imposible definir manualmente las reglas que llevan a los seres humanos a tomar decisiones. De esta imposibilidad nace el concepto de Machine Learning, donde en lugar de enseñarle al ordenador a, por ejemplo, clasificar imágenes según unas normas, creamos estructuras mas genéricas capaces de aprender a realizar esas tareas imitando como lo hacen otros sistemas.

1.2 Inteligencia Computacional

Por otro lado, la Inteligencia Computacional es otra rama de la informática cercana a la Inteligencia Artificial [1], con mucho solapamiento entre ambas, pero centrada en la solución de problemas no resolubles algorítmicamente. Algunos temas concretos que se suelen estudiar dentro de la Inteligencia Computacional son las Redes Neuronales, los Algoritmos Evolutivos y la Lógica Difusa (*Fuzzy Logic*).

La Lógica Difusa [2] en particular, es un tipo de lógica polivalente que contrasta con la lógica clásica en recoger valores de verdad difusos, en el conjunto de los números reales, en vez de binarios. En general, este tipo de lógica busca servir de puente entre la computación clásica de los ordenadores y el pensamiento aproximado de las personas, permitiendo tomar decisiones aproximadas y gestionar la incertidumbre.

1.3 Redes neuronales / Deep Learning

Para este trabajo, de los diferentes sistemas de Machine Learning existentes, nos centraremos en las Redes Neuronales (RN), y en concreto en las Redes Neuronales Profundas (*Deep Learning*). En este tipo de sistemas, empleamos pequeñas unidades computacionales (neuronas) y construimos redes a partir de ellas.

Estas redes, pese a su sencillez teórica (al menos en las variantes más elementales), demuestran una capacidad de cálculo realmente importante. Ya desde las primeras propuestas [3] se observó la gran capacidad de generalización que tienen. En concreto, es conocido el Teorema de Aproximación Universal, demostrado en una de sus primeras versiones en 1989 [4], y que establece la capacidad de aproximar cualquier función continua sobre intervalos acotados del conjunto de números reales con una red neuronal con tan solo una capa oculta y un número finito de neuronas.

En la práctica, hoy en día se trabaja sobre todo con Redes Neuronales Profundas, esto es, redes neuronales con múltiples niveles de capas de neuronas y arquitecturas más complejas. En concreto, destacan las Redes Neuronales Convolucionales, capaces de trabajar sobre imágenes, uno de los problemas más abordados por la Inteligencia Artificial actualmente.

1.4 Operadores OWAs

La segunda herramienta que compone la propuesta principal de este trabajo serán los Operadores OWA [5] (*Ordered Weight Aggregation*, Agregaciones Ponderadas basadas en Orden), conocidos por su aplicación dentro de la Lógica Difusa. Este tipo de operadores permiten modelar agregaciones habituales del lenguaje común, como el mínimo, el máximo o la media, y generalizarlas para crear modelos difusos sobre ellas (*soft max*, o máximo suavizado, por ejemplo).

Para esto, estos operadores se basan en una simple agregación ponderada, pero en lugar de aplicar los pesos a los elementos a agregar directamente, primero se realiza una ordenación de estos elementos.

1.5 Trabajos previos

En la literatura, ya se han realizado múltiples intentos de combinar la capacidad de los operadores OWA en CNNs [6]-[11]. Comentamos aquí algunas de ellas, a fin de ilustrar la motivación de nuestra propia propuesta.

La manera más explorada hasta el momento es la inserción de operadores OWA y otros operadores de agregación basados en medidas difusas [12], como las integrales Choquet o Sugeno, para combinar ensembles de clasificadores independientes [6]-[9]. En estos sistemas, realmente se pueden combinar clasificadores de cualquier tipo, pero son habituales los ensembles de CNN y otros tipos de redes neuronales, a fin de conseguir aumentar el campo de utilización o la precisión del sistema final. Por ejemplo, en clasificación de imágenes, emplear ensembles nos puede ayudar tanto a mejorar la capacidad de generalización de la red, como a directamente combinar redes independientes especializadas en casos concretos. El mecanismo suele funcionar entrenando y prediciendo con cada clasificador de forma completamente autónoma (de ahí la necesidad de emplear

clasificadores muy distintos), y realizando una agregación de las predicciones solo en la etapa final del proceso. Esta agregación, donde clásicamente se han empleado métodos de voto simples (mayoría simple, arrogancia, etc.) o agregaciones lineales, es donde podemos emplear medidas como los operadores OWA o integrales difusas [6].

Otra técnica habitual es el empleo de operadores de agregación basados en medidas difusas en el las capas de pooling de CNNs [10], [11]. En estas capas, donde agregamos una imagen por bloques para reducir su dimensionalidad, lo habitual es emplear como operadores el máximo y la media, pero podemos reemplazarlos por otros operadores. En [11], por ejemplo, los autores muestran como utilizar en una capa de pooling una agregación basada en una integral Choquet puede mejorar los resultados de la red.

También resulta de interés el trabajo realizado por Veal et al. [13], donde se desarrolla una Linear Order Statistic Neuron, una neurona artificial basada en un operador OWA. En principio esta neurona podría usarse como base para una red neuronal completa, reemplazando por las neuronas lineales que usamos habitualmente (no basadas en orden). En nuestro trabajo optaremos por una solución basada en aumentar, en vez de reemplazar, la información con información proveniente de OWAs.

Por último, debemos mencionar el trabajo de [14], que supone la inspiración original para nuestra propuesta. En este trabajo, los autores proponen la creación de una Capa Difusa, basada en operadores OWA. En su caso, la capa se integra en diferentes puntos de una CNN, recogiendo la información de la capa anterior y reemplazándola por completo por el resultado de aplicarle a esa información 6 operadores OWA predefinidos: máximo, mínimo, máximo suavizado, mínimo suavizado, media aritmética y un operador aleatorio. Estos operadores se aplican a nivel de canal, empleando como métrica de ordenación común a todos los operadores la entropía del canal. En este trabajo se aplicaba esta idea a un problema de segmentación semántica [15]. En nuestro caso, al intentar aplicarlo a problemas de clasificación de imagen nos encontramos rápidamente con una bajada importante de las métricas de precisión del sistema. En nuestra opinión, la cantidad de información que la red almacena de forma intrínseca en la ordenación de los canales de una capa convolucional es grande, y se pierde al realizar una operación de tipo OWA, suponiendo una pérdida de información importante para el sistema. Es por esto que nuestro enfoque se ha basado, no en la sustitución de la información de la red, si no en la aumentación de esta información. Además, en nuestro caso planteamos el aprendizaje autónomo de los parámetros de los operadores OWA, en vez de mantenerlos fijos como en [14].

1.6 Motivación

Los operadores OWA permiten una interesante perspectiva sobre ciertos tipos de información, y son capaces de interpretarla de maneras que no se pueden lograr fácilmente con otros tipo de operadores. Las redes neuronales, por su parte, han demostrado ya su gran capacidad de generalización, pero en principio la base sobre la que se apoyan es la de las agregaciones lineales simples, no ordenadas.

Nuestra propuesta será intentar dotar a las redes neuronales, en concreto a

las redes neuronales convolucionales diseñadas para la clasificación de imágenes, de la información que se puede obtener a partir de operadores de tipo OWA.

1.7 Objetivos

El objetivo de este trabajo es diseñar e implementar una nueva arquitectura de Red Neuronal Convolucional que incluya en sus capas operadores OWA, a fin de aumentar la información intrínseca de la red a través de estos operadores. En concreto, se analizarán las posibilidades de unión de ambos conceptos ya planteados, los trabajos previos, y sobre esta base se diseñará una nueva alternativa. Esta arquitectura de capa concreta (en adelante, capas OWA), se incluirá dentro de una Red Neuronal Convolucional estándar para comprobar si el funcionamiento de ésta mejora de forma significativa el rendimiento, así como su impacto en la complejidad computacional y tiempo de entrenamiento de la red original.

El resto de este trabajo se organiza de la siguiente manera. En el Capítulo 2 estudiaremos las bases de las redes convolucionales y operadores OWA, que nos sirven para enmarcar nuestro trabajo. Después, en el Capítulo 3 elaboraremos la arquitectura de capa concreta que estamos planteando, y coóo se integra en CNNs. En el Capítulo 4 describiremos el marco experimental sobre el que vamos a trabajar, incluyendo los experimentos que vamos a llevar a cabo. Los resultados se analizarán en el Capítulo 5. Por último, en el Capítulo 6 plantearemos las conclusiones que podemos obtener de nuestro trabajo y las posibles líneas de trabajo futuras sobre esta base.

Capítulo 2

Conceptos preliminares

2.1 Machine Learning

Hablamos de Machine Learning para referirnos, dentro del campo de la Inteligencia Artificial, a la rama dedicada al aprendizaje automático de las máquinas a través de la búsqueda de patrones en conjuntos de datos.

El objetivo suele ser doble: tanto realizar predicciones certeras sobre nuevos datos no conocidos en el entrenamiento, como el análisis de la propia información que dan los patrones reconocidos por el sistema.

Este tipo de aprendizaje se orienta especialmente a grandes cantidades de datos, como los que se manejan hoy en día en Internet o en sensores que vuelcan información en tiempo real. En estos casos, en función del tipo de datos de entrada tenemos varias maneras de trabajar, que se describen en las Secciones 2.1.1 a 2.1.4.

2.1.1 Supervisado

Llamamos aprendizaje supervisado al que se realiza sobre datos preetiquetados. Este es el formato más habitual de Machine Learning, donde tenemos un cierto número de ejemplos con cierta información x asociada (características del ejemplo), y estos ejemplos están etiquetados con un valor y, que puede ser discreto (clasificación en un número finito de clases) o continuo (regresión de un valor de característica real).

Asumimos que existe una función $x \rightarrow y$, que para cada entrada obtendría su etiqueta correspondiente, y buscamos un modelo que la aproxime. El proceso de entrenamiento del sistema será la búsqueda de esta aproximación, y después podremos introducir nuevos ejemplos no vistos y predecir sus etiquetas. Le estamos pidiendo al sistema que encuentre un modelo que transforme características de entrada en etiquetas de salida.

Ejemplos de este tipo de algoritmos serían las propias redes neuronales que usaremos en este trabajo, algoritmos de regresión lineal, árboles de decisión, support vector machines (SVM), etc.

2.1.2 No supervisado

En el aprendizaje no supervisado, en cambio, trabajamos con datos no etiquetados. En este caso, no se le da a la máquina un objetivo tan claro, si no que se busca usar la potencia de cálculo para probar y ajustar modelos variados, y localizar patrones de forma más general.

Este tipo de algoritmos se emplean para labores de modelado descriptivo, búsqueda de patrones... Un ejemplo interesante sería la realización de resúmenes de datos o incluso texto.

Algunos ejemplos de aprendizaje no supervisado son los algoritmos de clustering como k-means, algunas redes neuronales como los autoencoders, etc.

2.1.3 Semisupervisado

En la práctica, muchas veces nos encontramos con grandes cantidades de datos de lo que queremos conocer una característica concreta, pero que solo están etiquetados parcialmente. Este tipo de casos concretos se suelen denominar aprendizaje semisupervisado.

En general, la idea suele ser utilizar tanto la información proveniente de los datos etiquetados, como aquella intrínseca en la distribución de los datos no etiquetados, combinando técnicas de aprendizaje supervisado y no supervisado. Un ejemplo son los algoritmos de clustering que emplean unos cuantos datos etiquetados para señalar cuantos clusters se buscan y cuales son sus características principales, pero después depuran el conocimiento de estas características explorando los datos no etiquetados que, por proximidad, probablemente pertenezcan al mismo cluster.

2.1.4 Por refuerzo

Por último, también consideramos el aprendizaje por refuerzo. Este tipo de aprendizaje se basa en usar como fuente de información la respuesta del entorno ante las acciones de un cierto agente, controlado por la máquina.

Este tipo de problemas suelen definirse en base a un entorno concreto con un cierto número de acciones limitadas, y un mecanismo de refuerzo simple ante cada acción realizada, capaz de cambiar el estado del sistema. Este tipo de modelado es conocido como *Proceso de Decisión de Markov*.

Ejemplos de este tipo de aprendizaje serían las redes neuronales generativas adversarias, Q-learning, etc.

2.2 Clasificación y regresión

Como hemos señalado, dentro de los problemas de aprendizaje supervisado, nos suelen interesar dos en concreto: los problemas de clasificación y los problemas de regresión. Se distinguen principalmente por el tipo de variable a predecir:

- *Clasificación.* En este caso, la variable objetivo es una variable categórica y no ordenada. Por ejemplo la clasificación de proyectos en rentables o no rentables, la clasificación de imágenes en función de si contienen o no

ciertos elementos, la clasificación de textos en función de los temas que tratan, etc.

- *Regresión.* La variable objetivo es, o bien continua (un número real), o bien es discreta pero ordenada. Ejemplos habituales son la tasación de casas en función de sus características, predicciones meteorológicas (predicción de la temperatura en base a series históricas), etc.

En general, los métodos empleados para ambos problemas suelen ser similares, si bien la forma de aplicarlos suele cambiar. Por ejemplo, aunque empleemos una red neuronal para ambos tipos de problemas, emplearemos medidas de errores (funciones objetivo o de coste) diferentes: en el caso de clasificación una habitual es la *entropía cruzada*, mientras que para regresión el *error cuadrático medio* representa mejor el objetivo del problema.

Es muchos casos, ambos tipos de problemas son prácticamente equivalentes, y podemos alterar la definición de un problema para abordarlo con un sistema u otro. Por ejemplo, es habitual segmentar una variable objetivo continua en unas pocas clases, convirtiendo un problema de regresión en uno de clasificación. En el sentido contrario, podemos buscar medidas numéricas que indiquen indirectamente la clase del ejemplo, para convertir un problema de clasificación en uno de regresión.

2.2.1 Regresión logística

Un caso particular es la *regresión logística*. En este tipo de sistemas resolvemos un problema de clasificación con una variable objetivo binaria (con dos únicos valores, pertenece o no a una clase) a través de métodos de regresión. Para ello, modificamos el problema, considerando como variable objetivo la probabilidad de que cada ejemplo pertenezca o no a una de las clases.

2.2.2 Árboles de decisión

Respecto de la clasificación, uno de los modelos más claros, precursor de las redes neuronales que estudiaremos en este trabajo, son los árboles de decisión.

Un árbol de decisión se compone de una serie de pruebas y decisiones interrelacionadas, que acaban con la clasificación de un ejemplo cualquiera en una clase concreta. Se puede ver un ejemplo en la Figura 2.1.

Existen múltiples ejemplos de algoritmos para construir árboles de decisión a partir de una lista de ejemplos clasificados. Uno de los más simples es la división binaria recursiva, un algoritmo voraz que en cada paso escoge la decisión que minimiza una función de coste, habitualmente el índice de Gini, que mide la "pureza" de las hojas creadas a partir de esta decisión. La idea es escoger decisiones que separen fuertemente los elementos de la clase positiva y negativa.

2.3 Redes neuronales

Las redes neuronales biológicas, como las que se encuentran en el cerebro humano, sirven de inspiración para el diseño de redes neuronales artificiales. Los primeros modelos fueron propuestos en los años 40 [16], centrándose primero en

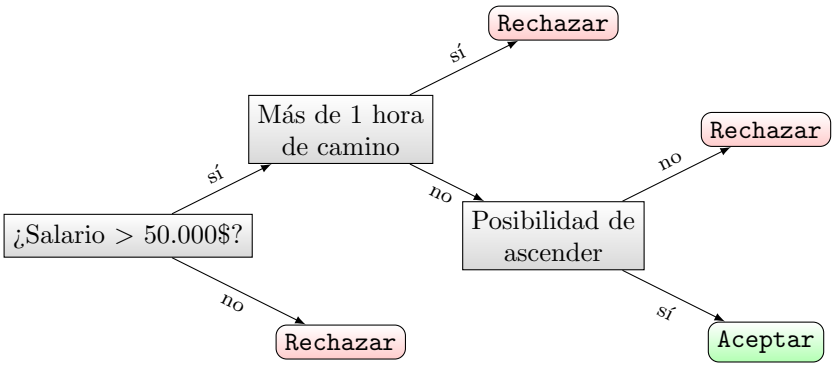


Figura 2.1: Ejemplo de un árbol de decisión.

la unidad más simple (la neurona o perceptron) y después construyendo modelos compuestos cada vez más complejos.

2.3.1 De la neurona al Perceptrón

El perceptron es uno de los modelos más simples de neurona artificial. Primero, observemos la estructura de una neurona biológica, mostrada en la Figura 2.2.

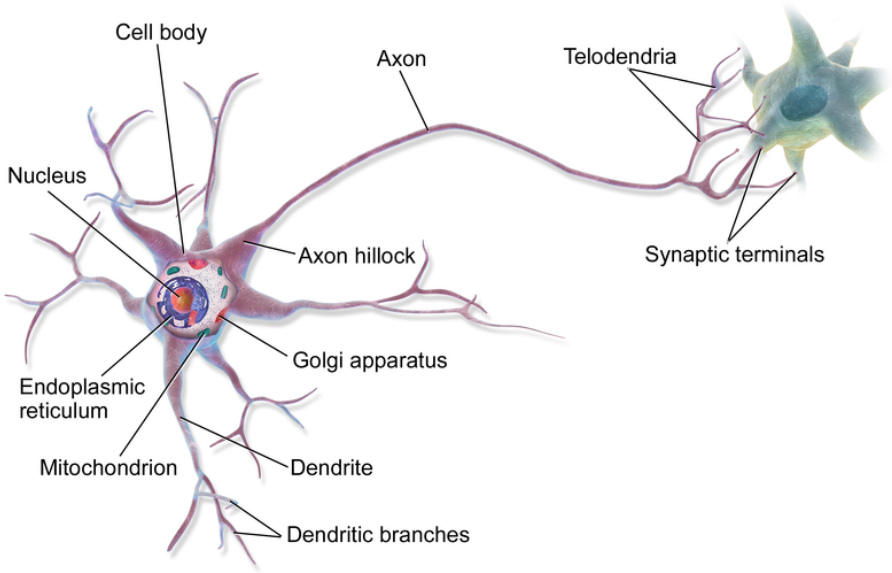


Figura 2.2: Diagrama de una neurona
Fuente: Wikimedia Commons

En la estructura de una neurona se reconocen tres partes diferenciadas:

- *Dendritas,* las terminaciones por las que una neurona recibe la información de las neuronas a las que está conectada (entrada).

- *Cuerpo de la neurona*, que procesa esas entradas, activándose o desactivándose.
- *Axón*, que transmite la activación de esa neurona hacia otras.

Si bien no se conoce completamente el funcionamiento de las sinapsis entre neuronas, este esquema simplificado es suficiente para plantear un modelo matemático, denominado perceptron. La Figura 2.3 representa de forma simple este modelo.

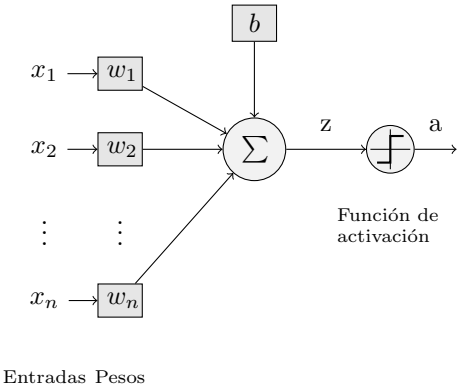


Figura 2.3: Esquema de un perceptron.

El modelo del perceptrón supone que la neurona funciona como una combinación lineal de los valores de activación de las neuronas de entrada, representados como x_i en el esquema. Estos pesos se agregan mediante combinación lineal, multiplicándolos por un vector de pesos w_i con el mismo tamaño que la entrada, n. Además, se agrega un peso especial b, denominado *bias*, que desplaza la activación completa, y no tiene entrada asociada (corresponde a una entrada constante 1). Tras agregar las activaciones de entrada de esta manera, se obtiene un valor real no acotado, que mediante una función de activación se convierte en el valor final a considerar.

Esta función de activación puede ser de varios tipos, siendo la más simple una función escalón (activar la neurona cuando la combinación lineal es mayor que 0). En caso de usar esta función, obtenemos la expresión matemática del perceptrón simple:

$$f(\mathbf{x}) = \begin{cases} 1 & \text{si } \mathbf{w} \cdot \mathbf{x} + b > 0, \\ 0 & \text{en otro caso} \end{cases} \tag{2.1}$$

Este perceptrón es ya capaz de realizar tareas simples de clasificación binaria, representando un hiperplano en el espacio de variables de la entrada (clasificación lineal).

2.3.2 Perceptrón multicapa / Redes Neuronales

Si bien el perceptrón es de por sí un clasificador razonable, fuera de problemas lineales no resulta de mucha utilidad. Es ahí donde las redes neuronales artificiales, agrupaciones de nodos o neuronas interconectados, pasan a ser importantes.

La arquitectura más simple para una red neuronal es el perceptrón multicapa, formada por perceptrones sencillos con activación sigmoide como nodos. Este tipo de arquitectura suele ser *densa*, esto es, cada nodo de cada capa tiene como entrada a todos los nodos de la capa anterior, y sirve como entrada a todos los nodos de la capa siguiente. Podemos ver un diagrama de esta arquitectura en la Figura 2.4, en este caso de una red con 4 neuronas en la capa de entrada, una capa oculta de 5 neuronas, y una neurona en la capa de salida.

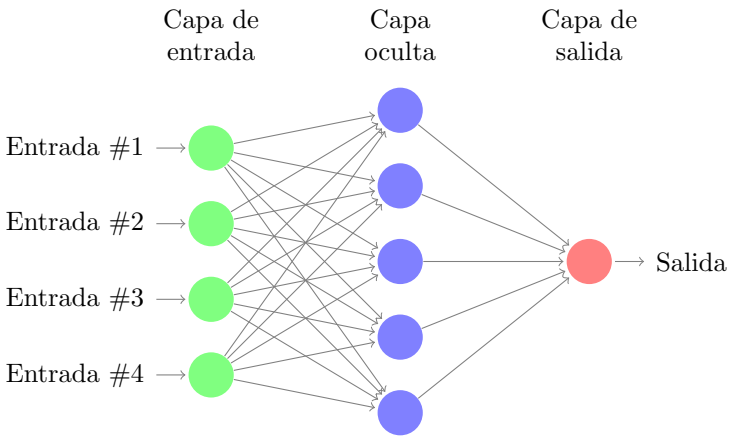


Figura 2.4: Ejemplo de perceptrón multicapa.
Fuente: Texample

En la práctica, las neuronas de una red suelen ser versiones más avanzadas del perceptrón, que emplean funciones de activación no lineales (en caso de usar lineales, cualquier estructura de más de una capa oculta de profundidad podría simplificarse a una con una sola capa oculta). En concreto, son habituales las activaciones sigmoides como tanh y logística:

$$y(z) = \tanh(z) \tag{2.2}$$

$$y(z) = (1 + e^{-z})^{-1} \tag{2.3}$$

Y en el caso de Deep Learning, es habitual también la función de activación ReLU (y variantes):

$$y(z) = max(0, z) \tag{2.4}$$

2.3.3 Retropropagación

En Redes Neuronales distinguimos dos modos de funcionamiento básico: propagación hacia adelante y hacia atrás (retropropagación).

La propagación hacia adelante sucede cuando tomamos un ejemplo y predecimos su salida utilizando la red. Básicamente empezamos calculando las activaciones de la primera capa en función de la entrada del ejemplo, de ahí tomamos esas activaciones y calculamos las de la segunda capa, y así sucesivamente hasta obtener una salida de la red, la predicción para ese ejemplo.

En el inicio de las redes neuronales, cuando aún no se habían desarrollado los algoritmos de aprendizaje, los pesos de la red se escribían a mano, y la propagación hacia adelante era el único modo de funcionar de las redes.

La retropropagación surgió de la necesidad de entrenar la red. En este caso, recorremos la red en el sentido contrario, partiendo de la salida y haciendo nuestro camino hacia la entrada. Primero, realizaremos una etapa de propagación hacia adelante sobre la red, con un ejemplo ya etiquetado. Estableceremos una función de coste o *loss* a minimizar, y calcularemos el error que se ha cometido en la salida de ese ejemplo, respecto de su variable objetivo deseada. A partir de ese error calcularemos la derivada parcial de ese error respecto a los pesos de las neuronas de la última capa de la red, y propagaremos el error hacia la capa anterior. Repitiendo este proceso de propagar errores y calcular derivadas para cada capa, obtenemos suficiente información como para actualizar los pesos de toda la red, ajustándola para que clasifique ese ejemplo más cerca de su valor real.

Este proceso, en la práctica se aplica de manera algo más compleja, pasando a la red múltiples ejemplos de entrada en un lote, procesándolos al mismo tiempo, y actualizando los pesos de la red respecto de la combinación de derivadas para esos ejemplos. Este tipo de acercamiento nos permite entrenar la red mucho más rápidamente.

2.4 Deep learning

En origen, las redes neuronales eran de reducido tamaño, debido especialmente a la dificultad y coste de entrenarlas. Hoy en día sin embargo, la técnica se ha ido decantando en muchos problemas por las redes cada vez más profundas, lo que denominamos Deep Learning.

Este tipo de redes presentan algunos problemas concretos, pero en general los modelos generados pueden alcanzar una mayor complejidad con facilidad. Problemas como la clasificación de imágenes, donde es casi imposible imaginar una red con una sola capa oculta capaz de detectar, por ejemplo, gatos, con Deep Learning pasan a ser razonables, al especializarse cada capa de la red en detectar características a un determinado nivel. En el caso de las imágenes, por ejemplo, las primeras capas de la red detectan características locales y genéricas como bordes o puntos, y conforme la información avanza por las capas se detectan características cada vez más complejas, como ojos u orejas. En la Figura 2.5 se ilustran diferentes características aprendidas por una red en diferentes capas.

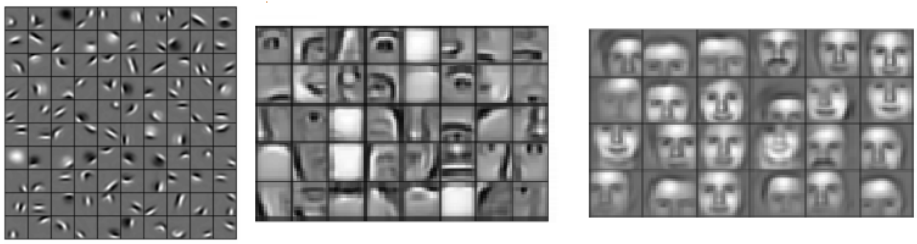


Figura 2.5: Características aprendidas por una red en diferentes capas.
Fuente: [17]

2.4.1 Regularización

Uno de los problemas habituales en machine learning y en concreto en Redes Neuronales es lo que conocemos como overfitting o sobreajuste. Las Redes Neuronales tienen la capacidad de modelar sistemas arbitrariamente complejos, y eso pude llevar a diferentes problemas cuando la complejidad del modelo empleado y la de la relación real entre variables no encajan.

En la Figura 2.6 podemos ver un ejemplo de los problemas que pueden surgir en el entrenamiento de redes neuronales y otros clasificadores. En esta figura, vemos en rojo la distribución real sobre la que se han generado unos ejemplos, en azul el modelo que estamos ajustando, y los puntos azul oscuro corresponden a los ejemplos recogidos, que incluyen algo de ruido. En la primera imagen, el modelo empleado es demasiado simple, y no recoge la verdadera distribución de los datos. A esto le denominamos subajuste o *underfitting*. La segunda hace un ajuste correcto. En la tercera, vemos que el modelo se ajusta perfectamente a los datos, pero en vez de recoger la verdadera distribución de estos, se ajusta para modelar el ruido presente, haciendo que el modelo se aleje de la distribución real.

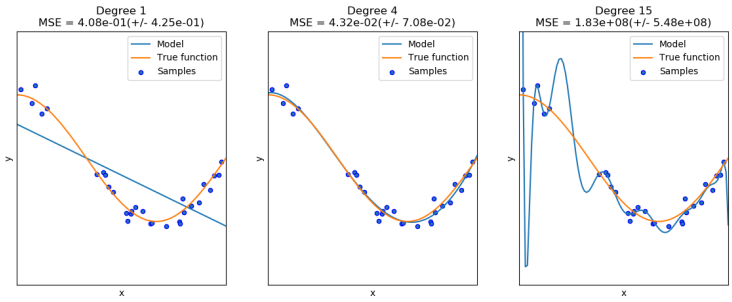


Figura 2.6: De izquierda a derecha: subajuste, ajuste correcto, sobreajuste.
Fuente: Scikit learn

En redes neuronales el sobreajuste se da con facilidad, ya que el proceso de aprendizaje se va ajustando progresivamente a los datos disponibles, y mientras la red sea lo suficientemente compleja puede ajustarse indefinidamente, hasta sobreajustar. Denominamos **regularización** a los sistemas que nos permiten evitar ese sobreajuste, garantizando que la red aprenda un modelo bien ajustado y conforme a los datos, sin sobreaprender los pequeños detalles irrelevantes o ruido que pueda encontrarse en estos.

Dropout

Una forma bastante habitual de regularización es el dropout [18]. La idea base del dropout es evitar que la red dependa excesivamente de unas pocas conexiones, y distribuir el conocimiento por toda red.

En concreto, la técnica se basa en ignorar por completo algunas neuronas durante el entrenamiento, cambiando qué neuronas se bloquean en cada paso de entrenamiento (cada lote procesado), de manera que todas entrenen en algún momento, pero ninguna neurona pueda volverse excesivamente vital. En la Fi-

gura 2.7 podemos ver cómo quedaría una red antes y después de aplicarle esta técnica.

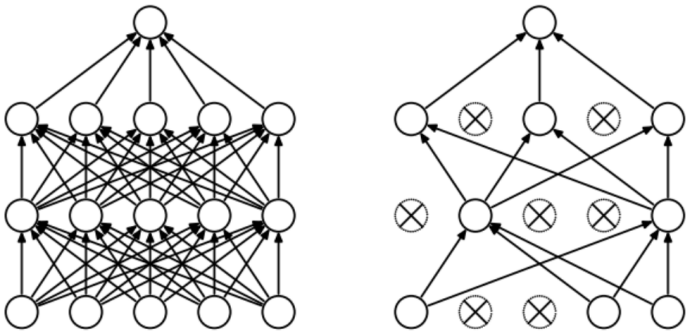


Figura 2.7: Diagrama de una red antes y después de aplicar dropout.
Fuente: [18]

ADAM

Otra técnica interesante se basa en el control de los hiperpárametros de aprendizaje, la manera de ir aprendiendo de la red. Un ejemplo notable de esta técnica es Adam [19], un algoritmo de optimización del ratio de aprendizaje.

Los algoritmos clásicos de aprendizaje basados en retropropagación suelen depender de un ratio de aprendizaje y del gradiente calculado en la última pasada de retropropagación de la red. Adam, amplia varios de estos, y se caracteriza por:

- Momento. Adam emplea momentos, de manera que agrega tanto los ratios de aprendizaje de la última iteración como los anteriores, teniendo en cuenta la dirección en la que se han ido ajustando los pesos recientemente. En concreto, Adam emplea estimaciones de los momentos primero y segundo del gradiente.
- Ratios de aprendizaje individuales para cada parámetro, de manera que si algún parámetro tiene una variación atípica (explosión del gradiente) no afecte al resto.

Estas características lo convierten en un potente sucesor a los métodos anteriores, especialmente RMSprop y AdaGrad [20], en los que se basa.

Batch normalization

Batch Normalization [21] (normalización por lotes) es una técnica crucial en el entorno del deep learning, que permite acelerar el aprendizaje y estabilizarlo en redes neuronales profundas.

La idea simple detrás del algoritmo es la de normalizar las activaciones de cada capa de la red. Este concepto se planteó principalmente para gestionar lo que se conoce como problema de desplazamiento de la covarianza interna (*internal covariance shift problem*). Este fenómeno se da en las redes muy profundas,

donde un cambio pequeño en un punto de la red puede extenderse por toda ella. Esto supone un origen de inestabilidad enorme en redes profundas, donde el reajustar ligeramente los pesos de la entrada, por ejemplo, puede trastocar los resultados finales de la red.

Más adelante se ha comprobado que, además, normalizar las activaciones en cada capa ayuda a "suavizar" los valores de la función objetivo, facilitando que los algoritmos de aprendizaje converjan más fácilmente [22].

2.4.2 Redes Neuronales Convolucionales

Uno de los modelos más conocidos actualmente de Deep Learning son las Redes Neuronales Convolucionales (CNNs, *Convolutional Neural Networks*), diseñadas para recoger y emplear la dimensión espacial de las entradas matriciales, en concreto de imágenes. La idea es que en una imagen, la relación entre activaciones de píxeles que están lejos entre sí es poco relevante, y lo que más debe pesar para tratar imágenes son las relaciones entre activaciones de píxeles cercanos.

A la hora de implementar, esto nos lleva a emplear un tipo de operación y neurona particulares: las capas convolucionales.

Una convolución es una operación matemática que se realiza entre una matriz de entrada y una matriz núcleo (también llamado *kernel*). En dos dimensiones, el caso más común, tenemos:

$$g(x,y) = \omega * f(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} \omega(s,t) f(x-s, y-t) \tag{2.5}$$

Donde $f(x,y)$ es la imagen original, ω es el núcleo de la operación, una matriz definida para $-a \leq s \leq a$ y $-b \leq t \leq b$.

Podemos ver un ejemplo de convolución en el sentido matemático en la Figura 2.8, en este caso con un núcleo 3×3 sobre una matriz de entrada de 7×7.

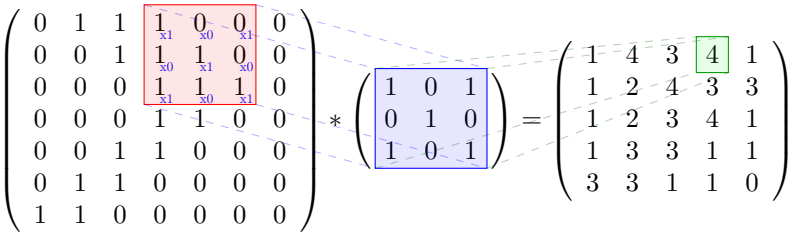


Figura 2.8: Ejemplo de una convolución.

Podemos entender una convolución como una comparación de patrones locales sobre una imagen: el núcleo representa un patrón (un borde en diagonal, por ejemplo), y al recorrer la imagen original buscamos los sitios donde el patrón encaja. La imagen resultante representa los lugares donde la original presentaba el patrón buscado.

En una CNN creamos capas de neuronas que realizan una convolución sobre la entrada. Las salidas de estas neuronas se "apilan", constituyendo una imagen con tantos canales como neuronas tenga la capa, y esta imagen se emplea como entrada de la siguiente capa.

2.4.3 Pooling

Normalmente, las CNNs suelen emplear, además de capas de convolución, capas de pooling. Este tipo de capas está orientado a resumir la información de la imagen, reduciendo su tamaño, de manera que no importa tanto la posición exacta de una característica detectada por un filtro, como su presencia general. Esto permite simplificar los filtros de las capas siguientes, haciéndolos menos sensibles a la variación en la posición exacta de los detalles.

La forma de implementación más habitual de estas capas es a través de una agregación de la imagen por bloques. Se selecciona un tamaño de bloque (habitualmente 2×2), y se realiza una agregación (normalmente un máximo o una media) de los píxeles dentro del bloque. Esta agregación operación suele realizarse en paralelo en todos los canales de la imagen. Vemos un ejemplo de max-pooling (pooling utilizando el máximo como agregación) en la Figura 2.9.

$$\left(\begin{array}{cc|cc} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right) \rightarrow \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{2.6}$$

Figura 2.9: Ejemplo de una operación max-pooling.

2.4.4 Ejemplos de arquitecturas

Con estas herramientas, surgen diversas arquitecturas de CNNs concretas que se han ido sucediendo en el tiempo como estándares para la clasificación de imágenes:

- LeNet [23], una de las arquitecturas clásicas de CNNs, con dos capas convolucionales y dos capas de pooling que utilizan como agregación la media.
- La familia VGG [24], que emplearemos en este trabajo. Se muestra un diagrama de la arquitectura en la Figura 2.10, en este caso de su versión VGG16. Esta arquitectura fue una de las pioneras en cuanto a profundidad, introduciendo en su forma más popular 16 capas con pesos (13 de ellas convolucionales, y otras 3 densas para clasificación).
- ResNet [25]. Esta arquitectura se diseñó con el objetivo de poder añadir un número arbitrario de capas sin que estas llegase a tener un impacto negativo en la precisión de la red. Para esto, se emplea un bloque especial "residual", que combina bloques convolucionales que pueden ser desactivados con bloques identidad, que no hacen nada. Esto permite a la red gestionar la complejidad final del sistema.

- DenseNet [26]. En este caso, se emplean bloques "densos", de una manera similar a la que realizaba ResNet. Sin embargo, la diferencia principal es que, en vez de pasarse el estado sólo entre bloques sucesivos, esta red hace que cada bloque le pase su salida a todos los siguientes. En la práctica, el objetivo es que cada bloque de la red tenga toda la información de salidas de los anteriores, y pueda trabajar con ella.

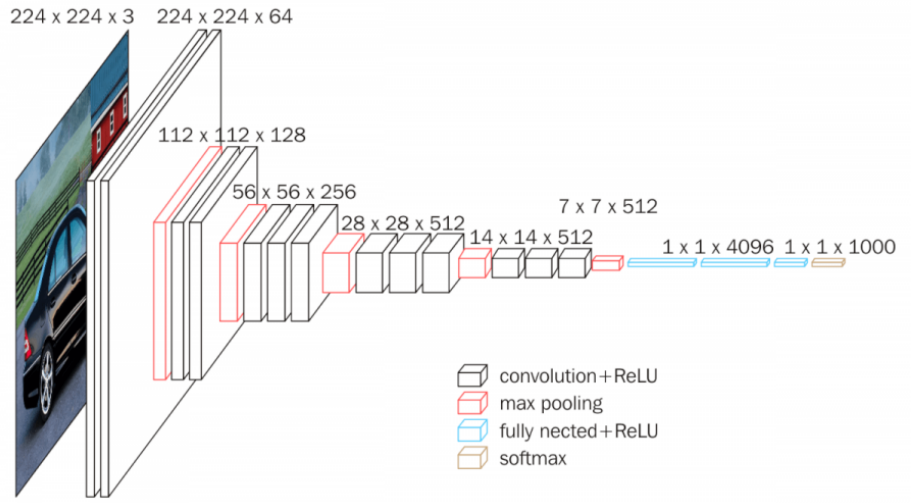


Figura 2.10: Arquitectura VGG16.
Fuente: Neurohive

2.5 Evaluación de modelos

La evaluación de modelos de Machine Learning en general, se suele basar en la creación de diferentes particiones para el estudio. En concreto, sobre un set de datos etiquetados, solemos separarlos en hasta tres particiones:

- Partición de entrenamiento (*training set*). Es la partición de datos más grande normalmente, y sobre la que el sistema realiza el grueso del aprendizaje.
- Partición de validación (*validation set*). Es una partición más pequeña, empleada para comprobar el funcionamiento de la red con diferentes conjuntos de hiperparámetros y seleccionar los hiperparámetros y configuraciones óptimas.
- Partición de test (*test set*). De tamaño similar a la partición de validación, se emplea como prueba final de la red. Estos datos no deben emplearse más que para obtener las métricas finales del sistema, estando completamente aislados del proceso de entrenamiento.

El objetivo principal de manejar particiones separadas es evitar obtener métricas demasiado optimistas del sistema, algo habitual si empleamos información de los datos sobre los que se toman durante el entrenamiento y ajuste.

En la práctica, sin embargo, puede resultar complicado tener datos suficientes para las tres particiones, y es habitual separar los datos en solo dos: entrenamiento y validación/test.

2.6 Operadores OWA

Los operadores OWA fueron propuestos inicialmente por Yager [5]. Se trata de mapeos $F\colon \mathcal{R}^n \to \mathcal{R}$ basados en una colección de pesos $W = [w_1, \ldots, w_n]$, habitualmente con la condición de que los pesos estén en el rango $w_i \in [0, 1]$ para cada $i = 1, \ldots, n$ y de que sumen 1 $\sum_{i=1}^{n} = 1$. Se definen por:

$$F(a_1, \ldots, a_n) = \sum_{j=1}^{n} w_j b_j \tag{2.7}$$

donde b_j representa el j-ésimo elemento más grande de a.

Esta definición puede descomponerse en dos partes a diseñar para aplicar un operador OWA: la ordenación y la agregación en sí. Si bien en el caso simple de que el vector a se componga de números naturales o reales, la ordenación resulta trivial, al aplicar OWAs a vectores de elementos de otros tipos tendremos que definir antes la ordenación de estos.

Algunos ejemplos notables serían máximo ($W = [1, 0, \ldots, 0]$), mínimo ($W = [0, \ldots, 0, 1]$), y la media aritmética ($W = \left[\frac{1}{n}, \ldots, \frac{1}{n}\right]$).

Capítulo 3

Propuesta

En este capítulo describiremos la arquitectura propuesta para la integración de capas OWA en arquitecturas CNN. En concreto, en la Sección 3.1 introduciremos la estructura general que va a seguir nuestro modelo de inserción. En la Sección 3.2 describiremos las funciones de ordenación que vamos a emplear, al trabajar por capas. Por último, en la Sección 3.3 describiremos la manera de realizar la agregación en sí y de gestionar los pesos de la capa.

3.1 Capa de agregación OWA

A la hora de implementar la nueva capa basada en OWA, hemos empleado la siguiente arquitectura, planteada en 3 fases:

- Ordenación de los canales de entrada en función de una métrica.
- Agregación de los canales de entrada ordenados ponderada por pesos.
- Concatenación de los nuevos canales a los canales de la capa anterior.

Las dos primeras fases hacen referencia a las dos partes habituales de aplicación de un operador OWA, la ordenación y la agregación en sí. Si bien normalmente la ordenación es evidente (ordenación de valores de mayor a menor), en nuestro caso al realizar una ordenación de canales debemos emplear métricas a nivel de canal para ordenarlos. La agregación se realiza mediante una combinación lineal de las capas con un vector de pesos.

La última fase supone la combinación de la información de los canales originales con la de los nuevos canales basados en operadores OWA. Si bien inicialmente se exploraron otras mecánicas, como la propuesta en [14], donde simplemente se reemplazaban los canales originales por los nuevos, en nuestra experiencia la pérdida de información es demasiado grande y el resultado no es adecuado para labores de clasificación de imágenes. En este trabajo nos hemos centrado en un sistema de concatenación, según el cual recogemos los C_{in} canales de entrada de cada imagen de un batch, aplicamos operadores OWA para obtener C_f nuevos canales, y sencillamente los concatenamos para obtener $C_{out} = C_{in} + C_f$ canales de salida. Esto supone, además, incrementar el número de canales de entrada de la siguiente capa de la red, que en nuestro caso suele ser una capa convolucional.

La arquitectura se muestra en la Figura 3.1.

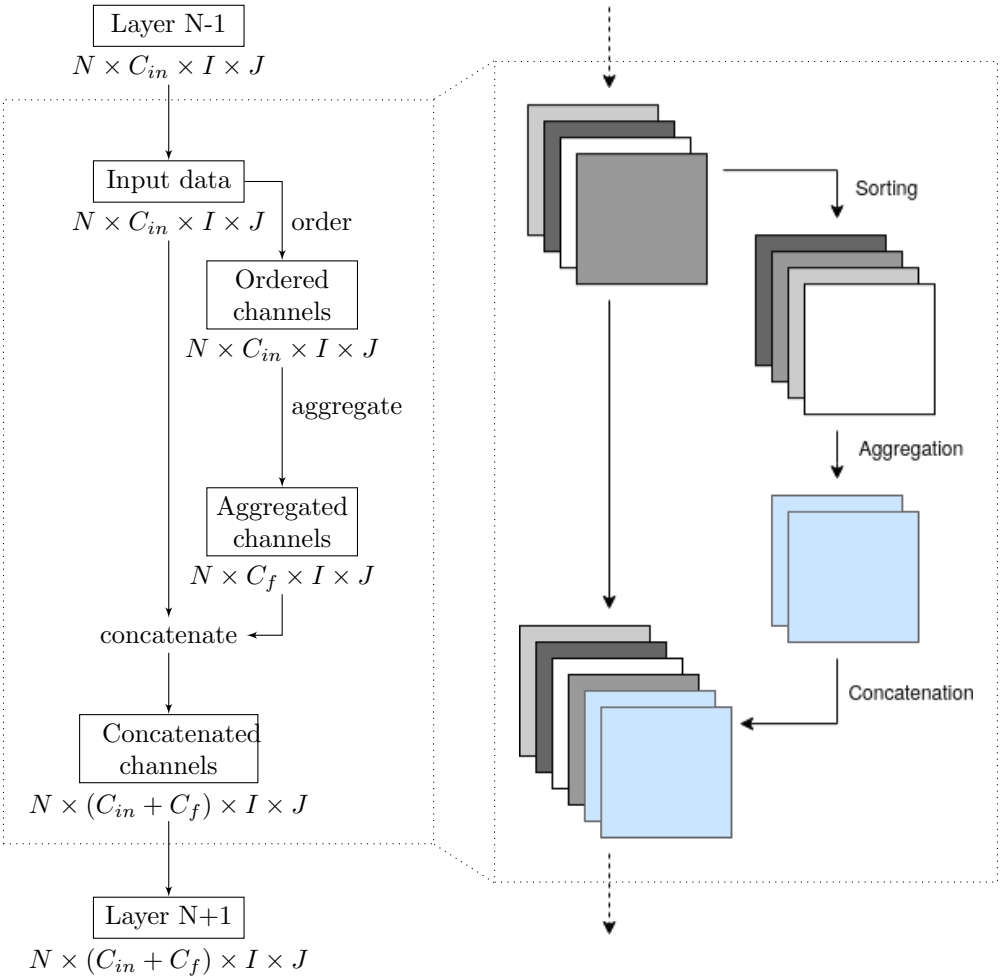


Figura 3.1: Estructura de la capa OWA.

3.2 Función de ordenación de canales

Normalmente las agregaciones OWA se definen como una suma ponderada sobre una serie de valores ordenados, donde los pesos de la suma se encuentran en el intervalo unidad y suman en total 1. En nuestro caso, al buscar realizar agregaciones de capas de activaciones de una red neuronal (que podemos tratar como matrices o imágenes de un solo canal), tenemos que definir el tipo de ordenación que empleamos, además de los pesos de la agregación.

Para esto, emplearemos funciones de ordenación de canales, que nos dan una métrica sobre la que ordenar los canales de una imagen. Serán funciones que toman una imagen de $I \times J$, con I el número de filas y J el número de columnas, y con un solo canal, y obtienen un medida concreta para esa imagen. En la implementación de este trabajo, empleamos funciones que procesan una matriz de $N \times C_{in} \times I \times J$, donde N es el número de imágenes, C_{in} el número de canales de cada imagen, I el número de filas y J el número de columnas, y obtienen una matriz de $N \times C_{in}$, donde cada elemento es el valor de la medida para cada canal de cada imagen. En nuestro caso, estas matrices e imágenes se corresponderán con las salidas intermedias de capas de una red neuronal, por lo que los píxeles se corresponderán con las activaciones de cada neurona en una capa concreta.

Como medidas hemos propuesto:

- *Entropía del canal*, medida por la entropía de Shannon, para el valor de cada píxel:

$$H(X) = -\sum_{i=1}^{I}\sum_{j=1}^{J} x_{ij} \log x_{ij} \tag{3.1}$$

  Como esta función esta diseñada originalmente para trabajar sobre vectores con sus elementos en el rango $x_{ij} \in [0, 1]$, y donde $\sum_{i=1}^{I}\sum_{j=1}^{J} x_{ij} = 1$, primero empleamos la función softmax sobre X para normalizar la entrada:

$$\text{Softmax}(x_{kl}) = \frac{e^{x_{kl}}}{\sum_{i=1}^{I}\sum_{j=1}^{J} e^{x_{ij}}} \tag{3.2}$$

  De forma intuitiva, podemos entender la medida de entropía como una medida del desorden de un conjunto de valores. En este caso, imágenes que contengan poca información (con valores de activación muy similares a lo largo de toda la imagen) tendrán una entropía máxima, mientras que imágenes con información muy clara (con unos cuantos valores concretos con una activación muy diferente al resto), tendrán entropía mínima.

- *Suma de los valores del canal*,

$$S(X) = \sum_{i=1}^{I}\sum_{j=1}^{J} x_{ij} \tag{3.3}$$

- *Variación Total* [27]. Calculamos las diferencias entre cada píxel y sus vecinos contiguos, tanto vertical como horizontalmente, y agregamos los valores absolutos de estas diferencias en toda la imagen.

$$TV_v(X) = \sum_{i=2}^{I}\sum_{j=1}^{J} |x_{i,j} - x_{i-1,j}| \tag{3.4}$$

$$TV_h(X) = \sum_{i=1}^{I} \sum_{j=2}^{J} |x_{i,j} - x_{i,j-1}| \tag{3.5}$$

$$TV(X) = TV_v(X) + TV_h(X) \tag{3.6}$$

Variación Total (TV), definida en [27], es una medida que considera la dimensión espacial de la imagen, y nos indica cuanta variación existe entre un píxel y sus vecinos. La medida de TV será alta cuando tengamos cambios radicales en la imagen, donde cada píxel se encuentre rodeado por píxeles de valor muy diferente, y será baja en imágenes con degradados suaves o zonas de activación uniforme.

- *Mediana de los valores del canal.* Otro operador OWA clásico es la mediana, que da un valor medio de activación resistente a los valores atípicos,

$$M(X) = median(x_{11}, \ldots, x_{IJ}) \tag{3.7}$$

donde el operador mediana devuelve el valor de la posición $ceil(I \cdot J/2)$ de los valores de X ordenados de mayor a menor, si $I \cdot J$ es impar, o la media aritmética de los valores de las posiciones $I \cdot J/2$ y $I \cdot J/2 + 1$ de los valores de X ordenados de mayor a menor, si $I \cdot J$ es par.

- *Máximo de los valores del canal.* En este contexto, el valor del píxel más activado del canal,

$$MAX(X) = \text{máx}(x_{11}, \ldots, x_{IJ}) \tag{3.8}$$

Además, se han creado dos medidas adicionales, que servirán para evaluar el impacto de los parámetros de la agregación OWA independientemente de su ordenación:

- *No ordenamiento.* Esta medida realiza una función identidad, asociándole a la primera capa el valor 1, a la segunda el 2, y así sucesivamente). Esto supone que la ordenación de canales mantenga el orden.

- *Ordenación aleatoria.* En este caso asignamos un valor aleatorio a los canales, haciendo que se ordenen de forma aleatoria.

Planteadas estas medidas, podemos ordenar fácilmente los canales de menor a mayor, y a partir de ahí realizar el siguiente paso, la agregación.

Como caso especial, también se ha implementado una mecánica de ordenación por píxel, donde en lugar de ordenar las capas independientemente, se realiza una ordenación simple de los valores que corresponden al mismo píxel $I \times J$ de cada canal de la imagen.

3.3 Función de agregación de capas

Para realizar la agregación nos podemos ceñir más fielmente al concepto de OWA clásico. Establecemos una función que recoja una matriz de $C_{in} \times I \times J$, donde C_{in} es el número de capas de la imagen, I el número de filas y J el de columnas, y obtenga una salida de $C_f \times I \times J$ donde C_f es el número de capas generadas, I el número de filas y J el de columnas. Estas C_f capas se generarán

a partir de combinaciones lineales de las C_{in} capas de entrada, a partir de una matriz de pesos.

A la hora de implementar, se consideran dos variables principales: aprendizaje de los pesos e inicialización de pesos.

3.3.1 Aprendizaje de los pesos

- *Pesos estáticos.* En este caso trabajaríamos con unos pesos predefinidos al inicio de la ejecución, y que no serían actualizados nunca. Sería el mismo funcionamiento que en [14], donde se inicializaban los pesos a ciertos operadores OWA clásicos (min, max, soft-min, soft-max, media y un OWA aleatorio). En nuestro caso, vamos a considerar más posibilidades de inicialización de los pesos, pero igualmente no los actualizaremos mediante retropropagación.

- *Pesos dinámicos.* En esta segunda implementación sí que consideramos los pesos para el aprendizaje. Esto nos lleva a tener algunas consideraciones especiales, ya que los valores de los pesos aprendidos a priori no tienen por qué cumplir las restricciones estándar de los operadores OWA (pesos $W = [w_1, \ldots, w_n]$, donde $w_i \in [0, 1]$ para todo $i = 1, \ldots, n$ y $\sum_{i=1}^{n} = 1$).

  Para garantizar que estas restricciones se cumplan a la hora de aplicar los pesos en la agregación, introducimos un paso extra de normalización de los pesos. En este paso, primero aplicamos la función ReLU a todos los pesos, eliminando los pesos negativos. Después, dividimos por la suma de los pesos, para garantizar que la suma total sea 1.

  $$ReLU(x) = max(x, 0) \tag{3.9}$$

  $$w_j = \frac{ReLU(x_j)}{\sum_{i=1}^{C_f} ReLU(x_i)} \tag{3.10}$$

  El resultado es un vector de pesos que puede ser aplicado directamente.

3.3.2 Inicialización de los pesos

Empleamos los siguientes tipos de inicialización de pesos:

- *Inicialización aleatoria.* Inicializamos generando pesos con pesos aleatorios las C_f agregaciones de salida. Cada peso se genera a partir de una distribución uniforme $U(0, 1)$. Esta inicialización resulta de poco interés en el caso de que los pesos se mantengan estáticos, pero resulta la más evidente para los pesos que se vayan a actualizar y aprender.

- *Inicialización trim.* Inicializamos los pesos a una matriz identidad truncada. Esto supone que la primera agregación de salida, inicialmente, se corresponda con la primera capa tras la ordenación (vector de pesos de agregación $W_1 = [1, 0, \cdots, 0]$), la segunda salida con la segunda capa tras la ordenación (vector $W_2 = [0, 1, \cdots, 0]$), y así sucesivamente.

  Este tipo de inicialización se corresponde con la operación de copiar directamente las C_f primeras capas de la salida después de ordenar, en el caso de nuestra implementación las capas de mínima métrica.

- *Inicialización trim-inversa.* Se realizar de la misma manera que con la inicialización trim, pero se copian como salidas las últimas C_f capas de la entrada en lugar de las primeras.

  Esta inicialización es complementaria a la trim, garantizando que en caso de que la información interesante para la red sea la de máximo en lugar de la de mínimo, se encuentre disponible.

- *Inicialización doble*: Se copian como salidas las primeras $C_f/2$ y las últimas $C_f/2$ capas de la entrada.

  Esta inicialización combina trim y trim-inversa, de manera que le ofrecemos a la red tanto las capas con métricas máximas como mínimas al mismo tiempo.

Capítulo 4

Marco experimental

4.1 Fastai / Pytorch

Para implementar los experimentos de este trabajo, hemos optado por emplear Fastai, una librería basada en Pytorch. En concreto, se han empleado las versiones 1.0.58 de Fastai y 1.3.1 de Pytorch, desplegadas sobre el entorno Anaconda 3, que facilita la instalación y gestión de paquetes de Python.

La motivación principal para decantarnos por Fastai es su simplicidad de uso y despliegue, y la disponibilidad de tener múltiples avances recientes en el campo del Deep Learning ya disponibles e implementados. Por ejemplo, la librería dispone de redes VGG preentrenadas ya disponibles (si bien hemos acabado implementándola desde cero por detalles técnicos), y de políticas de aprendizaje potentes, como es la política de gestión del ratio de aprendizaje 1cycle [28]. Esta política permite acelerar enormemente el aprendizaje de la red, facilitándonos el realizar más pruebas en el mismo tiempo.

Por otro lado, Pytorch destaca por su flexibilidad frente a otras librerías similares. En concreto, dispone de lo que se denomina grafo dinámico. Esta mecánica permite alejarse de los sistemas habituales donde se define el grafo de la red y después se ejecuta (como se emplea en TensorFlow, por ejemplo), y permite alterarlo en tiempo real. En nuestro caso nos permite, por ejemplo, modificar la red "en caliente", alterando una red ya entrenada para añadirle capas OWA a posteriori.

4.2 Datasets

Para probar la red hemos empleado principalmente dos datasets, CIFAR10 y CIFAR100 [29]. Ambos datasets constan de 60.000 imágenes en color, con una resolución de 32×32 píxeles. En el caso de CIFAR10 estas imágenes están clasificadas en 10 clases diferentes, sin solapamiento, y en el de CIFAR100 en 100.

Para entrenamiento y validación, ambos datasets tienen dos particiones ya predefinidas, con 50.000 imágenes para entrenamiento y 10.000 para validación. En ambos casos, las clases están perfectamente balanceadas, teniendo CIFAR10 5.000 imágenes de entrenamiento y 1.000 de validación para cada clase, y CIFAR100 500 de entrenamiento y 100 de validación por clase.

El motivo principal de elegir estos datasets es que, debido a su pequeño tamaño, la ejecución del aprendizaje es bastante rápida. En nuestro caso, al querer probar múltiples configuraciones, y tener una combinatoria de parámetros tan elevada, nos conviene poder realizar múltiples pruebas en una cantidad de tiempo razonable.

Para comprobar la capacidad de la red en un tamaño de imagen superior, también realizaremos un último experimento empleando el dataset Imagenette, una versión reducida del dataset de Imagenet [30]. Imagenette consta de 10 categorías fácilmente distinguibles, con 12,894 imágenes en una partición de entrenamiento y 500 en una partición de validación. Los ejemplos se encuentran distribuidos de forma equilibrada, con 50 ejemplos por clase en validación y 1300 en entrenamiento (salvo una clase con 1194 ejemplos en entrenamiento). Estas imágenes son a color, con un tamaño de imagen variable (en nuestro caso lo redimensionaremos a 256×256 píxeles).

4.3 Arquitectura

La capa propuesta, tal y como se presenta, podría insertarse en diferentes tipos de arquitecturas, especialmente las basadas en CNNs. En concreto, para nuestra implementación hemos seleccionado una red basada en VGG13, una de las variantes de la arquitectura VGG [24]. Esta arquitectura es uno de los estándares actuales en Deep Learning, y obtiene buenos resultados en un gran número de aplicaciones. En la Tabla 4.1 detallamos la versión concreta que estamos empleando, junto con los puntos donde vamos a insertar nuestras capas OWA.

Esta red ha sido ligeramente adaptada para nuestros datasets. Al tener CIFAR un tamaño de imagen relativamente pequeño, de 32×32 píxeles, al llegar a las últimas capas nos encontramos con imágenes extremadamente pequeñas, y en concreto llegamos a la capa del clasificador con una imagen de 1×1 píxeles. En este caso, pruebas preeliminares nos indican que sustituir la estructura con 3 capas densas de la propuesta original de VGG por un simple clasificador linear es suficiente, obteniendo los mismos resultados en una cantidad de tiempo menor. Esta misma modificación se encuentra en otros trabajos que aplican VGG a CIFAR [31]. En el caso de CIFAR10, el clasificador final tendrá 10 neuronas de salida, y en el de CIFAR100, 100.

Para Imagenette empleamos la misma estructura basada en VGG13 con un clasificador ligeramente reducido también, al tener solo 10 clases de salida. En este caso el clasificador estará compuesto por 3 capas densamente conectadas con 256, 256 y 10 neuronas cada una.

La elección de esta red en particular se debe sobre todo a su sencillez. Si bien otros modelos posteriores, como ResNet [25], la superan en precisión en los datasets más habituales, en general tienen arquitecturas más complejas y profundas. En nuestro caso, ese incremento de complejidad nos daría demasiados posibles puntos de inserción para las capas OWA, y complicaría mucho el análisis de los experimentos finales. Dentro de las arquitecturas VGG, pruebas preeliminares nos indican que la versión VGG13 ofrece resultados iguales a las más complejas (VGG16, la más empleada) para CIFAR, con tiempos de ejecución menores.

4.4 Parámetros

Para todos los experimentos hemos empleado los mismos hiperparámetros. En concreto, para CIFAR se ha definido como máximo ratio de aprendizaje para 1cycle $1e^{-2}$, un tamaño de lote de 1024, y se han realizado 30 épocas de entrenamiento por ejecución. Para decidir el ratio de aprendizaje óptimo, se empleó la herramienta *lr_finder* de Fastai sobre la red estándar sin introducir capas OWA, y después se comprobó en varias configuraciones que seguía siendo un ratio de aprendizaje adecuado para la red modificada con capas OWA. Afortunadamente, el empleo de la política 1cycle evita que el número de épocas o el ratio de aprendizaje influyan tanto en el aprendizaje como en un entrenamiento normal. El tamaño de lote se eligió ajustándonos a la memoria de la GPU disponible (en nuestro caso una GeForce RTX 2060).

En las pruebas con Imagenette, hemos empleado un ratio de aprendizaje de $6e^{-5}$ y un tamaño de lote de 64 imágenes, y se han realizado solo 10 épocas de entrenamiento (al ser este un dataset mucho más lento de entrenar).

Para el resto de parámetros se han mantenido los valores por defecto de Fastai, que incluyen una inicialización Kaiming [32]. No se han empleado capas dropout.

También hemos empleado *data augmentation* para regularización, siguiendo para CIFAR el patrón descrito en [33]. En concreto, hemos usado volteo lateral de la imagen con una probabilidad de 0,5 y *padding* de 4 píxeles (rellenados con una versión espejada de la imagen) seguido de un recorte aleatorio para recuperar la resolución de 32×32 píxeles.

Para Imagenette hemos empleado un patrón de data augmentation similar, en este caso con un padding de 32 píxeles sobre la imagen total de 256, y el mismo volteo horizonal con probabilidad de 0,5.

4.5 Métricas

Para evaluar las configuraciones, hemos realizado pruebas exhaustivas de diferentes configuraciones, tomando como referencia una copia de la red sin insertar ninguna capa OWA. Al ver que ninguna prueba ofrecía una mejora sustancial sobre la referencia, y que existía una gran varianza entre los resultados de diferentes ejecuciones, hemos optado por repetir cada experimento un cierto número de repeticiones (50 repeticiones para CIFAR y 10 repeticiones para Imagenette), y emplear análisis estadístico para comprobar si alguna ofrecía una mejora estadística respecto de la referencia. En concreto hemos optado por recoger los valores de precisión sobre la partición de validación de cada repetición tras la última repetición, y sobre ellos hemos obtenido:

- *Precisión media*, como una medida simple e intuitiva de la calidad de la configuración.
- *Desviación estándar de las precisiones*, para poner en perspectiva la precisión media, y observar si alguna de las configuraciones hace que los resultados tiendan más a extremos (mayor desviación estándar) o se centren más sobre la media (menor desviación estándar).
- *Test de Mann-Whitney U* [34], un test estadístico no paramétrico con el que compararemos las 50 precisiones observadas respecto de las 50 preci-

siones de referencia. Este test es elegido al no poder garantizar la normalidad de la distribución de resultados, y lo ejecutamos considerando como hipótesis nula que la precisión de la configuración de referencia es mejor o igual que la de la configuración estudiada. En general, consideraremos relevantes los resultados que arrojen en este test un p-valor $< 0{,}05$.

Además de la configuración de referencia estándar, se manejarán dos configuraciones de referencia adicionales para comprobar el impacto de la nueva arquitectura independientemente del uso de agregaciones OWA:

- *Referencia basada en no ordenación*: una configuración basada en la introducción de una capa OWA donde no se realice una ordenación de los canales de entrada. Esto nos permitirá comprobar si la simple combinación lineal añadida a través del operador OWA es responsable de cualquier mejora, o si realmente el hecho de que sea un operador basado en orden es crucial.

- *Referencia basada en ordenación aleatoria*: en este caso, ordenaríamos los canales de forma aleatoria. La idea sería acercar la propuesta hacia otros trabajos en los que se ha empleado el ruido como fuente de regularización para CNNs [35], para descartar que esa sea la fuente de la mejora observada en nuestros experimentos.

4.6 One Cycle

Uno de los detalles de implementación que más ha facilitado este trabajo es el empleo de la política de entrenamiento 1cycle [28].

La política 1cycle se basa en dividir el entrenamiento en dos fases, una durante la cual el ratio de aprendizaje va creciendo hasta alcanzar el máximo ratio programado, y otra durante la cual desciende de nuevo. La idea es emplear el ratio máximo de aprendizaje sólo en la zona intermedia del aprendizaje. Los límites desde los que se inicia el aprendizaje y hasta donde se reduce en la segunda fase son una décima parte del máximo ratio de aprendizaje. Además, se añade una pequeña fase extra en las últimas épocas de aprendizaje, donde se permite bajar el ratio de aprendizaje aún por debajo de ese límite. Podemos observar un ejemplo de evolución del ratio de aprendizaje en la Figura 4.1.

Al mismo tiempo que se hace esta gestión del ratio de aprendizaje, se realiza una gestión opuesta del momento, iniciando desde un valor más elevado, descendiendo a un mínimo, e incrementando después de nuevo.

La idea intuitiva es acompañar a la red en el patrón de aprendizaje habitual, haciendo al principio movimientos más lentos para iniciar el aprendizaje maximizar la fuerza del aprendizaje una vez se ha estabilizado, y al final reducir progresivamente conforme nos acercamos a una solución adecuada. En general, este tipo de gestión del ciclo de aprendizaje nos permite alcanzar resultados similares en una fracción del tiempo, y una mayor resistencia a las variaciones de los hiperparámetros.

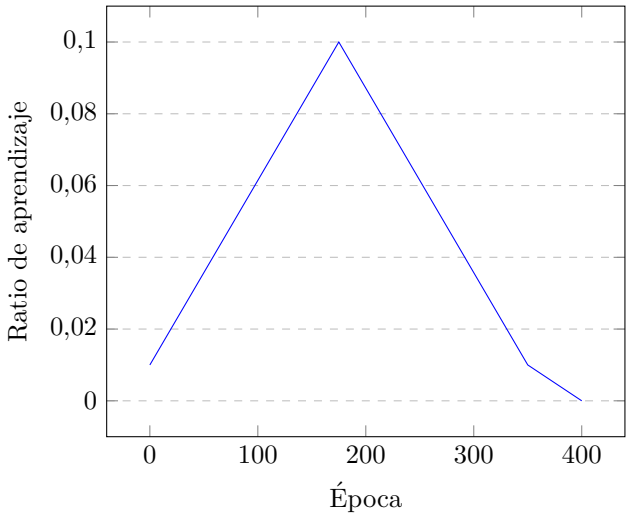


Figura 4.1: Evolución del ratio de aprendizaje en la política 1cycle.

4.7 Experimentos propuestos

En esta sección vamos a presentar los experimentos que hemos propuesto para validar nuestro modelo. Los resultados de estos experimentos se detallan en el Capítulo 5.

4.7.1 Configuraciones de punto de inserción

En el primer experimento buscamos comprobar la influencia del punto de inserción elegido (a qué profundidad de la red introducimos la capa OWA). Para esto, creamos una configuración fija con un número de características concreto ($C_f = 16$) y dos diferentes medidas de orden (suma de activaciones y variación total), y realizamos una prueba completa (50 repeticiones) con cada una de las posiciones posibles, desde OWA_1 hasta OWA_9.

La selección de esta configuración se basa en las pruebas preeliminares que se realizaron durante la implementación, donde por regla general los dos órdenes mencionados eran los que mejor funcionaban, y 16 características parecían suficientes. En los siguientes experimentos se variarán también estos parámetros para medir su influencia.

4.7.2 Configuraciones de número de características

En el segundo experimento tratamos de analizar la influencia del número de características aprendidas sobre la precisión final. En este caso, emplearemos las mejores configuraciones resultantes del primer experimento, y manteniendo el mismo orden y punto de inserción, variaremos la cantidad de características aprendidas. En concreto, exploraremos la inserción de $C_f = [4, 8, 16, 32]$ características.

La razón de elegir estos números es que en los experimentos preeliminares no hemos observado que la red tienda a aprender una gran variedad de carac-

terísticas nuevas, y por lo general parece replicar el mismo OWA en casi todas ellas. Por tanto, a priori no tiene sentido explorar un número demasiado elevado de características si no apreciamos una gran variedad en los resultados de este experimento.

4.7.3 Configuraciones de función de orden

En el tercer experimento realizaremos pruebas con diferentes medidas de orden. En concreto, emplearemos la mejor configuración del Experimento 2 tanto para CIFAR10 como para CIFAR100, sustituyendo la medida de orden que se haya empleado por todas las que hemos planteado en la Sección 3.2.

Dentro de este experimento, no solo comprobaremos si otras medidas de orden funcionan mejor que las planteadas, si no que también comprobaremos si las dos medidas de referencia (no ordenación y ordenación aleatoria) ofrecen mejora significativa respecto de la referencia base. Esto nos indicaría si otros factores no directamente asociados a la propuesta (incremento de la complejidad de la red, ruido introducido) tienen efecto en los resultados.

4.7.4 Complejidad computacional

Para medir la complejidad computacional, realizaremos un experimento concreto empleando diferentes números de características aprendidas ($C_f = [4, 8, 16, 32]$) para una de las ordenaciones propuestas (suma de activaciones), y con punto de inserción fijado en OWA_3, y después con diferentes ordenaciones con inserción en OWA_3 y número de características $C_f = 16$. Se realizarán estas pruebas para CIFAR10, sobre una única máquina, y se medirán los tiempos medios por época respecto del tiempo medio de la referencia. Este rango de configuraciones nos debería dar una idea del impacto computacional añadido por nuestro sistema.

4.7.5 OWA a nivel de píxel

Saliéndonos de los métodos principales de la propuesta, realizaremos también un experimento dedicado a probar la ordenación basada en píxel, en vez de por capa. Para este experimento, seleccionaremos un número de características aprendidas intermedio ($C_f = 32$), y probaremos a insertar capas de agregación OWA en la estructura en los diferentes puntos de inserción posibles, desde OWA_1 hasta OWA_9. Este experimento lo realizaremos solamente sobre CIFAR10.

4.7.6 Imagenette

Por último, y para cerrar el trabajo experimental, realizaremos una prueba general sobre un dataset con imágenes de mayor resolución, Imagenette. Realizaremos pruebas similares a los tres primeros experimentos con CIFAR, incluyendo:

- Referencia sin modificar.
- Inserción en las capas OWA_1 hasta OWA_9 con número de características aprendidas 16 y función de ordenación suma de activaciones.

- Sobre las mejores capas de la ejecución anterior, prueba con $C_f = [4, 8, 16, 32]$ y función de orden suma de activaciones.
- Sobre la mejor configuración de las ejecuciones anteriores, prueba con diferentes funciones de orden planteadas en 3.2.

Tabla 4.1: Arquitectura de la red*.

Nombre	Tamaño del Kernel	Paso	Tamaño de salida
entrada	-	-	$32 \times 32 \times 3$
conv1_1	3×3	1	$32 \times 32 \times 64$
OWA_1	-	-	$32 \times 32 \times (64 + C_f)$
conv1_2	3×3	1	$32 \times 32 \times 64$
maxpool	2×2	2	$16 \times 16 \times 64$
OWA_2	-	-	$16 \times 16 \times (64 + C_f)$
conv2_1	3×3	1	$16 \times 16 \times 128$
OWA_3	-	-	$16 \times 16 \times (128 + C_f)$
conv2_2	3×3	1	$16 \times 16 \times 128$
maxpool	2×2	2	$8 \times 8 \times 128$
OWA_4	-	-	$8 \times 8 \times (128 + C_f)$
conv3_1	3×3	1	$8 \times 8 \times 256$
OWA_5	-	-	$8 \times 8 \times (256 + C_f)$
conv3_2	3×3	1	$8 \times 8 \times 256$
maxpool	2×2	2	$4 \times 4 \times 256$
OWA_6	-	-	$4 \times 4 \times (256 + C_f)$
conv4_1	3×3	1	$4 \times 4 \times 512$
OWA_7	-	-	$4 \times 4 \times (512 + C_f)$
conv4_2	3×3	1	$4 \times 4 \times 512$
maxpool	2×2	2	$2 \times 2 \times 512$
OWA_8	-	-	$2 \times 2 \times (512 + C_f)$
conv5_1	3×3	1	$2 \times 2 \times 512$
OWA_9	-	-	$2 \times 2 \times (512 + C_f)$
conv5_2	3×3	1	$2 \times 2 \times 512$
maxpool	2×2	2	$1 \times 1 \times 512$
flatten	-	-	512
linear	-	-	10/100

* Las capas señaladas con OWA_x son los posibles puntos de inserción para las capas OWA.

Capítulo 5

Estudio experimental

5.1 Resultados

5.1.1 Configuraciones de punto de inserción

En el primer experimento, cuyos resultados se recogen en la Tabla 5.1, podemos comprobar una clara correlación entre los diferentes posibles puntos de inserción y la precisión final alcanzada por el sistema. De forma gráfica, se puede observar esta tendencia en la Figura 5.1, donde se observa tanto para CIFAR10 como para CIFAR100 un incremento bastante sólido en los puntos de inserción más bajos (OWA_2 a OWA_5) respecto de la referencia, especialmente para la ordenación por suma de activaciones.

Estadísticamente hablando, todos los puntos de inserción entre OWA_2 y OWA_5 con suma de activaciones mejoran la media con p-valor $< 0{,}05$. Específicamente, destacan las configuraciones en OWA_4 para CIFAR10, que alcanza un 92,55 % de precisión (referencia en 92,44 %), y OWA_5 para CIFAR100 con un 69,97 % (referencia en 69,74 %). Aunque estos resultados no supongan una mejora crítica, como hemos mencionado, son ambos estadísticamente significativos.

En este caso, creemos que el origen de la variación en la mejora puede deberse principalmente al tamaño de imagen. Como se vio en la Tabla 4.1, los puntos de inserción más altos, de OWA_6 en adelante, se corresponden con zonas de la arquitectura donde la imagen llega a tener un tamaño muy reducido, de 4×4 y 2×2 píxeles de resolución. Creemos que en estos tamaños, las medidas de ordenación de canales pueden no tener tanto sentido, ni aportar tanta información a la red.

5.1.2 Configuraciones de número de características

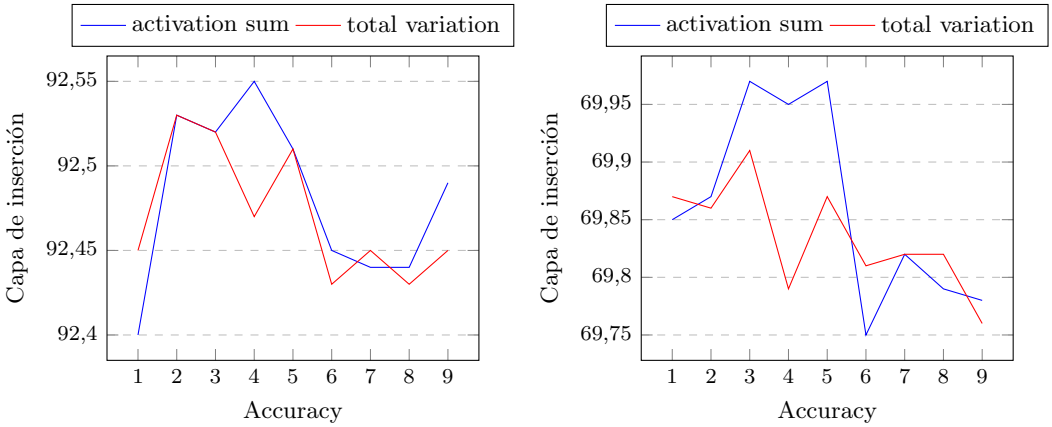
Para este segundo experimento hemos seleccionado, a partir de los resultados del primero, las configuraciones con suma de activaciones y puntos de inserción OWA_3 y OWA_4, al dar buenos resultados para ambos datasets.

Los resultados quedan recogidos en la Tabla 5.2. En este experimento, donde lo que variamos es la cantidad de características aprendidas (de nuevos canales generados a partir de OWAs), podemos observar como los números intermedios, 8 y 16 características, obtienen los mejores resultados (en todos los casos significativamente mejores que la referencia, con p-valor $< 0{,}05$). En 3 de las

Tabla 5.1: Resultados de las configuraciones por capas.

Orden	Capa	Precisión CIFAR10	Precisión CIFAR100
reference	-	$92{,}44 \pm 0{,}17$	$69{,}74 \pm 0{,}27$
activ_sum	OWA_1	$92{,}40 \pm 0{,}19$	$69{,}85 \pm 0{,}29^{\bullet}$
	OWA_2	$92{,}53 \pm 0{,}19^{\bullet}$	$69{,}87 \pm 0{,}32^{\bullet}$
	OWA_3	$92{,}52 \pm 0{,}18^{\bullet}$	$69{,}97 \pm 0{,}27^{\bullet}$
	OWA_4	$\mathbf{92{,}55 \pm 0{,}18}^{\bullet}$	$69{,}95 \pm 0{,}24^{\bullet}$
	OWA_5	$92{,}51 \pm 0{,}17^{\bullet}$	$\mathbf{69{,}97 \pm 0{,}28}^{\bullet}$
	OWA_6	$92{,}45 \pm 0{,}20$	$69{,}75 \pm 0{,}33$
	OWA_7	$92{,}44 \pm 0{,}17$	$69{,}82 \pm 0{,}25$
	OWA_8	$92{,}44 \pm 0{,}18$	$69{,}79 \pm 0{,}30$
	OWA_9	$92{,}49 \pm 0{,}20$	$69{,}78 \pm 0{,}25$
total_var	OWA_1	$92{,}45 \pm 0{,}18$	$69{,}87 \pm 0{,}26^{\bullet}$
	OWA_2	$92{,}53 \pm 0{,}15^{\bullet}$	$69{,}86 \pm 0{,}31^{\bullet}$
	OWA_3	$92{,}52 \pm 0{,}18^{\bullet}$	$69{,}91 \pm 0{,}24^{\bullet}$
	OWA_4	$92{,}47 \pm 0{,}17$	$69{,}79 \pm 0{,}23$
	OWA_5	$92{,}51 \pm 0{,}19^{\bullet}$	$69{,}87 \pm 0{,}31^{\bullet}$
	OWA_6	$92{,}43 \pm 0{,}20$	$69{,}81 \pm 0{,}27$
	OWA_7	$92{,}45 \pm 0{,}18$	$69{,}82 \pm 0{,}28$
	OWA_8	$92{,}43 \pm 0{,}18$	$69{,}82 \pm 0{,}29$
	OWA_9	$92{,}45 \pm 0{,}19$	$69{,}76 \pm 0{,}26$

Los resultados marcados con $^{\bullet}$ mejoran la precisión de la referencia con p-valor $<$ 0,05.



(a) CIFAR10 (b) CIFAR100

Figura 5.1: Precisión en función de la capa de inserción.

4 variaciones (todas excepto OWA_3 en CIFAR10), vemos reducirse la precisión final al llegar a 32 características, y en todas ellas 4 características obtienen peores resultados que el resto.

Podemos observar que, en este caso, 16 características parecen suficientes, y 8 características tienden a obtener resultados similares.

Tabla 5.2: Resultados de las configuraciones por número de características.

Capa	Características	Precisión CIFAR10	Precisión CIFAR100
reference	-	$92{,}44 \pm 0{,}17$	$69{,}74 \pm 0{,}27$
OWA_3	4	$92{,}49 \pm 0{,}19$	$69{,}82 \pm 0{,}32$
	8	$92{,}51 \pm 0{,}21^{\bullet}$	$69{,}88 \pm 0{,}26^{\bullet}$
	16	$92{,}52 \pm 0{,}18^{\bullet}$	$\mathbf{69{,}97 \pm 0{,}27^{\bullet}}$
	32	$\mathbf{92{,}57 \pm 0{,}16^{\bullet}}$	$69{,}82 \pm 0{,}28$
OWA_4	4	$92{,}47 \pm 0{,}17$	$69{,}81 \pm 0{,}33$
	8	$92{,}50 \pm 0{,}19^{\bullet}$	$69{,}91 \pm 0{,}31^{\bullet}$
	16	$92{,}55 \pm 0{,}18^{\bullet}$	$69{,}95 \pm 0{,}24^{\bullet}$
	32	$92{,}51 \pm 0{,}17^{\bullet}$	$69{,}90 \pm 0{,}29^{\bullet}$

Los resultados marcados con $^{\bullet}$ mejoran la precisión de la referencia con p-valor $<$ 0,05.

5.1.3 Configuraciones de función de orden

Para este experimento empleamos las dos configuraciones con mejor resultado para CIFAR10 y CIFAR100 independientemente: para CIFAR10 utilizamos punto de inserción en OWA_3, número de características 32, y para CIFAR100 punto de inserción en OWA_2 y número de características 16, y variamos la función de ordenación.

En los resultados, que se recogen en la Tabla 5.3, podemos confirmar que la suma de activaciones es la ordenación que obtiene mejor resultado para nuestra combinación de arquitectura y dataset. Aun así, vemos también buenos resultados para la variación total y para la activación máxima. En concreto, vemos que tanto suma de activaciones como la variación total mejoran significativamente la referencia (con p-valor $<$ 0,05), y la activación máxima lo consigue para CIFAR10.

5.2 Complejidad computacional

Para comprobar el impacto computacional de nuestra capa OWA, probamos diferentes configuraciones de la red en una misma máquina equipada con una tarjeta gráfica GeForce RTX 2060 GPU, 20GB de RAM, un procesador Intel Core i5-8500 CPU, y con Ubuntu 18.04 como sistema operativo. En la Tabla 5.4 mostramos los tiempos medios por época de la ejecución de las diferentes configuraciones para CIFAR10 (al tener el mismo número de imágenes que CIFAR100, los tiempos son los mismos para ambos datasets). El punto de inserción se deja fijado en OWA_3 para todas las configuraciones.

Tabla 5.3: Resultados de las configuraciones por funciones de ordenación.

Orden	Precisión CIFAR10	Precisión CIFAR100
reference	$92{,}44 \pm 0{,}17$	$69{,}74 \pm 0{,}27$
activ_sum	$\mathbf{92{,}57 \pm 0{,}16}^{\bullet}$	$\mathbf{69{,}97 \pm 0{,}27}^{\bullet}$
total_var	$92{,}55 \pm 0{,}19^{\bullet}$	$69{,}91 \pm 0{,}24^{\bullet}$
max_activ	$92{,}51 \pm 0{,}21^{\bullet}$	$69{,}74 \pm 0{,}28$
median_activ	$92{,}48 \pm 0{,}19$	$69{,}84 \pm 0{,}31$
entropy	$92{,}47 \pm 0{,}16$	$69{,}80 \pm 0{,}25$
random	$92{,}45 \pm 0{,}17$	$69{,}76 \pm 0{,}26$
no_sorting	$92{,}43 \pm 0{,}19$	$69{,}79 \pm 0{,}30$

Los resultados marcados con $^{\bullet}$ mejoran la precisión de la referencia con p-valor $<$ 0,05.

En estos resultados observamos un incremento de aproximadamente 10 % en los tiempos de ejecución respecto de la referencia para todas las configuraciones, sin haber una variación de tiempo considerable entre las diferentes configuraciones. El hecho de que esta variación sea constante, y no dependa ni del número de características aprendidas ni de la función de ordenación, indica que se trata probablemente de la implementación que hemos desarrollado y que podría ser optimizada.

Tabla 5.4: Tiempos de ejecución CIFAR10.

Orden	Características	Tiempo por época (s)
reference	-	9,67
activation_sum	4	10,70
activation_sum	8	10,77
activation_sum	16	10,77
activation_sum	32	10,73
activation_sum	16	10,77
total_variation	16	11,00
max_activation	16	10,86
median_activation	16	11,33
entropy	16	10,93
random	16	11,10
no_sorting	16	11,27

5.3 Matrices de pesos

Despueś de estos experimentos, resulta interesante detenernos a observar qué tipo de OWAs ha aprendido la red durante su entrenamiento. En la Figura 5.2 hemos recogido algunas de las matrices de pesos correspondientes a las capas de agregación del tercer experimento en el dataset de CIFAR10.

Las matrices, tal y como se muestran, se corresponden con los pesos ya procesados de la capa. Cada una de las filas de las imágenes se corresponden con una de las nuevas características aprendidas (8 en este caso), y cada una de las columnas corresponde con uno de los canales de entrada en este punto de inserción (64), ya ordenadas según la métrica indicada (de mayor a menor en estas gráficas).

Podemos observar, primero, una clara convergencia en las ordenaciones basadas en métricas respecto de las ordenaciones de referencia (no ordenación y ordenación aleatoria). Esto nos indica que, de alguna manera, la red está considerando estas ordenaciones y realizando agregación "lógicas", en concreto mínimos y máximos suavizados, sobre todo.

Como podemos observar en el caso de la variación total la red tiende a aprender pesos correspondientes a mínimos suavizados, siendo estos una agregación de los canales con menor variación total de la red en esa capa. En el caso de la suma de activaciones, se observa tanto mínimos suavizados como un máximo suavizado, indicándonos que la red está siendo capaz de extraer información tanto de la agregación de los canales con menor métrica como de los de mayor.

Resulta también interesante observar como en las agregaciones correspondientes a la mediana, las agregaciones de mínimos son más difusas, agregando con un valor de ponderación similar bastantes canales, mientras que se aprende también una agregación de máximo mucho más "crisp" o radical, valorando muy por encima la capa de máximo que a las siguientes en orden.

5.4 OWA a nivel de píxel

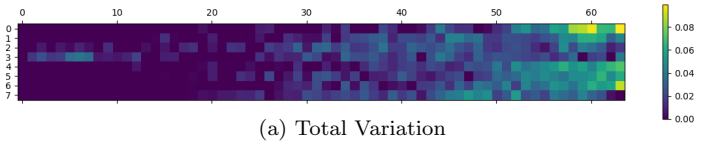
En la Tabla 5.5 podemos observar los resultados del cuarto experimento, donde hemos probado en CIFAR10 diferentes configuraciones de agregación OWA por píxel.

En estos resultados, vemos al menos 3 configuraciones que parecen mejorar la referencia de forma estadísticamente significativa. Sin embargo, sus precisiones quedan bastante por debajo de las obtenidas en agregación por capas, siendo la mejor un 92,51 % en la capa de inserción en OWA_5. Además, no se observa ningún patrón particular en estos resultados, por lo que bien podrían ser ruido estadístico.

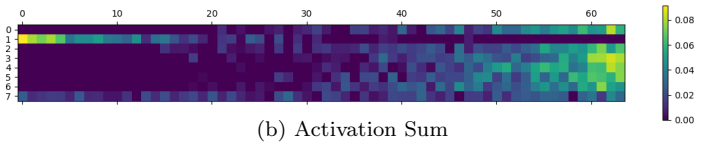
5.5 Imagenette

Finalmente, los resultados sobre Imagenette se detallan en las Tablas 5.6 y 5.7. Si bien estos resultados han sido obtenidos con solo 10 repeticiones de cada experimento, podemos apreciar tendencias bastante claras en ellos.

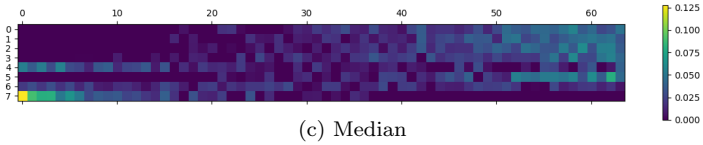
En la Tabla 5.6 podemos ver los resultados de aplicar nuestra capa OWA con $C_f = 32$ características en diferentes puntos de la red. En general, observamos una mejora más leve en las capas con suma de activaciones, pero con mayor independencia de la capa de inserción. En el caso de la variación total, hay una tendencia fuerte a mejorar solo en las capas más altas de la red (OWA_6 a OWA_9), pero la mejora parece más grande que en la suma de activaciones. En prácticamente todos los resultados con suma de activaciones, y en todos



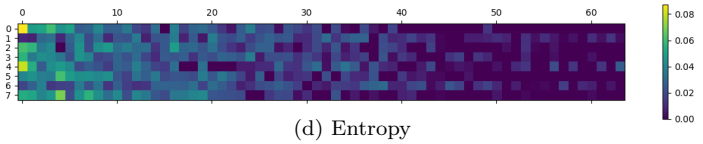
(a) Total Variation



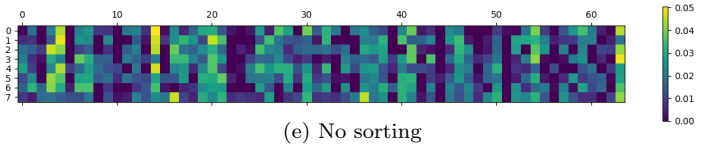
(b) Activation Sum



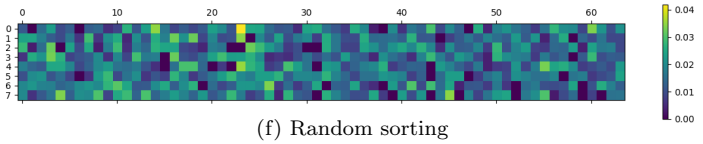
(c) Median



(d) Entropy



(e) No sorting



(f) Random sorting

Figura 5.2: Matrices de pesos de OWA para diferentes funciones de orden.

Tabla 5.5: Resultados de las configuraciones para OWA a nivel de píxel.

Capa	Precisión CIFAR10
reference	$92{,}44 \pm 0{,}17$
OWA_1	$92{,}46 \pm 0{,}16$
OWA_2	$92{,}49 \pm 0{,}14^{\bullet}$
OWA_3	$92{,}47 \pm 0{,}17$
OWA_4	$92{,}44 \pm 0{,}16$
OWA_5	$\mathbf{92{,}51 \pm 0{,}18}^{\bullet}$
OWA_6	$92{,}44 \pm 0{,}18$
OWA_7	$92{,}40 \pm 0{,}18$
OWA_8	$92{,}49 \pm 0{,}20^{\bullet}$
OWA_9	$92{,}46 \pm 0{,}16$

Los resultados marcados con $^{\bullet}$ mejoran la precisión de la referencia con p-valor $<$ 0,05.

los resultados entre OWA_6 y OWA_9 con variación total podemos apreciar un incremento estadísticamente significativo, con p-valor $< 0{,}05$.

Tabla 5.6: Resultados de las configuraciones de puntos de inserción para Imagenette.

Capa	Precisión Imagenette total_var	Precisión Imagenette activ_sum
Reference	$89{,}48 \pm 0{,}54$	$89{,}48 \pm 0{,}85$
OWA_1	$89{,}42 \pm 0{,}74$	$89{,}60 \pm 0{,}62$
OWA_2	$89{,}92 \pm 0{,}63$	$90{,}02 \pm 0{,}72^{\bullet}$
OWA_3	$89{,}90 \pm 0{,}57$	$90{,}00 \pm 0{,}56^{\bullet}$
OWA_4	$89{,}76 \pm 0{,}46$	$90{,}14 \pm 0{,}42^{\bullet}$
OWA_5	$89{,}78 \pm 0{,}47$	$90{,}06 \pm 0{,}55^{\bullet}$
OWA_6	$90{,}48 \pm 0{,}61^{\bullet}$	$90{,}28 \pm 0{,}88^{\bullet}$
OWA_7	$\mathbf{90{,}78 \pm 0{,}37}^{\bullet}$	$89{,}94 \pm 0{,}47^{\bullet}$
OWA_8	$90{,}16 \pm 0{,}44^{\bullet}$	$\mathbf{90{,}56 \pm 0{,}76}^{\bullet}$
OWA_9	$90{,}38 \pm 0{,}46^{\bullet}$	$90{,}48 \pm 0{,}69^{\bullet}$

Los resultados marcados con $^{\bullet}$ mejoran la precisión de la referencia con p-valor $<$ 0,05.

En la Tabla 5.7 realizamos el segundo experimento, en este caso sobre el número de características aprendidas, fijando las capas entre OWA_6 y OWA_9 y el número de características en $C_f = [4, 8, 16, 32, 64]$. Para variación total, vemos resultados estadísticamente significativos (p-valor $< 0{,}05$) en casi todas las configuraciones. Se aprecia una tendencia a tener mejores resultados con mayor número de características aprendidas, con todos los resultados para $C_f = 64$ por encima de 90,50 %, y casi todos los resultados con menos características

aprendidas por debajo. En el caso de la suma de activaciones, de nuevo observamos resultados estadísticamente significativos (p-valor $< 0{,}05$) en casi todas las configuraciones.

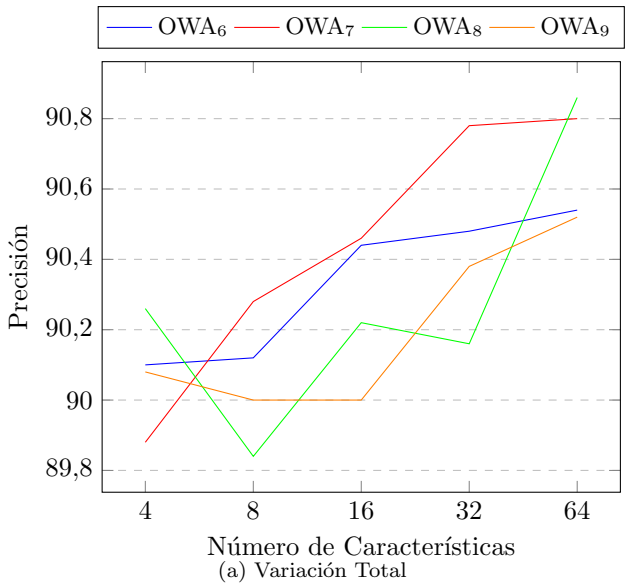
En la Figura 5.3 hemos reflejado para cada capa, la evolución de la precisión respecto al número de características totales, donde podemos apreciar la tendencia general a mejorar los resultados con mayor número de características aprendidas, especialmente en el caso de la variación total como métrica de ordenación.

Tabla 5.7: Resultados de las configuraciones de número de características aprendidas para Imagenette.

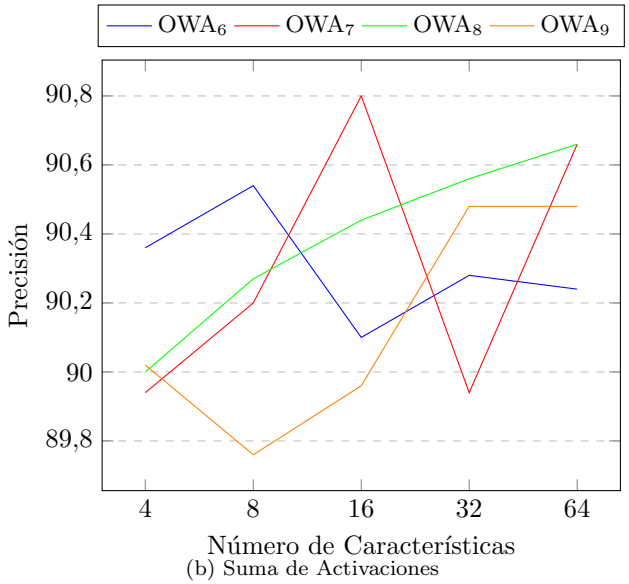
Características	Capa	Precisión Imagenette total_var	Precisión Imagenette activ_sum
reference	-	$89{,}48 \pm 0{,}54$	$89{,}48 \pm 0{,}85$
64	OWA_6	$90{,}54 \pm 0{,}54$ $^\bullet$	$90{,}24 \pm 0{,}81^\bullet$
	OWA_7	$90{,}80 \pm 0{,}61$ $^\bullet$	$90{,}66 \pm 0{,}30^\bullet$
	OWA_8	$\mathbf{90{,}86 \pm 0{,}51}$ $^\bullet$	$90{,}66 \pm 0{,}61^\bullet$
	OWA_9	$90{,}52 \pm 0{,}58$ $^\bullet$	$90{,}48 \pm 0{,}43^\bullet$
32	OWA_6	$90{,}48 \pm 0{,}61$ $^\bullet$	$90{,}28 \pm 0{,}88^\bullet$
	OWA_7	$90{,}78 \pm 0{,}37$ $^\bullet$	$89{,}94 \pm 0{,}47^\bullet$
	OWA_8	$90{,}16 \pm 0{,}44$ $^\bullet$	$90{,}56 \pm 0{,}76^\bullet$
	OWA_9	$90{,}38 \pm 0{,}46$ $^\bullet$	$90{,}48 \pm 0{,}69^\bullet$
16	OWA_6	$90{,}44 \pm 0{,}63$ $^\bullet$	$90{,}10 \pm 0{,}41^\bullet$
	OWA_7	$90{,}46 \pm 0{,}63$ $^\bullet$	$\mathbf{90{,}80 \pm 0{,}60}^\bullet$
	OWA_8	$90{,}22 \pm 0{,}71$ $^\bullet$	$90{,}44 \pm 0{,}74^\bullet$
	OWA_9	$90{,}00 \pm 0{,}50$ $^\bullet$	$89{,}96 \pm 0{,}43^\bullet$
8	OWA_6	$90{,}12 \pm 0{,}61$ $^\bullet$	$90{,}54 \pm 0{,}62^\bullet$
	OWA_7	$90{,}28 \pm 0{,}66$ $^\bullet$	$90{,}20 \pm 0{,}56^\bullet$
	OWA_8	$89{,}84 \pm 0{,}46$	$90{,}27 \pm 0{,}55^\bullet$
	OWA_9	$90{,}00 \pm 0{,}49$ $^\bullet$	$89{,}76 \pm 0{,}52$
4	OWA_6	$90{,}10 \pm 0{,}96$ $^\bullet$	$90{,}36 \pm 0{,}49^\bullet$
	OWA_7	$89{,}88 \pm 0{,}59$	$89{,}94 \pm 0{,}55^\bullet$
	OWA_8	$90{,}26 \pm 0{,}58$ $^\bullet$	$90{,}00 \pm 0{,}45^\bullet$
	OWA_9	$90{,}08 \pm 0{,}63$ $^\bullet$	$90{,}02 \pm 0{,}53^\bullet$

Los resultados marcados con $^\bullet$ mejoran la precisión de la referencia con p-valor $< 0{,}05$.

Por último, en la Tabla 5.8 podemos ver los resultados de mantener una configuración en el punto de inserción OWA_8 con número de características aprendidas $C_f = 64$ y diferentes métricas de ordenación. En concreto, observamos los mejores resultados con las métricas ya empleadas (suma de activaciones y variación total), junto con la mediana, que en este caso a ha demostrado también una considerable mejora respecto de la referencia. Estas tres medidas, junto con la activación máxima, consiguen todas resultados estadísticamente significativos (p-valor $< 0{,}05$). La entropía, la propuesta original de [14], en este experimento



(a) Variación Total



(b) Suma de Activaciones

Figura 5.3: Precisión en función del número de características en Imagenette.

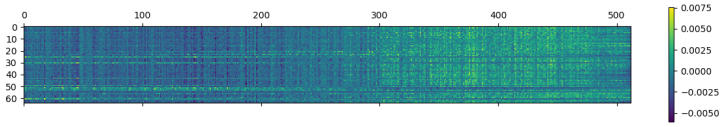
no ofrece mejora significativa, al igual que las ordenaciones de referencia (ordenación aleatoria y no ordenación). El comportamiento de las ordenaciones de referencia confirma que nuestros resultados se deben principalmente al hecho de emplear operadores OWA, basados en orden, y no al incremento del número de parámetros de la red.

En la Figura 5.4 hemos recogido las matrices de pesos para Imagenette de las tres mejores funciones de orden, variación total, suma de activaciones y mediana. En este caso, se muestras las matrices sin procesar para facilitar la visualización de los patrones que se forman. Podemos observar en los tres casos la tendencia a los mínimos suavizados, pero curiosamente se aprecia como en la variación total y suma de activaciones estos mínimos se apartan del mínimo absoluto. En el caso de la variación total, por ejemplo, vemos que el operador OWA se ha centrado principalmente en torno a la capa 400 (una vez ordenadas por su métrica).

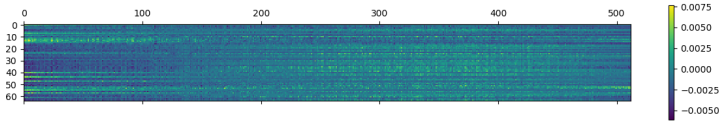
Tabla 5.8: Resultados de las configuraciones por funciones de ordenación para Imagenette.

Orden	Precisión Imagenette
reference	$89{,}48 \pm 0{,}54$
total_var	$\mathbf{90{,}86 \pm 0{,}51}^{\bullet}$
median_activ	$90{,}69 \pm 0{,}57^{\bullet}$
activ_sum	$90{,}66 \pm 0{,}61^{\bullet}$
max_activ	$89{,}84 \pm 0{,}32^{\bullet}$
entropy	$89{,}72 \pm 0{,}56$
random	$89{,}80 \pm 0{,}69$
no_sorting	$89{,}34 \pm 0{,}55$

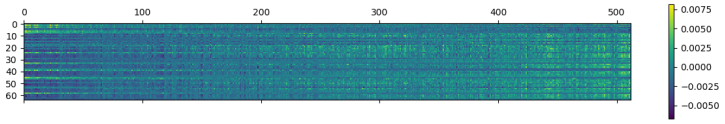
Los resultados marcados con $^{\bullet}$ mejoran la precisión de la referencia con p-valor $<$ 0,05.



(a) Total Variation



(b) Activation Sum



(c) Median

Figura 5.4: Matrices de pesos de OWA para diferentes funciones de orden sobre Imagenette.

Capítulo 6

Conclusiones y líneas futuras

6.1 Conclusiones

Tras los experimentos planteados, podemos confirmar que la propuesta de capa OWA es capaz de mejorar una arquitectura CNN habitual, mejorando su precisión sin un coste computacional sustancial. Hemos podido comprobar que la red parece manejar correctamente la idea de OWA, convergiendo los pesos de los operadores OWA planteados a operadores OWA conocidos (mínimos y máximos suavizados, principalmente). Hemos probado los resultados en diferentes datasets, comprobando el impacto de éstos en la mejora que ofrece el nuevo método, y comprobando como la mejora ha sido mayor al pasar de datasets de prueba como CIFAR a datasets más complejos y realistas como Imagenette.

Si bien no hemos podido localizar una forma de optimizar los parámetros de estas nuevas capas, o de hacer más resistente a la arquitectura respecto de estas variaciones de parámetros, esta primera aproximación confirma que explotar esta idea puede llevar a mejoras interesantes en las CNNs y otras redes neuronales profundas.

6.2 Líneas futuras

Sobre las líneas planteadas en este estudio, vemos al menos tres líneas futuras a explorar.

Primero, la optimización de la arquitectura planteada en este artículo, investigando maneras de calcular los puntos de inserción óptimos y el número de características a emplear, de forma que optimicemos la ganancia del sistema. Los resultados apenas alcanzan un uno por ciento de mejora, pero creemos que con la configuración de parámetros adecuada y otros ajustes, esta mejora podría superarse. Dentro de este trabajo tampoco se han podido explorar todas las posibilidades, en particular los sistemas de inicialización de la matriz de pesos, que podrían mejorar aún los resultados expuestos.

En segundo lugar, el empleo de sistemas similares en otros datasets, problemas y aplicaciones. En nuestro caso hemos aplicado la idea en un par de datasets

bastante pequeños para poder ampliar la cantidad de experimentos realizados, y nos hemos centrado en problemas de clasificación simple de imágenes. Esta misma idea, sin embargo, se puede ampliar de forma directa a otras estructuras (combinarla con ResNet o DenseNet, por ejemplo), problemas (detección de patrones, escalado de imágenes, coloreado automático...) y datasets (ImageNet y otros datasets con tamaño de imagen y complejidad superior).

Por último, creemos que el concepto general de "aumentación de mapas de características" es un terreno interesante que explorar. Al igual que los operadores OWA, existen múltiples agregaciones y operaciones con las que se pueden generar mapas de características difíciles de obtener para una red convolucional, pero que aporten información a esta. Sin embargo, pocas de estas operaciones podrán sustituir directamente a una operación convolucional sin un impacto negativo en los resultados. Es por esto que creemos que el sistema propuesto puede ser una buena forma de añadir potencia de cálculo a las redes neuronales convolucionales sin realizar grandes sacrificios en el proceso.

Bibliografía

[1] D. Wlodzislaw, "What is Computational Intelligence and what could it become?", ene. de 2003.

[2] L. Zadeh, "Fuzzy sets", en, *Information and Control*, vol. 8, n.º 3, págs. 338-353, jun. de 1965.

[3] I. Goodfellow, Y. Bengio y A. Courville, *Deep Learning*. MIT Press, 2016.

[4] G. Cybenko, "Approximation by superpositions of a sigmoidal function", en, *Math. Control Signal Systems*, vol. 2, n.º 4, págs. 303-314, dic. de 1989.

[5] R. R. Yager, "On ordered weighted averaging aggregation operators in multicriteria decisionmaking", *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 18, n.º 1, págs. 183-190, 1988.

[6] G. J. Scott, R. A. Marcum, C. H. Davis y T. W. Nivin, "Fusion of Deep Convolutional Neural Networks for Land Cover Classification of High-Resolution Imagery", *IEEE Geoscience and Remote Sensing Letters*, vol. 14, n.º 9, págs. 1638-1642, 2017.

[7] G. J. Scott, K. C. Hagan, R. A. Marcum, J. A. Hurt, D. T. Anderson y C. H. Davis, "Enhanced Fusion of Deep Neural Networks for Classification of Benchmark High-Resolution Image Data Sets", *IEEE Geoscience and Remote Sensing Letters*, vol. 15, n.º 9, págs. 1451-1455, 2018.

[8] D. T. Anderson, G. J. Scott, M. Islam, B. Murray y a. R. Marcum, "Fuzzy choquet integration of deep convolutional neural networks for remote sensing", en *Computational Intelligence for Pattern Recognition*, Springer, 2018, págs. 1-28.

[9] X. Du y A. Zare, "Multiple Instance Choquet Integral Classifier Fusion and Regression for Remote Sensing Applications", *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, n.º 5, págs. 2741-2753, 2019.

[10] C. A. Dias, J. C. S. Bueno, E. N. Borges, S. S. C. Botelho, G. P. Dimuro, G. Lucca, J. Fernandéz, H. Bustince y P. L. J. Drews Junior, "Using the Choquet Integral in the Pooling Layer in Deep Learning Networks", en *Fuzzy Information Processing*, G. A. Barreto y R. Coelho, eds., Springer, 2018, págs. 144-154.

[11] C. A. Dias, J. C. S. Bueno, E. N. Borges, G. Lucca, H. Santos, G. P. Dimuro, H. Bustince, P. L. J. D. Junior, S. S. C. Botelho y E. Palmeira, "Simulating the Behaviour of Choquet-Like (pre) Aggregation Functions for Image Resizing in the Pooling Layer of Deep Learning Networks", en *International Fuzzy Systems Association World Congress*, Springer, 2019, págs. 224-236.

[12] J. M. Keller, D. Liu y D. B. Fogel, *Fundamentals of Computational Intelligence: Neural Networks, Fuzzy Systems, and Evolutionary Computation.* Wiley, 2016.

[13] C. Veal, A. Yang, A. Hurt, M. A. Islam, D. T. Anderson, G. Scott, J. M. Keller, T. C. Havens y B. Tang, "Linear Order Statistic Neuron", en *2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 2019, págs. 1-6.

[14] S. R. Price, S. R. Price y D. T. Anderson, "Introducing Fuzzy Layers for Deep Learning", en *2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 2019, págs. 1-6.

[15] E. Shelhamer, J. Long y T. Darrell, "Fully Convolutional Networks for Semantic Segmentation", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, n.º 4, págs. 640-651, 2017.

[16] W. S. McCulloch y W. Pitts, "A logical calculus of the ideas immanent in nervous activity", en, *Bulletin of Mathematical Biophysics*, vol. 5, n.º 4, págs. 115-133, dic. de 1943.

[17] H. Lee, R. Grosse, R. Ranganath y A. Y. Ng, "Unsupervised learning of hierarchical representations with convolutional deep belief networks", *Commun. ACM*, vol. 54, n.º 10, págs. 95-103, oct. de 2011.

[18] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever y R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", *Journal of Machine Learning Research*, vol. 15, págs. 1929-1958, 2014.

[19] D. P. Kingma y J. Ba, "Adam: A Method for Stochastic Optimization", *arXiv:1412.6980 [cs]*, ene. de 2017.

[20] J. Duchi, E. Hazan e Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization", *Journal of Machine Learning Research*, vol. 12, n.º Jul, págs. 2121-2159, 2011.

[21] S. Ioffe y C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", en, en *International Conference on Machine Learning*, jun. de 2015, págs. 448-456.

[22] S. Santurkar, D. Tsipras, A. Ilyas y A. Madry, "How Does Batch Normalization Help Optimization?", *arXiv:1805.11604 [cs, stat]*, abr. de 2019.

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

[24] K. Simonyan y A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition", *arXiv 1409.1556*, 2014.

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

[26] G. Huang, Z. Liu, L. v. d. Maaten y K. Q. Weinberger, "Densely Connected Convolutional Networks", en *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, págs. 2261-2269.

[27] L. I. Rudin, S. Osher y E. Fatemi, "Nonlinear total variation based noise removal algorithms", *Physica D: Nonlinear Phenomena*, vol. 60, n.º 1, págs. 259-268, 1992.

[28] L. N. Smith, "A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay", *arXiv 1803.09820*, 2018.

[29] A. Krizhevsky, "Learning multiple layers of features from tiny images.", *Master's thesis, Department ofComputer Science, University of Toronto*, 2009.

[30] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li y L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database", en *CVPR09*, 2009.

[31] S. Liu y W. Deng, "Very deep convolutional neural network based image classification using small training sample size", en *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, 2015, págs. 730-734.

[32] K. He, X. Zhang, S. Ren y J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification", en *2015 IEEE International Conference on Computer Vision (ICCV)*, Santiago, Chile: IEEE, dic. de 2015, págs. 1026-1034.

[33] K. He, X. Zhang, S. Ren y J. Sun, "Identity Mappings in Deep Residual Networks", *arXiv 1603.05027*, 2016.

[34] H. B. Mann y D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other", *The annals of mathematical statistics*, págs. 50-60, 1947.

[35] C. M. Bishop, "Training with Noise is Equivalent to Tikhonov Regularization", en, *Neural Computation*, vol. 7, n.º 1, págs. 108-116, ene. de 1995.