UNIVERSIDAD PÚBLICA DE NAVARRA

TRABAJO DE FIN DE MASTER

Efecto de la sustitución de la función de Pooling en Redes Neuronales Convolucionales

---

Autor:
Iosu RODRÍGUEZ

Supervisor:
Nicanor Humberto BUSTINCE

Máster Universitario en Ingeniería Informática

Escuela Técnica Superior de Ingeniería Industrial, Informática y Telecomunicación
Departamento de Estadística, Informática y Matemáticas

June 1, 2020

UNIVERSIDAD PÚBLICA DE NAVARRA

Abstract

Escuela Técnica Superior de Ingeniería Industrial, Informática y Telecomunicación
Departamento de Estadística, Informática y Matemáticas

Máster Universitario en Ingeniería Informática

Efecto de la sustitución de la función de Pooling en Redes Neuronales Convolucionales

by Iosu RODRÍGUEZ

Las redes neuronales convolucionales llevan marcando, en los últimos años, el estado del arte absoluto en tareas de clasificación y segmentación de imagen. Sin embargo, y a pesar de que en este tiempo se han introducido nuevas operaciones y arquitecturas que buscan mejorar su rendimiento, las funciones básicas en base a las que todos estos modelos operan, esto es, la aplicación de fitros de convolución para extraer características y la reducción de imagen en el llamado proceso de "pooling", han visto pocos intentos de mejora.

En este trabajo, se ha buscado sustituir las operaciones clásicas de pooling por funciones más avanzadas, como pueden ser la integral de sugeno o la combinación de funciones crecientes. Además, también se propone la utilización de un proceso de toma de decisión multimodal, capaz de combinar la información aprendida por distintas variantes de estos modelos.

Agradecimientos

En primer lugar, me gustaría agradecer a la Universidad Pública de Navarra por la enseñanza de calidad de la que me ha dotado, y que me ha permitido crecer como profesional y como persona. En concreto, al departamento de Estadística, Informática y Matemáticas, por abrirme sus puertas y permitirme, desde el primer momento, formar parte de proyectos tremendamente interesantes que me han ayudado a dar el salto del aula a un entorno de trabajo, en el que me he sentido como en casa desde el primer día.

A este respecto, quisiera agradecer a todos mis compañeros de departamento, de quienes he aprendido y quienes han estado siempre dispuestos a echarme una mano. En especial, me gustaría agradecer a Javier Fumanal, por haber sido un apoyo enorme en los momentos de mayor estrés, y haber estado siempre dispuesto a sacrificarse en los periodos en que yo estaba más atareado. Del mismo modo, quisiera hacer mención especial a Aitor Castillo y Borja de la Osa, por esas conversaciones en el café y durante las comidas, que tanta falta hacen para desconectar del trabajo.

Por supuesto, me gustaría dedicar un agradecimiento especial a mi supervisor, Humberto Bustince, por haber confiado siempre en mí, y habérmelo demostrado desde el primer momento, incluso en aquellos momentos en que yo mismo tenía dudas sobre mis capacidades o el rumbo del trabajo. Su apoyo y tutelaje me ha servido para superar los miedos que acompañan, inevitablemente me temo, a la labor investigadora, y dar los primeros pasos en un área fascinante.

Tengo que dedicar otra mención especial a Javier Fernández, por haber sido un profesor excepcional a lo largo de toda la carrera, además de la persona que me ofreció la oportunidad de entrar a formar parte de estos proyectos en primer lugar, y que ha estado siempre ahí para ayudarnos en todo momento, no solo a mí, sino a todos los miembros del departamento.

Muchas gracias también a Julio La Fuente, por sus inestimables aportaciones teóricas, sin las cuales, este trabajo nunca hubiese sido posible.

A mi familia, con cuyo apoyo he contado en todo momento, incluidos esos fines de semana en que, estresado y encerrado en mi habitación, han sabido darme el espacio que necesitaba, y los ánimos que me han ayudado a seguir adelante.

Y por último, a Alejandro, a Fermín, a Uxue, a Iñaki, a Patxi, a Jesús, a Jaime, a Elena, a Bea, a Elisa, y a tantos otros amigos a los que estoy tremendamente agradecido de haber tenido la fortuna de conocer.

Gracias a todos.

Contenidos

Lista de Figuras

Lista de Tablas

Sección 1

Introducción

El cerebro humano es un órgano capaz de realizar una ingente cantidad de tareas básicas, de las más diversas índoles, con resultados increíblemente precisos. No es algo extraño encontrarnos en una situación como la siguiente:

Es una mañana de verano. Nuestro cerebro envía una serie de impulsos a los músculos de nuestros brazos para que adquieran la postura, dediquen la fuerza exacta y ejecuten la serie de movimientos necesarios para levantar las persianas de nuestra habitación. Entonces, los rayos del sol —objeto que reconocemos inmediatamente, aislándolo del resto de la escena sin esfuerzo, y cuya mera presencia trae a nuestro consciente grandes cantidades de conocimiento almacenado— nos deslumbran, y en ese mismo instante, somos ya capaces de aventurarnos a deducir, mediante un proceso de inferencia lógica, que probablemente vaya a ser un día caluroso. Todo esto, en fracciones de segundo, y mientras, de manera inconsciente, el mismo cerebro responsable de cada una de esas operaciones, estaba haciéndose cargo de gestionar procesos automáticos de nuestro cuerpo en los que ni siquiera hemos reparado.

No es de extrañar, por tanto, que la persecución de un intento por replicar los mecanismos a partir de los cuáles opera una máquina tan extraordinaria, haya sido una de las metas históricas de la inteligencia artificial.

Aunque estamos lejos de este objetivo, que muchos consideran una mera ilusión inalcanzable, es un hecho que se han dado ya los primeros pasos por aplicar estos conceptos a la resolución de problemas concretos dentro del área. A los algoritmos de Machine Learning que siguen esta lógica, se los conoce como “redes neuronales”.

1.1 El comienzo: El perceptrón

La idea de “red neuronal” no es un concepto novedoso. Ya en el año 1958, Rosenblatt desarrollaba un algoritmo al que apodó como "perceptrón", basado a su vez en el modelo de neurona artificial presentado por McCulloch y Pitts en 1943 [1]. Según Rosenblatt, se puede considerar al perceptrón como una caja negra, a la que es posible enseñar a discriminar entre dos formas generalizadas, o “perceptos”, mediante un proceso que le enseñe a devolver la misma señal de salida para todos los estímulos que provengan de dos clases arbitrariamente constituidas [2].

El perceptrón es un algoritmo de aprendizaje supervisado que resuelve problemas de clasificación binaria. A nivel práctico, esto se consigue mediante un clasificador lineal, capaz de diferenciar entre dos conjuntos de datos linealmente separables. El algoritmo del perceptrón se limita, por tanto, al aprendizaje de los coeficientes y el término independiente de la ecuación del hiperplano capaz de separar los ejemplos de ambas clases. A los primeros se les llama pesos, mientras que al restante se lo conoce como sesgo (en adelante referido como “bias”, por su nombre en inglés).

La ecuación de un perceptrón con pesos $\mathbf{w} \in \mathbb{R}^n$ y bias $b \in \mathbb{R}$, que recibe como entrada un ejemplo $\mathbf{x} \in \mathbb{R}^n$, es:

$$f(\mathbf{x}) = \begin{cases} 1 & \text{si } \mathbf{w} \cdot \mathbf{x} + b \geq 0, \\ 0 & \text{en otro caso} \end{cases} \tag{1.1}$$

Tanto los pesos como el bias se inicializan aleatoriamente, y sus valores se modifican iterativamente, de modo similar a los algoritmos de aprendizaje supervisado actuales. Tras introducir un ejemplo $\mathbf{x_i}$ de la clase y_i en el clasificador y obtener una predicción $p_i = f(\mathbf{x_i})$, cada peso w_j se modifica en base a la siguiente expresión:

$$w_j := w_j + \gamma(p_i - y_i)x_{ij} \tag{1.2}$$

donde γ es un valor en $(0, 1]$ que regula el ritmo de aprendizaje, y j es el índice del parámetro que estamos modificando. Cuando p_i e y_i sean idénticos, los pesos se mantendrán estáticos, mientras que cuando sean distintos, la frontera de decisión se desplazará en la dirección que permita alternar la predicción p_i.

Cabe notar, que el caso de una única neurona de salida puede extenderse a n neuronas de salida, en lo que actualmente se conoce como capa de salida. Con una generalización de este tipo, y estrategias como la de un clasificador OVA (One VS All) es sencillo entrenar n perceptrones independientes, cada uno de ellos entrenado para predecir cada una de las posibles clases, y generalizar así la aplicación del perceptrón a problemas de clasificación multi-clase.

No tardaron en idearse otras variantes del perceptrón que utilizasen ligeras variaciones en la modelización de la neurona simulada. Este fue el caso del ADALINE (ADAptative LINear Element), que prescindía de la discretización de la salida en base a un umbral, devolviendo así un valor real como salida [3]. Además, a la hora de calcular el error, el ADALINE no solo tiene en cuenta el que la predicción sea correcta o no, sino que mide lo precisa que ha sido, calculando la distancia entre todas las predicciones del conjunto de entrenamiento y sus respectivas clases reales por medio del error cuadrático medio. Esto hace posible utilizar métodos como el descenso por gradiente para actualizar los pesos de la neurona de forma más eficiente.

La mayor limitación de todas estas estrategias, no obstante, es la incapacidad de garantizar la convergencia del algoritmo de entrenamiento cuando las clases a entrenar no son linealmente separables, como Minsky y Papert demostraron en [4], al probar que es imposible aprender la operación de la disyunción exclusiva.

1.2 Baches en el camino: El perceptrón multicapa y el algoritmo de retropropagación

Minsky y Papert no se quedaron ahí, sino que ofrecieron una solución a la incapacidad de diferenciar entre clases no linealmente separables. Una única neurona (o capa de neuronas) no tiene la capacidad para solucionar este problema, pero si se repite la estructura de la capa de salida, y se coloca como capa intermedia entre la entrada del modelo y esta, de modo que la salida de cada una de ellas se envíe como entrada de la capa de salida, se pasa a obtener una arquitectura con mayor capacidad de generalización, que permite combinar clasificadores lineales, por medio de funciones no lineales, para obtener fronteras de decisión no lineales. Estas capas intermedias se conocen actualmente como “capas ocultas”, y no están limitadas a una única por modelo, sino que puede colocarse una cantidad arbitraria de ellas.

De este modo, se pasa ya a tener una estructura similar a la de las neuronas del cerebro humano, conectadas unas con otras mediante sus sinapsis. Se está, por tanto, hablando ya de una estructura en forma de red: de red neuronal.

No obstante, Minsky y Papert encontraron otro problema, más severo y complejo de resolver que el anterior, y que paralizaría la investigación en el campo durante casi veinte años, en lo que pasaría a conocerse como "el invierno de la Inteligencia Artificial" [5].

El problema venía de la dificultad por ajustar los pesos de las capas ocultas de la red. Mientras que en la capa de salida existía un método de evaluación que permitía deducir, de manera directa, la estrategia necesaria para ajustar los pesos de la neurona de salida, lo mismo no podía decirse de las capas intermedias. Las salidas producidas por estas capas son enviadas a la capa de salida, sin pasar por ningún método de evaluación que permita indicar si son correctas o no.

La solución tardaría en llegar, no solo por la propia dificultad del problema, sino también por el clima de desinterés en la materia imperante los próximos años. En 1974, Werbos, inspirado por los trabajos del neurólogo austriaco Freud [6], y los múltiples desarrollos independientes que en la década de los 60 desarrollaran un algoritmo de optimización conocido como retropropagación (o "backpropagation" en inglés) [7], presentó en su tesis la posibilidad de aplicar dicho algoritmo a la optimización de todos los pesos de una red neuronal [8]. No obstante, su trabajo fue ignorado por la mayoría de la comunidad científica, y no sería hasta una década más tarde, cuando Rumelhart, Hinton y Williams rescataran la propuesta y popularizaran el método [9].

Este momento marcaría el resurgimiento del interés por las redes neuronales, y los desarrollos de la época siguen siendo fundamentales a día de hoy, como veremos en la sección 2.

1.3 Visión Artificial: Del cerebro al nervio óptico

Paralelamente a los avances en inteligencia artificial centrados en el desarrollo de modelos como el perceptrón multicapa, otra línea de investigación comenzó a tratar de adaptar estos mismos conceptos al campo de la visión artificial.

En 1968, Hubel y Wiesel publicaron [10], donde investigaban la respuesta que se producía en el córtex cerebral de algunos primates, como macacos y monos araña, al recibir distintos estímulos visuales. Durante este estudio, realizaron una categorización de las neuronas del córtex cerebral en base al tipo de estímulo que las activaba, distinguiendo entre célculas simples, complejas e hipercomplejas. Las simples se caracterizaban por campos receptivos muy pequeños, que ofrecían respuestas muy alta ante líneas rectas orientadas, mientras que las complejas e hipercomplejas contaban con campos receptivos más amplios, con una mayor invariabilidad a rotaciones y transformaciones.

En 1980, Fukushima desarrolló, en base a la teoría de Hubel y Wiesel, el "Neocognitrón", una arquitectura de red neuronal para reconocimiento de patrones, que hacía uso de los conceptos de célculas simples y células complejas, a las que en su trabajo apodaba "s-cells" y "c-cells" respectivamente [11]. Esta arquitectura se encontraba dividida, al igual que el perceptrón multicapa, en distintas capas. Cada una de estas capas está compuesta, alternamente por "s-cells" o "c-cells". Las primeras son las encargadas de extraer las características de una imagen de entrada, mediante la evaluación de las distintas regiones de la imagen, mientras que las segundas

conceden cierta invarianza contra rotación o distorsión a estas características extraídas. Hoy en día, es fácil ver la relación directa entre "s-cells" y capas de convolución, así como "c-cells" y capas de pooling. Fukushima indica, incluso, que las características extraídas por las primeras capas de la red tienden a ser formas sencillas, que se combinan para dar lugar a características más complejas en las siguientes capas, del mismo modo que sucede en las redes modernas.

La mayor diferencia entre el Neocognitrón y las redes convolucionales, no obstante, reside en su algoritmo de aprendizaje. Mientras que una red convolucional actualiza sus pesos mediante el proceso de retropropagación, del mismo modo que lo hace el perceptrón multicapa, el neocognitrón sigue un proceso de entrenamiento no supervisado, basado en el refuerzo de aquellas conexiones que reciben estímulos ya de por sí intensos.

La primera red convolucional moderna fue presentada en 1998 en [12], donde se empleó una arquitectura más cercana a lo que hoy entendemos por una arquitectura de este tipo para la clasificación de dígitos escritos a mano, utilizando el actualmente estándar dataset MNIST, que también se presentó en dicho artículo.

1.4 Deep Learning: El nuevo resurgir de las redes neuronales

Pese al resurgimiento en popularidad de las redes neuronales, y el amplio estudio que se les dedicó en los 90, el campo sufriría aún una nueva caída en popularidad.

Para que las redes neuronales pudiesen adquirir su máximo potencial, resolviendo tareas cada vez más complejas, era necesario aumentar el número de capas, neuronas, y por tanto parámetros de estas redes. Esto suponía dos problemas principales, que fueron los que motivaron su segunda caída.

Por un lado, a mayor número de parámetros, mayor capacidad de cómputo se hace necesaria para llevar a cabo el proceso de entrenamiento de las redes. La infraestructura de la época no estaba capacitada para lidiar con las estructuras modernas, que acostumbran a emplear millones de parámetros. Por el otro, para resolver tareas complejas, es necesario contar con un alto volumen de datos de entrenamiento. Aunque hoy en día existen enormes datasets públicos y las grandes corporaciones digitales invierten mucho esfuerzo en recopilar datos para su posterior tratamiento, esto era algo poco común en la época.

Es cierto que durante los 90 y la primera década de los 2000 se propusieron nuevas ideas que posteriormente resultarían determinantes, como el algoritmo de descenso por gradiente estocástico [12] o las redes neuronales recurrentes LSTM [13], pero 2009 fue el año que realmente propulsó hacia delante el campo, ahora ya conocido como "Deep Learning" por la cantidad de capas de los modelos utilizados. En ese año, Li, de la universidad de Stanford lanzó la competición ImageNet [14], una competición que ponía al alcance de los participantes 14 millones de imágenes del mundo real, clasificadas en una estructura jerárquica de clases, que serviría para entrenar los nuevos modelos del futuro. A su vez, Madhavan y Ng entrenaron una arquitectura de deep learning utilizando una GPU ("Graphics Processing Unit" o Unidad de Procesamiento Gráfico) programada en el lenguaje CUDA, liberado un año antes por Nvidia [15], para resolver un problema de aprendizaje no supervisado [16].

En 2012, Krhizevsky et al. lograron un hito histórico cuando ganaron la competición ImageNet con un modelo de red neuronal convolucional mucho más profunda que las existentes hasta la fecha, a la que apodaron AlexNet, y que entrenaron también utilizando una GPU [17].

Este sería, a partir de ese momento, el modelo que fijaría el estándar de trabajo en la línea del Deep Learning en adelante. La utilización de GPUs para el entrenamiento de las redes, gracias a su capacidad para ejecutar en paralelo miles de operaciones simples troncales a la utilización de las redes neuronales, y la utilización de grandes conjuntos de datos a partir de los cuáles entrenar los nuevos modelos.

1.5 Objetivo

En este trabajo nos centraremos en las redes neuronales convolucionales. En concreto, y tal y como se explicará más adelante, nuestro objetivo es sustituir el proceso encargado de reducir las imágenes de características generadas por una red de este tipo, que acostumbra a utilizar funciones de agregación muy simples, como el máximo o la media, por funciones más avanzadas.

Probaremos distintas funciones, como órdenes estadísticos y la integral de sugeno, así como combinaciones crecientes de estas funciones, basándonos en una sólida base matemática, y comprobaremos el efecto de las nuevas modificaciones en tres tipos de arquitecturas diferentes, cada una con mayor complejidad y número de parámetros que la anterior.

Por último, presentaremos un método de toma de decisión multimodal con el que pretendemos combinar las predicciones obtenidas por los distintos modelos, entrenados con distintas variantes de estas funciones, para tratar de mejorar el funcionamiento individual de los mismos.

Sección 2

Preliminares

2.1 Redes Neuronales Convolucionales

En el campo del “Deep Learning", se llama red neuronal convolucional a un tipo de red neuronal que opera sobre datos donde la información local es relevante, como pueden ser series temporales, imágenes o vídeo. El proceso de esta red puede dividirse en dos fases: Una primera, dedicada a la extracción de características relevantes a partir de estos datos, y una segunda más especializada que, en base a las características extraídas y al problema abordado, devuelve un resultado que trata de aproximarse lo máximo posible a la salida esperada para dicho problema.

En este caso, trabajamos con una tarea de clasificación de imagen, por lo que esta segunda parte será un clasificador que nos devolverá un vector de probabilidades que representarán la probabilidad de pertenencia a cada una de las posibles clases predichas. Otro ejemplo de problema habitualmente abordado mediante este tipo de arquitecturas es el de segmentación semántica de una imagen, donde habitualmente se devuelve la probabilidad de pertenencia a cada clase para cada pixel de la imagen de entrada.

Las redes convolucionales deben su nombre a la parte que actúa a modo de extractor de características. A nivel conceptual, suele describirse esta parte como compuesta por una serie de “capas" que aplican distintas transformaciones a los datos de entrada secuencialmente, esto es, la capa i-ésima opera sobre la salida generada por la capa $i-1$-ésima.

2.1.1 Capa de convolución

En las aplicaciones de procesamiento de imagen, la convolución de distintos “filtros" sobre una imagen es una herramienta habitual empleada en problemas de segmentación, detección de bordes o reducción de imagen. No obstante, los valores de dichos filtros deben ser establecidos a priori, lo que requiere experiencia en el campo y a menudo largos procesos de ajuste de parámetros al problema en cuestión. En contraposición, las redes convolucionales parten de una serie de filtros aleatorios que se ajustan de manera automática mediante el proceso de aprendizaje de la red.

Una capa de convolución está compuesta por N filtros W de dimensión $k \times k \times c$. Dada una entrada X de dimensión $h \times w \times c$, cada uno de los filtros de convolución se coloca sobre cada ventana de tamaño $k \times k \times c$ centrada en cada píxel $x_{m,n}$ de X. El filtro de convolución se va desplazando a lo largo de toda la imagen, tras aplicar la convolución a cada una de las ventanas de la imagen. La salida $Y_{m,n}$ generada para la ventana centrada en el pixel $x_{m,n}$ se calcula en base a la siguiente fórmula:

$$Y_{m,n} = \sum_{d=1}^{c} \sum_{j=1}^{k} \sum_{i=1}^{k} X_{m+i-\frac{k-1}{2}-1, n+j-\frac{k-1}{2}-1, d} \cdot W_{i,j,d} \tag{2.1}$$

Al resultado Y producido por un filtro de convolución se le llama imagen de características, dado que ofrece información sobre la presencia o ausencia de la característica buscada por el filtro en cuestión. Dado que se cuenta con N filtros de convolución, se generarán N imágenes de características que, para poder ser tratadas de manera cómoda por las siguientes capas de la red, se apilan unas con otras, dando lugar a una imagen de características tridimensional. Así, el canal d-ésimo de Y contiene el resultado de aplicar el filtro de convolución d-ésimo a X.

2.1.2 Capa de Pooling

Para que el proceso anterior sea útil, el número de filtros aprendidos en la red deberá ser elevado, y por consiguiente, la dimensionalidad de la imagen de características aumentará notablemente. Dado que la eficiencia y facilidad de entrenamiento del clasificador de nuestra red mejorarán si se opera sobre un vector de características de pocos atributos, resulta necesario reducir la imagen de características generada. Es aquí donde las capas de "pooling" entran en juego.

La función de estas capas es la de reducir la imagen de características gradualmente, tratando de preservar la máxima información significativa posible. A este fin, la matriz de entrada $X \in \mathbb{R}^{m \times n \times c}$ se descompone en ventanas disjuntas $X^{\alpha\beta}$ de dimensión $k \times k \times c$. Por cada canal de $X^{\alpha\beta}$, se lleva a cabo la siguiente operación:

$$Y_c^{\alpha\beta} = f(X_c^{\alpha\beta}) \tag{2.2}$$

donde $f : \mathbb{R}^{k \times k} \to \mathbb{R}$ es una función de agregación real, habitualmente el máximo o la media, y $X^{\alpha\beta} \in \mathbb{R}_c^{k \times k}$ es el conjunto de valores de la ventana de coordenadas α y β en el canal c.

El paso de X a través de una capa de pooling de tamaño $k \times k$, reduce el número de atributos por un factor de $k \times k$. Las sucesivas capas de pooling de la red siguen reduciendo el número de atributos, hasta llegar a un número aceptable para el clasificador de la red.

2.1.3 Función de activación

La salida generada por una capa de convolución, al igual que la de la neurona de un perceptrón, es una salida lineal, dado que todas las operaciones internas son operaciones lineales. Aunque esto a priori pueda parecer una característica deseable para facilitar la derivación de la misma, y por tanto el proceso de aprendizaje, conlleva una serie de problemas que impiden su uso.

Minsky y Papert ya probaron la necesidad de aplicar funciones no lineales a la salida de las neuronas de un perceptrón, cuando proponían la utilización de un perceptrón multicapa [4]. Si no se aplicasen no linearidades a la salida de una capa de Deep Learning, el hecho de añadir nuevas capas ocultas al perceptrón no tendría ninguna utilidad, dado que la combinación lineal de n funciones lineales sigue devolviendo una función lineal. Es decir, seguiría siendo imposible generar fronteras de decisión no lineales que clasificasen correctamente conjuntos de datos no linealmente separables.

Históricamente, las primeras funciones de activación empleadas eran variantes de la función sigmoide, como por ejemplo la función logística:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.3}$$

que devuelve valores en $(0, 1)$ o la función tangencial hiperbólica:

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.4}$$

cuyo rango es $(-1, 1)$. Dichas funciones, en especial la logística, otorgan una interpretación de probabilidad a la salida de la función que puede resultar útil a la hora de interpretar el comportamiento de la neurona. Del mismo modo, ambas funciones son derivables en todo su dominio, lo que evita problemas a la hora de aplicar el algoritmo de retropropagación para optimizar los parámetros de la red.

No obstante, en la práctica, existen limitaciones a la aplicabilidad de estas funciones en redes muy profundas. El problema del gradiente desvaneciente (o "vanishing gradient") se da cuando, a lo largo de la propagación de la derivada de la función de coste a los parámetros de la red, se aplica la derivada de estas funciones de activación a valores muy pequeños o muy grandes. En estos casos, y debido a las asíntotas verticales a las que tienden estas funciones, los valores de salida se aproximan mucho al cero, y la modificación aplicada a los pesos se vuelve despreciable, impidiendo el correcto entrenamiento de la red. Además, incluso para el resto de valores de entrada, los valores devueltos siempre son menores que 1, lo que reduce progresivamente el gradiente propagado hacia atrás hasta desvanecerse.

Nair y Hinton propusieron en 2010 una nueva función de activación, conocida como ReLU ("Rectified Linear Unit" o Unidad Lineal Rectificada) que soluciona este problema [18]. La ecuación de esta función es la siguiente:

$$f(x) = \begin{cases} 0 & \text{si } f(x) < 0 \\ x & \text{en otro caso} \end{cases} \tag{2.5}$$

Las ventajas son claras: las derivadas de la función son directas, lo que la convierte en la función de activación más eficiente computacionalmente. A pesar de no ser derivable en todo su dominio, a nivel práctico esto no supone un problema, dado que las redes neuronales no suelen alcanzar óptimos locales, sino que solo reducen su error hasta puntos aceptables, y por tanto no llegan a alcanzar un punto en el que el gradiente llegue a hacerse cero [19].

Por otro lado, sin ser una función lineal, mantiene un comportamiento lineal para cualquier valor positivo de entrada, lo que facilita la optimización del modelo [20]. Además, el problema del gradiente desvaneciente se mitiga para todas las entradas positivas.

Es importante mencionar que, pese a que en el pasado se sospechaba que una ventaja adicional de esta función de activación era el hecho de que producía salidas dispersas para cada capa [21], en los últimos años parece haber indicios de que esto no supone realmente mejora alguna, como parece probar el hecho de que la activación Maxout pueda mejorar su funcionamiento para ciertos problemas, sin producir salidas dispersas [22].

2.1.4 Propagación hacia adelante

Como ya hemos comentado, uno de los puntos fuertes de la arquitectura de una red convolucional, es el hecho de que hace posible combinar en un mismo modelo un extractor de características y un clasificador. Las capas de convolución y pooling componen la lógica principal del extractor de características. El clasificador, por su parte, puede ser de muchos tipos, aunque es muy común la utilización de un perceptrón multicapa, que comparte algoritmo de optimización con el extractor de características.

Un perceptrón multicapa es una arquitectura formada por neuronas artificiales dispuestas en diferentes "capas". Todas las neuronas de cada capa están conectadas con todas las neuronas de la capa siguiente. Estas capas pueden ser de tres tipos: la capa de entrada, que es la que recibe los valores de un ejemplo a clasificar, cada uno en una neurona; la capa de salida, que cuenta con tantas neuronas como clases de salida, y que devuelve a partir de cada una de ellas, una probabilidad de pertenencia a dicha clase; y las capas ocultas, que están conectadas entre sí. La elección en el número de neuronas y capas no es un proceso trivial, y suele requerir cierto grado de prueba y error.

Nótese que en el caso de una red convolucional, la capa de entrada del clasificador recibe como entrada las características devueltas por el extractor de características, que actúa a modo de preprocesado de los datos de entrada de la red. Aunque estas características se estructuran en forma matricial, la solución más habitual es redimensionar dicha matriz para darle forma de vector de valores que pueda ser aceptado por el clasificador.

Las conexiones entre todas estas capas están ponderadas, como en el caso del perceptrón simple, y se inicializan de manera aleatoria.

Propagación hacia adelante es el nombre con el que se conoce el proceso secuencial por el cual la información de entrada de cualquier red neuronal va siendo transformada, capa a capa, hasta producir un resultado en la capa de salida. El resultado producido por la neurona i-ésima de la capa l-ésima se calcula mediante la siguiente expresión:

$$x_i^{(l)} = f(x^{(l-1)} \cdot w_i^{(l)} + b_i^{(l)}) \tag{2.6}$$

donde $x^{(l-1)}$ es un vector fila que contiene la salida de todas las neuronas de la capa $(l-1)$-ésima, $w_i^{(l)}$ es un vector columna con los pesos de las conexiones de cada neurona de la capa anterior con esta, $b_i^{(l)}$ es el bias de esta neurona y $f(x) : \mathbb{R} \to \mathbb{R}$ es la función de activación no lineal de la neurona.

Un proceso similar se lleva a cabo en la fase del extractor, donde cada capa produce, bien una matriz de características, como en el caso de las capas de convolución, bien una reducción de la misma, como sucede en las de pooling. El resultado de cada capa se envía como entrada de la siguiente, y se sigue operando secuencialmente.

El paso del extractor al clasificador suele limitarse a tomar la matriz de características tridimensional producida por la última capa del primero, y transformarla en un vector unidimensional que se envía a la capa de entrada del segundo, para seguir con el proceso normal de propagación.

En la salida producida por la red, y para darle interpretación de vector de probabilidades, lo que servirá para facilitar el uso de funciones de coste estándar, es común normalizar los valores producidos por la capa de salida, mediante la llamada función softmax [23]. En realidad, softmax es el nombre que se le asigna a la función exponencial normalizada, por su posible interpretación como una versión suavizada de la función máximo. La expresión de la función softmax $f : \mathbb{R}^C \to [0,1]^C$, es la siguiente:

$$f(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{C} e^{x_j}} \text{ para } i = 1 \dots C \tag{2.7}$$

2.1.5 Algoritmo del Gradiente y retropropagación

Como ya hemos mencionado, la red cuenta con una serie de parámetros, como los filtros de convolución, o los pesos del clasificador, que se inicializan aleatoriamente y adaptan sus valores mediante un proceso iterativo de optimización. Los métodos más habitualmente empleados son el algoritmo del gradiente y sus variantes [24].

El algoritmo del gradiente es un algoritmo clásico de optimización que opera de la siguiente manera: En primer lugar, en base al problema a resolver, se fija una función de coste que determinará el rendimiento de la red, comparando los resultados esperados con los obtenidos. Si estos coinciden, los parámetros se mantendrán estables, mientras que si existen diferencias entre ambos, se procederá a su modificación, en busca de la minimización (o maximización) de la función de coste.

Una de las funciones de coste más populares, es la entropía cruzada. Si se tienen C posibles clases en las que clasificar un ejemplo, la función de coste para dicho ejemplo será:

$$J(\mathbf{P}, \mathbf{Y}) = -\sum_{i=1}^{C} Y_i log(P_i) \tag{2.8}$$

donde $\mathbf{P}$ es un vector de C elementos obtenido como salida de la red, tal que P_i representa la probabilidad de pertenencia asociada a la clase i-ésima e $\mathbf{Y}$ es un vector de C elementos tal que $Y_i = 1$ si el ejemplo pertenece a la clase i-ésima, o $Y_i = 0$ en caso contrario.

En base a la función de error definida, el proceso de actualización de los parámetros de la red es mediante el cálculo de las derivadas parciales de la función de coste, con respecto al parámetro en cuestión. Dado que los valores generados en la capa i-ésima son transformados por las capas posteriores, previo a la obtención del vector de salida $\mathbf{Y}$, las derivadas parciales de $J(\mathbf{P}, \mathbf{Y})$ deben tener en cuenta dichas transformaciones. El proceso a seguir se conoce como "retropropagación", debido a que las derivadas se calculan partiendo del final de la red y aplicando la regla de la cadena hacia atrás, capa a capa, hasta alcanzar el parámetro a actualizar.

Una vez se ha calculado la derivada parcial con respecto al parámetro θ, $\frac{\partial J(\mathbf{P},\mathbf{Y})}{\partial \theta}$, este se actualiza según la siguiente fórmula:

$$\theta := \theta - \gamma \frac{\partial J(\mathbf{P}, \mathbf{Y})}{\partial \theta} \tag{2.9}$$

donde $\gamma \in \mathbb{R}^+$ es un factor conocido como "factor de aprendizaje", que controla el ritmo al que se actualizan los valores de la red, y que suele tomar valores pequeños, habitualmente en el intervalo $(0, 1)$.

Existen distintas estrategias relacionadas con el ritmo de actualización de los pesos, que pueden actualizarse tras predecir cada ejemplo, tras analizar el error acumulado por todos los ejemplos, o el de ciertos "lotes" (batches) de ejemplos. En todos los casos, el proceso se repite de manera iterativa durante el proceso de entrenamiento de la red, hasta que cierta condición de parada sea cumplida.

La variante más comúnmente empleada es la conocida como descenso por gradiente estocástico (o "stochastic gradient descent", en inglés). Esta variante trabaja actualizando los valores de los parámetros a partir del error cometido para un único ejemplo, o lote de ejemplos (normalmente referido como descenso por gradiente de mini-lotes o "mini-batch gradient descent"), del conjunto de entrenamiento totales. El gradiente calculado es, por tanto, una aproximación del gradiente real. De esta

forma, se sacrifica algo de precisión a la hora de actualizar los pesos del modelo, en pos de aliviar la carga computacional que supone el cálculo del gradiente total [25].

Es interesante mencionar, no obstante, que hay indicios de que el ruido introducido al emplear una aproximación del gradiente real puede beneficiar al resultado obtenido, ya que permite converger a mínimos locales distintos al que correspondería si se emplease el auténtico gradiente de la función de coste [26].

2.2 Funciones Crecientes en $\mathbb{R}$

A continuación recogemos cierta teoría matemática en base a la cuál desarrollaremos la experimentación que presenta en la Sección 5.

A partir de ahora, se asume que $2 \leq n \in \mathbb{N}$, $1 \leq r \in \mathbb{N}$.

2.2.1 Funciones crecientes

Llamamos función creciente a una función $\mathbf{A}\colon \mathbb{R}^n \to \mathbb{R}$ tal que $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, $\mathbf{x} \leq \mathbf{y}$ implica $\mathbf{A}(\mathbf{x}) \leq \mathbf{A}(\mathbf{y})$

2.2.2 Estadísticos de orden

Anotación 2.2.1. Sea $N = \{1, \ldots, n\}$ y Σ_n para el grupo de todas las permutaciones de N. Si $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ y $\sigma \in \Sigma_n$, denotamos por $\mathbf{x}_\sigma = (x_{\sigma(1)}, \ldots, x_{\sigma(n)})$; si $x_{\sigma(1)} \leq \cdots \leq x_{\sigma(n)}$, lo denotamos por $\sigma \in \mathbf{x}_{(\nearrow)}$.

Sea $r \in \{1, \ldots, n\}$. Denotamos por $\mathbf{OS}_r$ al r-ésimo estadístico de orden, esto es, la función $\mathbf{OS}_r\colon \mathbb{R}^n \to \mathbb{R}$ dada por, si $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$, $\mathbf{OS}_r(\mathbf{x}) = x_{\sigma(r)}$, donde $\sigma \in \mathbf{x}_{(\nearrow)}$.

Como casos particulares tenemos el máximo $\mathbf{max} = \mathbf{OS}_n$ y el mínimo $\mathbf{min} = \mathbf{OS}_1$.

2.2.3 Integral de Sugeno

Definición 2.2.2. Una medida difusa en N es una función $\nu\colon 2^N \to [0, +\infty)$ tal que

1. $\nu(\emptyset) = 0$ y
2. $S \subseteq T \subseteq N$ implica $\nu(S) \leq \nu(T)$.

Definición 2.2.3. La *integral de Sugeno* asociada a la medida difusa ν es la función

$$\mathbf{S}_\nu\colon \mathbb{R}^n \to \mathbb{R}$$

dada por, para $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$,

$$\mathbf{S}_\nu(\mathbf{x}) = \mathbf{max}\left(\mathbf{min}(x_{\sigma(1)}, \nu(N_1^\sigma)), \ldots, \mathbf{min}(x_{\sigma(n)}, \nu(N_n^\sigma))\right),$$

donde $\sigma \in \mathbf{x}_{(\nearrow)}$ y $N_i^\sigma = \{\sigma(i), \ldots, \sigma(n)\}$.

Sección 3

Revisión del estado del arte en Deep Learning

En los últimos años la investigación en el campo del Deep Learning ha visto un interés desmedido, y como consecuencia se han desarrollado avances en muchas direcciones distintas, que sería imposible resumir en un mismo trabajo. Por ello, en esta sección vamos a presentar las ideas que han influido directa o indirectamente en el desarrollo de nuestros experimentos.

Presentamos múltiples técnicas que se han utilizado en las arquitecturas con las que se ha llevado a cabo la experimentación de este trabajo, y que mostraremos en la sección 3. Aunque algunas resultan más específicas a arquitecturas concretas, la mayoría han visto una amplia aceptación, con muchas de ellas formando parte del kit de herramientas básico con el que se desarrollan nuevos modelos de Deep Learning, junto con las capas de convolución o pooling.

3.1 Batch Normalization

Un problema conocido a la hora de entrenar una arquitectura de Deep Learning es el desplazamiento covariante interno (o "internal covariate shift") [27]. Este problema se da en una situación como la siguiente:

Al comienzo de la fase de entrenamiento los pesos de todas las capas de la red se inicializan aleatoriamente. Cuando la red recibe un ejemplo a clasificar, estos pesos transforman la entrada de manera secuencial, hasta producir una salida en la última capa. Pasadas n iteraciones, no obstante, estos pesos habrán sido modificados en un intento por optimizar la arquitectura para minimizar la función de coste definida. Por tanto, para un mismo ejemplo de entrada $\mathbf{x}$, la entrada que recibirá la capa i-ésima de la red, para $i > 1$, será distinta en la iteración n de lo que fuera en la iteración 1.

Esto puede dificultar la optimización de los pesos de capas más profundas de la red, puesto que las obliga a aprender a producir salidas similares para distribuciones de entrada diferentes, que en realidad se corresponden con una distribución inicial idéntica.

La capa de "batch normalization" se introdujo con la idea de aliviar este problema en [28]. Para ello, y suponiendo que se trabaje con "mini-lotes" de ejemplos, la salida generada por una capa se normaliza en base a la media y desviación estándar del conjunto de ejemplos del lote. Para el $k - simo$ elemento de la entrada $\mathbf{x} \in \mathbb{R}^m$, $x^{(k)}$, el resultado producido será:

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mu_{(k)}}{\sqrt{\sigma^2_{(k)} + \epsilon}} \tag{3.1}$$

donde $\mu_{(k)}$ y $\sigma(k)$ son la media y desviación estándar del valor $\mathbf{x^{(k)}}$ respectivamente, calculadas sobre todos los ejemplos del mini-lote. La misma lógica se sigue para salidas de la forma $x \in \mathbf{R}^{n \times n}$, como las producidas por una capa de convolución.

Para evitar limitar la capacidad de representación de la red, tras el proceso de normalización, la salida producida se escala y desplaza en base a dos nuevos parámetros $\gamma^{(k)}$ y $\beta^{(k)}$ que se aprenden a la vez que el resto de parámetros de la red, mediante la operación:

$$y^{(k)} = \gamma^{(k)} \hat{x}^{(k)} + \beta^{(k)} \tag{3.2}$$

Esta operación haría posible, incluso deshacer el efecto producido por la normalización, si esto fuera lo más eficiente.

Como efecto beneficioso añadido, la "batch normalization" también mejora el flujo del gradiente a través de la red, al hacerlo menos dependiente de la escala de los parámetros. Si se combina con funciones de activación como la función ReLU, otro de los beneficios de esta capa es que evita que el flujo de información quede interrumpido cuando los valores producidos son negativos, dado que desplazará de nuevo estos valores, fijados a cero por la función de activación.

No obstante, en los últimos años ha surgido cierta discusión sobre si el objetivo inicial para el que la capa fue inventada, solucionar el problema de desplazamiento covariante interno, es realmente lo que esta capa soluciona. En [29] los autores comparan la eficiencia de una arquitectura VGG [30] sin "batch normalization", con "batch normalization" y con "batch normalization" pero seguida de ruido artificial, que simula el problema del desplazamiento covariante interno. Los resultados muestran que el comportamiento en términos de eficiencia de las dos últimas redes es prácticamente idéntico, y superior al de la primera.

Los autores atribuyen el impacto positivo de la aplicación de una capa de batch normalization, en cambio, al suavizado de la función de coste, y demuestran que reemplazar la función de normalización por otras estrategias de normalización produce un efecto similar, con resultados prácticamente idénticos en términos de eficiencia, a los de una capa de este tipo.

3.2 Dropout

Las redes neuronales son proclives a sobreajustar sus parámetros a los datos del conjunto de entrenamiento, lo que daña la capacidad de generalización del modelo y su eficacia tratando ejemplos ajenos a este conjunto de datos. A lo largo de los años se han presentado distintas estrategias de regularización, como el uso de términos de penalización L1 o L2 en la función de coste [31], o la propia "batch normalization" de la que hemos hablado anteriormente.

La estrategia de dropout es otro método de regularización más, cuyo funcionamiento resulta muy simple [32]:

Durante el entrenamiento, lo que ocurre en cada iteración es que la salida de una neurona se desactiva, o fija a cero, en base a una probabilidad p fijada de ante mano. Este mismo proceso se repite para todas las neuronas de la red, generando, en cada iteración, una subred compuesta por aquellas neuronas que no han sido desactivadas de manera aleatoria.

Esto tiene varias ventajas: por un lado, el comportamiento de cada neurona no pueden codepender excesivamente del resto, dado que sus conexiones podrían desactivarse aleatoriamente. Si se quiere obtener un modelo robusto, capaz de adaptarse a la desactivación de ciertas neuronas, será necesario que estas sean independientes unas de otras, lo que produce un efecto regularizador en la red.

Por otro lado, una vez se ha concluido el entrenamiento, el efecto del dropout se reemplaza por una ponderación de la salida producida por cada neurona, multiplicándola por la probabilidad p. En la práctica, este comportamiento se aproxima a la combinación de las 2^n posibles subredes que pueden construirse desactivando una red dada con n neuronas, en una suerte de ensemble ejecutado de forma extremadamente más eficiente que la combinación real de todas esas posibles redes.

Aunque en teoría, la misma lógica puede aplicarse tras la salida de las capas de convolución, esto a menudo tiene un efecto pernicioso sobre el comportamiento de la red, como se prueba experimentalmente en [33], por lo que en la práctica rara vez se emplea de este modo.

3.3 Global Average Pooling

La capa de pooling global medio (o "Global Average Pooling", también conocida como GAP), fue propuesta en [34] como una alternativa al uso de un perceptrón multicapa como clasificador final de la red.

En lugar de contar con dicho clasificador, la idea de los autores es la de prescindir de este clasificador y hacer cargo de su lógica al propio extractor de características. Esto se consigue colocando como última capa de la red, una convolución con tantos filtros como número de clases existentes. El resultado de dicha capa es el de una matriz de características con tantos canales como clases posibles. Tras esto, cada una de las imágenes de características obtenidas se reduce a un único valor utilizando la media aritmética, de modo que finalmente obtenemos un vector con tantos elementos como número de clases. A este vector se le puede aplicar la función "softmax" para conceder a sus valores la interpretación de probabilidades de pertenencia en base a las cuáles puede realizarse una predicción.

La lógica detrás de esta estrategia es la de contar con un clasificador más afín al proceso del extractor de características de una red convolucional. Los filtros de características obtenidos por esta última capa se encargarían de aprender la correspondencia entre una matriz de características y un vector de clases, dando a la salida resultante la interpretación de mapas de confianza de cada una de las clases.

Un efecto de la aplicación de esta capa es que elimina la necesidad de ajustar los pesos del hipotético perceptrón multicapa que se habría empleado en una red convolucional estándar, lo que supone una reducción notable en el número de parámetros y una menor probabilidad de sobreajuste al conjunto de entrenamiento.

3.4 MLPConv

La capa "MLPConv" es el nombre que los autores del concepto de "Global Average Pooling" dan a la capa que presentan al mismo tiempo en [34]. El nombre deriva del hecho de que esta capa sustituye la operación de una capa de convolución, por la aplicación de un perceptrón multicapa.

Esto es, para una imagen de entrada X de dimensión $h \times w \times c$, se recorren cada una de las ventanas de tamaño $k \times k \times c$ centradas en cada píxel $x_{m,n}$ de X, como en el caso de la convolución estándar. No obstante, estos valores son empleados

en este caso como entrada de un perceptrón multicapa con una capa de entrada de $k \cdot k \cdot c$ neuronas, que utiliza la función ReLU como función de activación. Este perceptrón puede tener múltiples capas ocultas, y la salida y producida por su efecto, de dimensión $1 \times 1 \times d$, donde d será el número de neuronas fijadas en la capa de salida, puede interpretarse como la salida que produciría una capa de convolución con d filtros distintos.

Nótese que, en la práctica, la implementación de una capa de este tipo se puede obtener mediante una capa de convolución estándar con filtros de tamaño $k \times k$, seguida de tantas capas de convolución con filtros 1×1 como capas ocultas quieran emplearse. El número de filtros de cada una de estas capas representará, en este caso, el número de neuronas de cada una de ellas.

La utilización de un perceptrón multicapa viene motivada por el hecho de que este es capaz de aproximar cualquier función, y por tanto debería poder aprender la mejor función de extracción de características posible. Este es el paso lógico tras las capas "Maxout" presentadas en [22], que tratan de aproximar cualquier función convexa por medio de una función lineal por partes.

3.5 Supervisión de capas ocultas

Conforme se van añadiendo más capas a los modelos de Deep Learning, el proceso de entrenamiento se vuelve más complicado, debido a posibles problemas de gradiente desvaneciente o explosivo (el proceso contrario por el cuál este se vuelve excesivamente grande). Las arquitecturas modernas, que emplean decenas o cientos de capas, introducen a menudo herramientas para lidiar con esta inestabilidad durante el entrenamiento. Una de estas herramientas es la "supervisión de capas ocultas" (o "hidden layer supervision" en inglés) presentada en [35].

Como su nombre indica, este tipo de estrategia "supervisa" la salida producida por cada una de las capas ocultas de la red, en el sentido de que utiliza esta información para tratar de predecir la clase de cada ejemplo, como si se tratase de la salida generada por una red truncada a la altura de dicha capa. En términos prácticos, podemos entender esto como la colocación de un clasificador tras la salida de cada capa oculta, que devuelva un vector de probabilidades con tantos valores como clases posibles. Es decir, tras cada capa, contaremos con un clasificador similar, aunque no necesariamente del mismo tipo exacto, que el clasificador final de la red.

Es importante remarcar que la salida generada por una capa oculta se sigue enviando sin modificar a la siguiente capa, a la par que al clasificador, de modo que podemos entenderlo como una bifurcación del flujo de salida de la capa. No es necesario realizar ninguna modificación, por lo tanto, a la entrada recibida por la siguiente capa oculta, pues sigue siendo idéntica a la que recibiría si no empleásemos ningún mecanismo de supervisión.

Si para una red concreta con L capas ocultas, fijamos una función de coste $J(\mathbf{P}, \mathbf{Y})$, p. ej. la entropía cruzada descrita en la Ecuación 2.8, añadiremos un término extra a esta función que ponderará la salida generada por cada uno de los nuevos clasificadores añadidos. De este modo, la nueva función de coste pasará a ser la siguiente:

$$\hat{J}(\mathbf{P}, \mathbf{Y}) = J(\mathbf{P}, \mathbf{Y}) + \sum_{l=1}^{L} \alpha_l j_l(\mathbf{P}, \mathbf{Y}) \tag{3.3}$$

donde $j_l(\mathbf{P}, \mathbf{Y})$ indica el error cometido por el predictor añadido tras la capa l-ésima, conocida como coste compañero (o "companion loss"), y α_l es el parámetro

que regula la importancia de dicho término dentro del coste total. Nótese que los autores originales formulan la ecuación anterior suponiendo la utilización de clasificadores L2-SVM [36], esto es, clasificadores SVM con regularización L2, pero la Ecuación 3.3 es una generalización de dicha expresión.

Puesto que el objetivo de la red sigue siendo el de obtener una predicción final correcta, independiente de la información predicha por el resto de clasificadores, podemos añadir un hiperparámetro γ a la ecuación anterior, que servirá como umbral para desechar la influencia del coste compañero una vez que $j_l(\mathbf{P}, \mathbf{Y}) \leq \gamma$. La forma de la ecuación resultante será, en este caso:

$$\hat{J}(\mathbf{P}, \mathbf{Y}) = J(\mathbf{P}, \mathbf{Y}) + \sum_{l=1}^{L} \alpha_l [j_l(\mathbf{P}, \mathbf{Y}) - \gamma]_+ \tag{3.4}$$

Otra alternativa es la de mantener la Ecuación 3.3, pero reducir gradualmente el valor de α_l a lo largo del entrenamiento, hasta hacerlo cero.

El resultado de utilizar este tipo de mecanismo de supervisión es el de obtener características más discriminatorias en cada capa de la red, que deberían resultar en un entrenamiento más eficiente del clasificador, acelerando la convergencia del algoritmo de entrenamiento. Además, esta lógica cumple al mismo tiempo cierto efecto regularizador sobre el modelo, que al mismo tiempo reduce la probabilidad de encontrar problemas de gradiente desvaneciente o explosivo.

3.6 DenseNet

DenseNet es el nombre de una arquitectura de DeepLearning introducida en 2017 por Huang et al. en [37]. Inspirada en gran medida por las arquitectuas ResNet [38], DenseNet está basada en en el concepto de conexiones identidad (o "identity conexions") utilizado en estas arquitecturas, que consiste en propagar la salida de una capa, no solo a la capa siguiente, sino a las n capas siguientes de la red, favoreciendo así el flujo del gradiente en el proceso de retropropagación y facilitando el entrenamiento de la red. DenseNet lleva esta idea un paso más allá y propone conectar todas las capas secuenciales de la red, de modo que, para una red con L capas, la salida de la i-ésima capa se envía como entrada a las siguientes $l^+ = i + 1 \ldots L$ capas, junto con las salidas de las anteriores $l^- = 1 \ldots i - 1$ capas.

El objetivo que persigue esta estrategia es doble: por un lado, dado que las arquitecturas ResNet han demostrado que las conexiones identidad son una herramienta muy eficaz a la hora de prevenir problemas de entrenamiento como el gradiente desvaneciente y explosivo, DenseNet busca reutilizar este concepto para permitir el entrenamiento de una red muy profunda. Por otro, y de manera un tanto contraintuitiva, se reduce la necesidad de aprender un número tan elevado de parámetros como estas otras arquitecturas.

La explicación detrás de este hecho se apoya en el concepto que los autores denotan como "conocimiento global" de la red. Puesto que todas las salidas producidas por cada capa de la red se mantienen presentes a lo largo de la propagación hacia delante, podemos entender que el "conocimiento" aprendido por una capa, la salida generada por esta, se agrega a este conocimiento global, haciendo innecesario reaprender las mismas representaciones a distintas alturas de la red, como sucede con otras arquitecturas de redes muy profundas [39]. Esto permite utilizar capas con muy pocos filtros, lo que facilita más aún el entrenamiento.

La arquitectura de una DenseNet puede dividirse en una serie de bloques fundamentales, cada uno de los cuáles cumple una función y está compuesto, a su vez, de varias capas distintas. Estos son:

- Bloque de entrada: Consiste en el primer filtrado de la imagen de entrada de la red, utilizando una capa de convolución seguida de una capa de "batch normalization". Si el tamaño de las imágenes de entrada es muy grande, podemos añadir una capa de pooling para realizar una primera reducción. Tras este primer bloque, la salida generada se envía al primer bloque denso de la red.

- Bloques densos: Son la parte más importante de la red, ya que es la que introduce la lógica de conexiones entre capas. Cada bloque denso está compuesto por una secuencia repetitiva de las siguientes capas: batch normalization, ReLU como función de activación, y convolución. Opcionalmente, podemos añadir también una capa de dropout tras la capa de convolución si encontramos problemas de sobreajuste de la red. Teniendo en cuenta que la salida de cada convolución persiste a lo largo de todo el proceso de la red, el número de filtros de cada una deberá ser relativamente pequeño. Este parámetro k se mantiene constante para toda la red, y recibe el nombre de "factor de crecimiento", pues es el número de nuevas imágenes de características que se añaden al conocimiento global de la red tras cada capa.

  Para reducir el número de filtros sobre los que cada capa de convolución debe trabajar, y por tanto el número de canales de los filtros a aprender y el número de parámetros de la red en conjunto, podemos anteponer a cada una de las secuencias anteriores, otra idéntica con filtros de convolución de tamaño 1×1, que haga las veces de cuello de botella. El número de filtros de estas capas cuello de botella quedará fijado de antemano, para evitar el crecimiento lineal en el número de filtros aprendidos por cada capa sucesiva que implica la conexión de las capas.

- Capas de transición: Tras cada bloque denso se procede a la reducción del tamaño de las imágenes de características. Este bloque está compuesto por una capa de "batch normalization", seguida de una capa de convolución con filtros de tamaño 1×1, para finalizar con una capa de pooling. Existe una variante de la arquitectura DenseNet que aprovecha estas capas de transición para comprimir el conocimiento global de la misma, en base a un parámetro $0 < \theta < 1$. En este caso, el número de filtros de la capa de convolución queda fijado a θm, donde m es el número de filtros producidos por las capas densas hasta el momento.

- Clasificador: Para terminar, la red aplica una capa de pooling global promedio con el que la salida producida por el último bloque denso se reduce a un vector de m valores. Este se introduce como entrada a un perceptrón sin capas ocultas, con tantas neuronas de salida como clases tenga el problema.

Sección 4

Objetivo y Justificación Matemática

4.1 Objetivo

Con los conceptos teóricos un poco más fijados, recordamos de nuevo el objetivo del trabajo: modificar el proceso de funcionamiento de la función de Pooling. Como hemos comentado en 2.1.2, la función de una capa de pooling es la de reducir la información de salida producida por acción de una o varias capas de convolución, de manera que la dimensionalidad de las características extraídas se vaya reduciendo progresivamente, en vistas a ser empleada como entrada de un clasificador.

Las funciones más comunes, e históricamente empleadas para llevar a cabo este proceso, han sido el máximo y la media aritmética, funciones de agregación estándar, muy utilizadas en otros procesos de fusión de datos. Nuestra hipótesis es, sin embargo, que, al menos para algunas arquitecturas o problemas, deben existir reducciones que, sin suponer un cambio grave en el comportamiento de la red, ofrezcan un mejor rendimiento, como ya se ha demostrado en otros campos como el de la reducción de imagen [40].

Existen, de hecho, pruebas empíricas en artículos previos que parecen probar esta hipótesis. En [41], los autores prueban el efecto de combinar máximo y media en una misma operación, por medio de una combinación convexa, consiguiendo mejorar los resultados que cualquiera de estos métodos individuales reporta. Nuestro objetivo será similar: por un lado, probaremos distintas funciones, más allá del máximo y la media y analizaremos sus resultados; por otro, probaremos combinaciones de estas, aunque sin limitarnos a garantizar que sean convexas, dado que experimentalmente comprobamos una mejora en los resultados cuando evitamos imponer dicha restricción.

En un intento por analizar las propiedades matemáticas que es conveniente exigir a estas funciones, y basándonos en el hecho de que, tanto el máximo, como la media, como la combinación de ambas, dan como resultado funciones crecientes, imponemos esta propiedad a las combinaciones a utilizar, como parte de un futuro estudio que analizará el efecto de otras propiedades.

4.2 Justificación matemática

El siguiente desarollo matemático se ha producido con la colaboración del doctor Julio Pedro Lafuente del departamento de Estadística, Informática y Matemáticas de la Universidad Pública de Navarra, y sirve de justificación para la experimentación que presentamos en la Sección 5.

A continuación, mostramos las condiciones según las cuales una combinación de funciones crecientes obtiene como resultado una función creciente. Dichas combinaciones serán las que empleemos como combinaciones de funciones en la fase de pooling.

4.3 Condiciones de crecimiento

Fijamos $\mathbf{0} = (0,\dots,0)$, $\mathbf{1} = (1,\dots,1)$, $\mathbf{e}_i = (0,\dots,\underset{i}{1},\dots,0)$.

Anotación 4.3.1. Sean $\mathbf{A}_1,\dots,\mathbf{A}_r\colon \mathbb{R}^n \to \mathbb{R}$ funciones crecientes. Fijamos

$$\mathcal{I}(\mathbf{A}_1,\dots,\mathbf{A}_r) = \\ = \{(\alpha_1,\dots,\alpha_r) \in \mathbb{R}^n \mid \sum_{i=1}^{r} \alpha_i \mathbf{A}_i\colon \mathbb{R}^n \to \mathbb{R} \text{ es una función creciente}\}.$$

Proposición 4.3.2. *$(\mathcal{I}(\mathbf{A}_1,\dots,\mathbf{A}_r),+)$ es un semigrupo con elemento neutro $(0,\dots,0)$. Además, $(\mathbb{R}^+ \cup \{0\})^r \subseteq \mathcal{I}(\mathbf{A}_1,\dots,\mathbf{A}_r)$.*

Prueba. Sea $\mathbf{x},\mathbf{y} \in \mathbb{R}^n$, $\mathbf{x} \leq \mathbf{y}$.
Si $(\alpha_1,\dots,\alpha_r),(\beta_1,\dots,\beta_r) \in \mathcal{I}(\mathbf{A}_1,\dots,\mathbf{A}_r)$, tenemos

$$\begin{aligned}((\alpha_1+\beta_1)\mathbf{A}_1 + \cdots + (\alpha_r+\beta_r)\mathbf{A}_r)(\mathbf{x}) &= \\ &= (\alpha_1\mathbf{A}_1 + \cdots + \alpha_r\mathbf{A}_r)(\mathbf{x}) + (\beta_1\mathbf{A}_1 + \cdots + \beta_r\mathbf{A}_r)(\mathbf{x}) \\ &\leq (\alpha_1\mathbf{A}_1 + \cdots + \alpha_r\mathbf{A}_r)(\mathbf{y}) + (\beta_1\mathbf{A}_1 + \cdots + \beta_r\mathbf{A}_r)(\mathbf{y}) \\ &= ((\alpha_1+\beta_1)\mathbf{A}_1 + \cdots + (\alpha_r+\beta_r)\mathbf{A}_r)(\mathbf{y})\end{aligned}$$

Sean $0 \leq \alpha_1,\dots,\alpha_r \in \mathbb{R}$ y $\mathbf{x},\mathbf{y} \in \mathbb{R}^n$, $\mathbf{x} \leq \mathbf{y}$; entonces $\mathbf{A}_i(\mathbf{x}) \leq \mathbf{A}_i(\mathbf{y})$, por lo tanto $\alpha_i\mathbf{A}_i(\mathbf{x}) \leq \alpha_i\mathbf{A}(\mathbf{y})$; así que

$$\begin{aligned}(\alpha_1\mathbf{A}_1 + \cdots + \alpha_r\mathbf{A}_r)(\mathbf{x}) = \alpha_1\mathbf{A}_1(\mathbf{x}) + \cdots + \alpha_r\mathbf{A}_r(\mathbf{x}) \leq \\ \leq \alpha_1\mathbf{A}_1(\mathbf{y}) + \cdots + \alpha_r\mathbf{A}_r(\mathbf{y}) = (\alpha_1\mathbf{A}_1 + \cdots + \alpha_r\mathbf{A}_r)(\mathbf{y});\end{aligned}$$

en consecuencia $(\alpha_1,\dots,\alpha_r) \in \mathcal{I}(\mathbf{A}_1,\dots,\mathbf{A}_r)$. El resto es inmediato.

Proposición 4.3.3. *Sean $\mathbf{A}_1,\dots,\mathbf{A}_r\colon \mathbb{R}^n \to \mathbb{R}$ funciones crecientes idempotentes y $\alpha_1,\dots,\alpha_r \in \mathbb{R}$; entonces $\alpha_1\mathbf{A}_1 + \cdots + \alpha_r\mathbf{A}_r$ es idempotente si y solo si $\alpha_1 + \cdots + \alpha_r = 1$.*

Prueba. Si $\mathbf{A} := \alpha_1\mathbf{A}_1 + \cdots + \alpha_r\mathbf{A}_r$ es idempotente, $1 = \mathbf{A}(\mathbf{1}) = \alpha_1 + \cdots + \alpha_r$.
Ahora, si $\alpha_1 + \cdots + \alpha_r = 1$, sea $x \in \mathbb{R}$; entonces $\mathbf{A}(x\,\mathbf{1}) = \alpha_1\mathbf{A}_1(x\,\mathbf{1}) + \cdots + \alpha_r\mathbf{A}_r(x\,\mathbf{1}) = \alpha_1 x + \cdots + \alpha_r x = (\alpha_1 + \cdots + \alpha_r)x = x$.

Lemma 4.3.4. *Si $\mathbf{A}_1,\dots,\mathbf{A}_r\colon \mathbb{R}^n \to \mathbb{R}$ son funciones tales que $\mathbf{A}_i(\mathbf{0}) = 0$ y $\mathbf{A}_i(\mathbf{1}) = 1$, $i = 1,\dots,r$, y $\alpha_1,\dots,\alpha_r \in \mathbb{R}$ son tales que $\alpha_1\mathbf{A}_1 + \cdots + \alpha_r\mathbf{A}_r$ es creciente, entonces $\alpha_1 + \cdots + \alpha_r \geq 0$*

Prueba. Con $\mathbf{A} = \alpha_1\mathbf{A}_1 + \cdots + \alpha_r\mathbf{A}_r$, $0 = \mathbf{A}(\mathbf{0}) \leq \mathbf{A}(\mathbf{1}) = \alpha_1 + \cdots + \alpha_r$.
Pueden aparecer coeficientes negativos entre los elementos de $\mathcal{I}(\mathbf{A}_1,\dots,\mathbf{A}_r)$:

Observación 4.3.5. Sean $(\alpha_1,\dots,\alpha_r) \in \mathcal{I}(\mathbf{A}_1,\dots,\mathbf{A}_r)$, donde $\mathbf{A}_i\colon \mathbb{R}^n \to \mathbb{R}$ es una función creciente, $i = 1,\dots,r$, y $\mathbf{A} = \alpha_1\mathbf{A}_1 + \cdots + \alpha_r\mathbf{A}_r$; si $\alpha_i \neq 0$ tenemos que

$$\mathbf{A}_i = \frac{1}{\alpha_i}\mathbf{A} - \frac{\alpha_1}{\alpha_i}\mathbf{A}_1 - \dots\widehat{i}\dots - \frac{\alpha_r}{\alpha_i}\mathbf{A}_r,$$

por lo tanto $\left(\frac{1}{\alpha_i}, -\frac{\alpha_1}{\alpha_i}, .\overset{\widehat{i}}{.}., -\frac{\alpha_r}{\alpha_i}\right) \in \mathcal{I}(\mathbf{A}, \mathbf{A}_1, .\overset{\widehat{i}}{.}., \mathbf{A}_r)$.

Corolario 4.3.6. *Dadas las funciones crecientes* $\mathbf{A}, \mathbf{A}_1, \ldots, \mathbf{A}_r \colon \mathbb{R}^n \to \mathbb{R}$, *para cada* $0 \leq \alpha_1, \ldots, \alpha_r \in \mathbb{R}$ *existe una función* $\mathbf{B} \colon \mathbb{R}^n \to \mathbb{R}$ *tal que* $\mathbf{A} = \mathbf{B} - \alpha_1\mathbf{A}_1 - \cdots - \alpha_r\mathbf{A}_r$. □

Prueba. Por 4.3.2 $\mathbf{B} = \mathbf{A} + \alpha_1\mathbf{A}_1 + \cdots + \alpha_r\mathbf{A}_r$ es una función creciente.

Sin embargo, podría ser que no se permitieran elementos negativos para algún $\mathcal{I}(\mathbf{A}_1, \ldots, \mathbf{A}_r)$ concreto. Este es siempre el caso si $r = 1$:

Proposición 4.3.7. *Sea* $\mathbf{A} \colon \mathbb{R}^n \to \mathbb{R}$ *una función creciente no-constante. Sea* $\alpha \in \mathbb{R}$. *Entonces* $\alpha\mathbf{A}$ *es una función creciente si y solo si* $\alpha \geq 0$.

Prueba. Por hipótesis, existen $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ tales que $\mathbf{A}(\mathbf{x}) < \mathbf{A}(\mathbf{y})$. Tomamos $\mathbf{z} \in \mathbb{R}^n$ tal que $\mathbf{z} < \mathbf{x}$ y $\mathbf{z} < \mathbf{y}$. Entonces tenemos $\mathbf{z} < \mathbf{y}$ y $\mathbf{A}(\mathbf{z}) < \mathbf{A}(\mathbf{y})$. Como $\alpha < 0$, entonces $\alpha\mathbf{A}(\mathbf{z}) > \alpha\mathbf{A}(\mathbf{y})$ y $\alpha\mathbf{A}$ no es creciente.

Observe que si $\mathbf{A} \colon \mathbb{R}^n \to \mathbb{R}$ es una función constante, tenemos que $\alpha\mathbf{A}$ también es constante para todo $\alpha \in \mathbb{R}$.

Observación 4.3.8. Tenemos el caso trivial de considerar $\mathcal{I}(\mathbf{A}, \mathbf{A})$ para una función creciente $\mathbf{A} \colon \mathbb{R}^n \to \mathbb{R}$: si $\mathbf{A}$ es constante, entonces $\mathcal{I}(\mathbf{A}, \mathbf{A}) = \mathbb{R}^2$, y si $\mathbf{A}$ no es constante, entonces $\mathcal{I}(\mathbf{A}, \mathbf{A}) = \{(\alpha, \beta) \in \mathbb{R}^2 \mid \alpha + \beta \geq 0\}$.

Lemma 4.3.9. *Sea* $2 \leq r \in \mathbb{N}$. *Para* $i = 1, \ldots, r$, *sean* $\mathbf{A}_i \colon \mathbb{R}^n \to \mathbb{R}$ *funciones crecientes y* $\alpha_i \in \mathbb{R}$ *tal que* $\alpha_1\mathbf{A}_1 + \cdots + \alpha_r\mathbf{A}_r$ *es también una función creciente. Si existen* $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, $\mathbf{x} < \mathbf{y}$, *tales que*

$$\mathbf{A}_r(\mathbf{x}) < \mathbf{A}_r(\mathbf{y}) \text{ and } \mathbf{A}_i(\mathbf{x}) = \mathbf{A}_i(\mathbf{y}),\ i = 1, \ldots, r-1,$$

entonces $\alpha_r \geq 0$.

Prueba. $(\alpha_1\mathbf{A}_1 + \cdots + \alpha_r\mathbf{A}_r)(\mathbf{x}) \leq (\alpha_1\mathbf{A}_1 + \cdots + \alpha_r\mathbf{A}_r)(\mathbf{y})$ por hipótesis, por lo tanto

$$\alpha_r(\mathbf{A}_r(\mathbf{x}) - \mathbf{A}_r(\mathbf{y})) \leq \alpha_1(\mathbf{A}_1(\mathbf{y}) - \mathbf{A}_1(\mathbf{x})) + \cdots + \alpha_{r-1}(\mathbf{A}_{r-1}(\mathbf{y}) - \mathbf{A}_{r-1}(\mathbf{x})),$$

de donde se cumple la tesis.

Lemma 4.3.10. *Sean* $\mathbf{x} = (x_1, \ldots, x_n)$, $\mathbf{y} = (y_1, \ldots, y_n) \in \mathbb{R}^n$ *tales que* $\mathbf{x} \leq \mathbf{y}$ *y sean* $\sigma, \tau \in \Sigma_n$ *que cumplen* $\sigma \in \mathbf{x}_{(\nearrow)}$ *y* $\rho \in \mathbf{y}_{(\nearrow)}$. *Entonces* $\mathbf{x}_\sigma \leq \mathbf{y}_\tau$.

Prueba. Asumimos $x_i > x_{i+1}$; si $y_i > y_{i+1}$, con la transposición $\epsilon = (i\ i+1)$, tenemos $\mathbf{x}_\epsilon \leq \mathbf{y}_\epsilon$; si $y_i \leq y_{i+1}$, como $x_{i+1} < x_i \leq y_i$, tenemos $\mathbf{x}_\epsilon \leq \mathbf{y}$; iterando con este proceso obtenemos $\sigma, \rho \in \Sigma_n$ con $\mathbf{x}_\sigma \leq \mathbf{y}_\rho$ y $x_{\sigma(1)} \leq \cdots \leq x_{\sigma(n)}$. Para aliviar la notación, asumimos que $x_1 \leq \cdots \leq x_n$ y $\mathbf{x} \leq \mathbf{y}$. Ahora, si $y_i > y_{i+1}$, tenemos $x_i \leq x_{i+1} \leq y_{i+1}$ y $x_{i+1} \leq y_{i+1} < y_i$, por lo tanto, de nuevo con $\epsilon = (i\ i+1)$, tenemos $\mathbf{x} \leq \mathbf{y}_\epsilon$; reiterando con este proceso, concluimos que existen $\sigma, \rho \in \Sigma_n$ tales que $x_{\sigma(1)} \leq \cdots \leq x_{\sigma(n)}$, $y_{\rho(1)} \leq \cdots \leq y_{\rho(n)}$ y $\mathbf{x}_\sigma \leq \mathbf{y}_\rho$. Ahora, si $\sigma', \rho' \in \Sigma_n$ satisfacen $x_{\sigma'(1)} \leq \cdots \leq x_{\sigma'(n)}$ y $y_{\rho'(1)} \leq \cdots \leq y_{\rho'(n)}$, tenemos $\mathbf{x}_\sigma = \mathbf{x}_{\sigma'}$ y $\mathbf{y}_\rho = \mathbf{y}_{\rho'}$.

4.4 Combinaciones de Órdenes Estadísticos

Anotación 4.4.1. Sea $k \in N$, $a, b \in \mathbb{R}$, $a \leq b$. Fijamos

$$(a, b \mid k+1) = a(\mathbf{e}_1 + \cdots + \mathbf{e}_k) + b(\mathbf{e}_{k+1} + \cdots + \mathbf{e}_n)$$

si $k \leq n$ y $(a, b \mid n) = a(\mathbf{e}_1 + \cdots + \mathbf{e}_n)$.

Proposición 4.4.2. *Sean $i_1, \dots, i_r \in N$, $i_1 < \cdots < i_r$. Entonces*

$$\mathcal{I}(\mathbf{OS}_{i_1}, \dots, \mathbf{OS}_{i_r}) = \{(\alpha_1, \dots, \alpha_r) \mid \alpha_1, \dots, \alpha_r \geq 0\}$$

Prueba. Con Prop. 4.3.2 es suficiente demostrar que si $\alpha_1\mathbf{OS}_{i_1} + \cdots + \alpha_r\mathbf{OS}_{i_r}$ es creciente para $\alpha_1, \dots, \alpha_r \in \mathbb{R}$, entonces $\alpha_1, \dots, \alpha_r \geq 0$.

Podemos asumir que $r > 1$. Sea $j \in \{1, \dots, r\}$ y consideremos i_j.

Tomemos $\mathbf{x} = (0, 1 \mid i_j + 1)$, $\mathbf{y} = (0, 1 \mid i_j)$ Tenemos $\mathbf{OS}_{i_k}(\mathbf{x}) = \mathbf{OS}_{i_k}(\mathbf{y}) = 0$ if $k < j$ y $\mathbf{OS}_{i_k}(\mathbf{x}) = \mathbf{OS}_{i_k}(\mathbf{y}) = 1$ if $k > j$. Por otro lado, $\mathbf{OS}_{i_j}(\mathbf{x}) = 0 < 1 = \mathbf{OS}_{i_j}(\mathbf{y})$. Por el Lemma 4.3.9 tenemos $\alpha_j \geq 0$.

Fijamos **AM** para la media aritmética, esto es la función $\mathbf{AM}\colon \mathbb{R}^n \to \mathbb{R}$ dada por $\mathbf{AM}(\mathbf{x}) = \frac{x_1 + \cdots + x_n}{n}$ si $\mathbf{x} = (x_1, \dots, x_n) \in \mathbb{R}^n$. Como por definición $\mathbf{AM} = \frac{1}{n}(\mathbf{OS}_1 + \cdots + \mathbf{OS}_n)$, tenemos

Proposición 4.4.3. *Sean $i_1, \dots, i_r \in N$, $i_1 < \cdots < i_r$, $r < n$. Entonces*

$$\mathcal{I}(\mathbf{AM}, \mathbf{OS}_{i_1}, \dots, \mathbf{OS}_{i_r}) = \{(\alpha, \beta_1, \dots, \beta_r) \mid \alpha, \alpha + n\beta_1, \dots, \alpha + n\beta_r \geq 0\}.$$

Prueba. Asumamos que $\mathbf{A} := \alpha\mathbf{AM} + \beta_1\mathbf{OS}_{i_1} + \cdots + \beta_r\mathbf{OS}_{i_r}$ es creciente. Fijemos $R = \{i_1, \dots, i_r\}$. Entonces

$$\mathbf{A} = \frac{\alpha}{n} \sum_{i \in N \setminus R} \mathbf{OS}_i + \left(\frac{\alpha}{n} + \beta_j\right) \sum_{j=1}^{r} \mathbf{OS}_{i_j},$$

luego por Prop. 4.4.2 $\mathbf{A}$ es creciente si y solo si $\frac{\alpha}{n}, \frac{\alpha}{n} + \beta_1, \dots, \frac{\alpha}{n} + \beta_r \geq 0$.

Análogamente

Proposición 4.4.4. *Tenemos*

$$\mathcal{I}(\mathbf{AM}, \mathbf{OS}_1, \dots, \mathbf{OS}_n) = \{(\alpha, \beta_1, \dots, \beta_n) \mid \alpha + n\beta_1, \dots, \alpha + n\beta_n \geq 0\}. \quad \square$$

4.5 Combinaciones con Integral de Sugeno

Escribimos $\nu_S = \nu(S)$ para $S \subseteq N$, $\mathbf{x}_\sigma = (x_{\sigma(1)}, \dots, x_{\sigma(n)})$, $\nu_i^\sigma = \nu(N_i^\sigma)$, $N_i = \{i, \dots, n\}$ y $\nu_i = \nu(N_i)$. So $\mathbf{S}_\nu(\mathbf{x}) = \bigvee_{i=1}^n (x_{\sigma(i)} \wedge \nu_i^\sigma)$. Sabemos que

$$S_\nu(x_1, \dots, x_n) = \bigvee_{S \subseteq N} [\nu_S \wedge (\bigwedge_{i \in S} x_i)].$$

Lemma 4.5.1. *Si $S \subseteq N$, entonces existe $\sigma \in \Sigma_n$, $i \in N$ tal que $N_i^\sigma = S$ (y por lo tanto, para un medida difusa $\nu\colon 2^N \to [0, +\infty)$, $\nu_i^\sigma = \nu_S$.).*

Prueba. Si $S = N$, tomamos $i = 1$ y σ la identidad. Asumamos que $S \subset N$. Sea $S = \{j_1, \dots, j_r\}$, $|S| = r$, y $N \setminus S = \{k_1, \dots, k_{n-r}\}$. Consideremos la permutación

$$\sigma = \begin{pmatrix} 1 & \dots & n-r & n-r+1 & \dots & n \\ k_1 & \dots & k_{n-r} & j_1 & \dots & j_r \end{pmatrix}.$$

Fijemos $i = n - r + 1$. Entonces $N_i^\sigma = S$.

Lemma 4.5.2. *Let $i_1, \dots, i_r \in N$, $i_1 < \cdots < i_r$ and $\alpha_1, \dots, \alpha_r, \beta \in \mathbb{R}$. If $\alpha_1\mathbf{OS}_{i_1} + \cdots + \alpha_r\mathbf{OS}_{i_r} + \beta\mathbf{S}_\nu$ is creciente, then $\alpha_1, \dots, \alpha_r \geq 0$.*

Prueba. Sea $j \in \{1,\dots,r\}$. Sean $a,b \in \mathbb{R}$ tales que $\nu_1 < a < b$ y tomemos $\mathbf{x} = (a,b \mid i_j+1), \mathbf{y} = (a,b \mid i_j)$. Entonces $\mathbf{OS}_{i_s}(\mathbf{x}) = \mathbf{OS}_{i_s}(\mathbf{y})$ para todo $s \in \{1,\dots,\hat{j},\dots,r\}$, $\mathbf{OS}_{i_j}(\mathbf{x}) = a < b = \mathbf{OS}_{i_j}(\mathbf{y})$ y $\mathbf{S}_\nu(\mathbf{x}) = \mathbf{S}_\nu(\mathbf{y}) = \nu_1$. Por el Lemma 4.3.9, $\alpha_j \geq 0$.

Definición 4.5.3. Diremos que la medida difusa $\nu\colon 2^N \to [0,+\infty)$ es *estricta en* $k \in N$ si, o bien $k = n$ o existe $\sigma \in \Sigma_n$ tal que $\nu^\sigma_k > \nu^\sigma_{k+1}$. Diremos que ν es *estricta* si es estricta en k para todo $k \in N$.

Lemma 4.5.4. *Si la medida difusa $\nu\colon 2^N \to [0,+\infty)$ es estricta en k para algún $k \in N$, entonces existen $\mathbf{x},\mathbf{y} \in \mathbb{R}^n$, $\mathbf{x} < \mathbf{y}$ tales que $\mathbf{OS}_i(\mathbf{x}) = \mathbf{OS}_i(\mathbf{y})$ para $k \neq i \in N$ y $\mathbf{OS}_k(\mathbf{x}) = \mathbf{S}_\nu(\mathbf{x}) < \mathbf{OS}_k(\mathbf{y}) = \mathbf{S}_\nu(\mathbf{y})$.*

Prueba. Sea $\sigma \in \Sigma_n$ tal que $\nu^\sigma_k > \nu^\sigma_{k+1}$ (si $k = n$ tomemos σ la identidad en N, con independencia de si ν_n es igual a 0 o no). Sea $a,b \in \mathbb{R}$, $\nu^\sigma_{k+1} \leq a < b \leq \nu^\sigma_k$ y tomemos $\mathbf{x},\mathbf{y} \in \mathbb{R}^n$ de tal forma que $\mathbf{x}_\sigma = (a,b \mid k+1)$, $y_\sigma = (a,b \mid k)$. Tenemos $\mathbf{OS}_i(\mathbf{x}) = \mathbf{OS}_i(\mathbf{y})$ si $i \neq k$ y $\mathbf{S}_\nu(\mathbf{x}) = \mathbf{OS}_k(\mathbf{x}) < \mathbf{OS}_k(\mathbf{y}) = \mathbf{S}_\nu(\mathbf{y})$.

Proposición 4.5.5. *Sean $i_1,\dots,i_r \in N$, $i_1 < \cdots < i_r$, $r < n$. Asumamos que existe $k \in N \setminus \{i_1,\dots,i_r\}$ tal que la medida difusa $\nu\colon 2^N \to [0,+\infty)$ es estricta en k. Entonces*

$$\mathcal{I}(\mathbf{OS}_{i_1},\dots,\mathbf{OS}_{i_r},\mathbf{S}_\nu) = \{(\alpha_1,\dots,\alpha_r,\beta \mid \alpha_1,\dots,\alpha_r,\beta \geq 0)\}.$$

Prueba. Por el Lemma 4.5.4 existen $\mathbf{x},\mathbf{y} \in \mathbb{R}^n$, $\mathbf{x} < \mathbf{y}$ tales que $\mathbf{OS}_{i_j}(\mathbf{x}) = \mathbf{OS}_{i_j}(\mathbf{y})$, $j = 1,\dots,r$, y $\mathbf{S}_\nu(\mathbf{x}) < \mathbf{S}_\nu(\mathbf{y})$, por lo tanto si $\alpha_1\mathbf{OS}_{i_1} + \cdots + \alpha_r\mathbf{OS}_{i_r} + \beta\mathbf{S}_\nu$ is creciente, entonces $\beta \geq 0$ por el Lemma 4.3.9. Aplicamos ahora el Lemma 4.5.2 y la Prop. 4.3.2.

Corolario 4.5.6. *Sea $i_1,\dots,i_r \in N$, $i_1 < \cdots < i_r < n$. Para cada medida difusa $\nu\colon 2^N \to [0,+\infty)$ tenemos*

$$\mathcal{I}(\mathbf{OS}_{i_1},\dots,\mathbf{OS}_{i_r},\mathbf{S}_\nu) = \{(\alpha_1,\dots,\alpha_r,\beta) \mid \alpha_1,\dots,\alpha_r,\beta \geq 0)\}.$$

Prueba. Por Prop. 4.5.5, dado que ν es estricta en n.

Lemma 4.5.7. *Sean $\mathbf{x},\mathbf{y} \in \mathbb{R}^n$, $\mathbf{x} \leq \mathbf{y}$. Fijemos $a_i = y_i - x_i$, $i = 1,\dots,n$. Entonces $\mathbf{S}_\nu(\mathbf{y}) - \mathbf{S}_\nu(\mathbf{x}) \leq a_j$ para algún $j \in \{1,\dots,n\}$.*

Prueba. Sea $\emptyset \neq S \subseteq N$, $x_k = \bigwedge_{i\in S} x_i$ y $y_m = \bigwedge_{i\in S} y_i$. Si $m = k$, entonces $(\bigwedge_{i\in S} y_i) - (\bigwedge_{i\in S} x_i) = y_k - x_k = a_k$; si $m \neq k$, $(\bigwedge_{i\in S} y_i) - (\bigwedge_{i\in S} x_i) = y_m - x_k \leq y_k - x_k = a_k$. Fijemos $k(S) = k$. De manera directa, tenemos $[\nu_S \wedge (\bigwedge_{i\in S} y_i)] - [\nu_S \wedge (\bigwedge_{i\in S} x_i)] \leq a_{k(S)}$.

Asumamos que $T,U \subseteq N$ son tales que $\mathbf{S}_\nu(\mathbf{y}) = \nu_T \wedge (\bigwedge_{i\in T} y_i)$ y $\mathbf{S}_\nu(\mathbf{x}) = \nu_U \wedge (\bigwedge_{i\in U} x_i)$. Entonces $\nu_T \wedge (\bigwedge_{i\in T} x_i) \leq \nu_U \wedge (\bigwedge_{i\in U} x_i)$ por lo tanto

$$\begin{aligned}\mathbf{S}_\nu(\mathbf{y}) - \mathbf{S}_\nu(\mathbf{x}) &= \nu_T \wedge (\bigwedge_{i\in T} y_i) - \nu_U \wedge (\bigwedge_{i\in U} x_i)\\ &\leq \nu_T \wedge (\bigwedge_{i\in T} y_i) - \nu_T \wedge (\bigwedge_{i\in T} x_i) \leq a_{k(T)}.\end{aligned}$$

Proposición 4.5.8. *Sea $\nu\colon 2^N \to [0,+\infty)$ una medida difusa. Si*

$$\alpha_1,\dots,\alpha_n,\alpha_1+\beta,\dots,\alpha_n+\beta \geq 0,$$

entonces $\alpha_1\mathbf{OS}_1 + \cdots + \alpha_n\mathbf{OS}_n + \beta\mathbf{S}_\nu$ es creciente. Si ν es estricta en $k \in N$, entonces $\alpha_k + \beta \geq 0$; por lo tanto si ν es estricta,

$$\mathcal{I}(\mathbf{OS}_1,\dots,\mathbf{OS}_n,\mathbf{S}_\nu) = \{(\alpha_1,\dots,\alpha_n,\beta) \mid \alpha_1,\dots,\alpha_n,\alpha_1+\beta,\dots,\alpha_n+\beta \geq 0\}.$$

Prueba. Fijemos $\mathbf{A} = \alpha_1\mathbf{OS}_1 + \cdots + \alpha_n\mathbf{OS}_n + \beta\mathbf{S}_\nu$.

Asumamos que $\alpha_1, \ldots, \alpha_n, \alpha_1 + \beta, \ldots, \alpha_n + \beta \geq 0$.Sean $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, $\mathbf{x} \leq \mathbf{y}$ y $\sigma \in \mathbf{x}_{(\nearrow)}$, $\rho \in \mathbf{y}_{(\nearrow)}$. Fijemos $a_i = y_{\rho(i)} - x_{\sigma(i)}$, $1 \leq i \leq r$ (tengamos en cuenta el Lemma 4.3.10). Entonces

$$\mathbf{A}(\mathbf{y}) - \mathbf{A}(\mathbf{x}) = \alpha_1 a_1 + \cdots + \alpha_n a_n + \beta(\mathbf{S}_\nu(\mathbf{y}) - \mathbf{S}_\nu(\mathbf{x})).$$

Veamos que $\mathbf{A}(\mathbf{y}) - \mathbf{A}(\mathbf{x}) \geq 0$. Si $\beta \geq 0$ resulta obvio. Asumamos que $\beta < 0$. Por el Lemma 4.5.7 existe un $j \in N$ tal que $\mathbf{S}_\nu(\mathbf{y}) - \mathbf{S}_\nu(\mathbf{x}) \leq a_j$, por lo tanto $\beta(\mathbf{S}_\nu(\mathbf{y}) - \mathbf{S}_\nu(\mathbf{x})) \geq \beta a_j$ y por consiguiente

$$\mathbf{A}(\mathbf{y}) - \mathbf{A}(\mathbf{x}) \geq \alpha_1 a_1 + \cdots + (\alpha_j + \beta)a_j + \cdots + \alpha_n a_n \geq 0.$$

Así que $\mathbf{A}$ es creciente.

Asumamos que este es el caso. Por el Lemma 4.5.2, $\alpha_1, \ldots, \alpha_n \geq 0$.

Supongamos que ν es estricta $k \in N$. Por el Lemma 4.5.4, existen $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, $\mathbf{x} < \mathbf{y}$ tales que $\mathbf{OS}_i(\mathbf{x}) = \mathbf{OS}_i(\mathbf{y})$, $i \neq k$, y $a := \mathbf{S}_\nu(\mathbf{x}) = \mathbf{OS}_k(\mathbf{x}) < b := \mathbf{S}_\nu(\mathbf{y}) = \mathbf{OS}_k(\mathbf{y})$. Como $\mathbf{A}$ es creciente, $0 \leq \mathbf{A}(\mathbf{y}) - \mathbf{A}(\mathbf{x}) = (\alpha_k + \beta)(b - a)$. Así, $\alpha_k + \beta \geq 0$, $k = 1 \ldots, n$.

Proposición 4.5.9. *Sea $\nu\colon 2^N \to [0, +\infty)$ una medida difusa. Tenemos*

$$\mathcal{I}(\mathbf{AM}, \mathbf{S}_\nu) = \{(\alpha, \beta) \in \mathbb{R}^2 \mid \alpha, \alpha + n\beta \geq 0\}.$$

Prueba. Fijemos $\mathbf{A} = \alpha\mathbf{AM} + \beta\mathbf{S}_\nu$; así que $\mathbf{A} = \frac{\alpha}{n}(\mathbf{OS}_1 + \cdots + \mathbf{OS}_n) + \beta\mathbf{S}_\nu$, y la thesis se cumple por la Prop. 4.5.8, dado que ν es estricta en n.

Sección 5

Resultados experimentales

5.1 Combinación de funciones crecientes en la fase de pooling

Para comprobar el impacto de la función empleada en la fase de Pooling de una red convolucional, hemos probado una serie de arquitecturas diferentes y medido su tasa de acierto en un problema de clasificación de imagen, sobre el dataset CIFAR10 [42], estándar en la literatura. Este dataset está compuesto por 50000 imágenes de entrenamiento y 10000 de test, de 32x32 píxeles a color, clasificadas en 10 clases distintas. Existe una variante del mismo dataset, el CIFAR100, que únicamente se diferencia en la clasificación de las imágenes, esta vez en 100 clases, más concretas que las 10 del CIFAR10.

A continuación, explicamos brevemente las arquitecturas empleadas, cuyas características e hiperpárametros están especificados en sus respectivas tablas.

5.1.1 Arquitectura 1

Se trata de una arquitectura clásica, formada por dos bloques de convolución y pooling, seguidos de un perceptrón de 3 capas que hace las veces de clasificador final de la red. Como función de activación para las capas intermedias utilizamos la función ReLU, y como activación de la capa de salida, la función softmax, que devuelve como resultado un vector con las probabilidades de pertenencia de la imagen a cada una de las clases posibles. Como función de coste hemos empleado la entropía cruzada.

Además, para agilizar el aprendizaje de la red, y a modo de regularización, hemos añadido una capa de Batch Normalization tras cada capa de pooling. La arquitectura está resumida en la Tabla 5.1.

5.1.2 Arquitectura 2

Esta arquitectura añade unas cuantas novedades con respecto a la primera, que se traducen en una mejora sustancial de la clasificación de la red. La arquitectura es idéntica a la presentada en [43], y la hemos escogido por el parecido en el acercamiento al problema a tratar.

El extractor de características está compuesto por 3 bloques con la siguiente forma: En primer lugar, aplicamos una convolución clásica de tamaño 3x3, seguida de una capa “mlpconv”, también de tamaño 3x3. Las capas de pooling están situadas tras las dos primeras capas “mlpconv” y utilizan filtros de tamaño 3x3, con desplazamientos de la ventana de 2 píxeles. Además, tras cada capa de convolución estándar hemos añadido una capa de supervisión interna.

Tipo de capa	Tamaño de salida	Kernel
Conv1	32x32x64	3x3
ReLU	32x32x64	-
Pool1	16x16x64	2x2
BatchNorm1	16x16x64	-
Conv2	16x16x64	3x3
ReLU	16x16x64	-
Pool2	8x8x64	2x2
BatchNorm2	8x8x64	-
Flatten	4096	-
MLP1	384	-
MLP2	192	-
MLP3	10	-

TABLE 5.1: Descripción de la arquitectura 1.

Por último, esta red carece de un clasificador al uso para generar su predicción. En su lugar, la última capa de la red es otra "mlpconv" con tantos filtros como clases posibles, lo que supone que la salida de dicha capa tendrá tantos canales como clases. A esta salida se le aplica una capa de pooling global promedio, generando como salida un vector con tantos elementos como canales, o clases. A este vector posteriormente se le aplica la función softmax, para recuperar el vector de probabilidades que se utiliza como predicción de la red.

Los detalles de la arquitectura se reflejan en la Tabla 5.2

5.1.3 Arquitectura 3

La última arquitectura empleada es una DenseNet compuesta por los bloques que describimos a continuación. La red cuenta con tres bloques densos, cada uno de ellos con 16 capas de convolución, seguidas de una capa de Dropout con probabilidad 0,2. Además, hemos establecido la tasa de crecimiento k a 12, lo que significa que cada una de estas capas tendrá 12 filtros.

Utilizamos, además, capas de convolución cuello de botella, lo que implica añadir otras de estas 16 capas a cada uno de los bloques densos. Hemos fijado el número de imágenes de características generadas por estas capas a $4k = 64$ capas. Esto significa, a su vez, que cada uno de los 12 filtros, de cada una de las 16 capas de convolución de cada bloque, tendrá $4k$ canales.

Por último, adoptamos la política de compresión de la red, lo que supone que en cada capa de transición se comprime el conocimiento global de la red a la mitad.

La arquitectura se concreta en la Tabla 5.3.

5.2 Experimentación

5.2.1 Detalles de implementación

Los modelos que hemos entrenado para llevar a cabo la experimentación, han sido implementados en Python, haciendo uso de la librería PyTorch [44], que provee un motor de diferenciación automática, así como una implementación del concepto de tensor que facilita tanto la operación entre variables, como la integración con GPUs.

A este respecto, hemos entrenado los modelos haciendo uso de una NVIDIA Quadro RTX 6000, y con la ayuda del software GNU Parallel [45] para sistemas

Tipo de capa	Tamaño de salida	Kernel
Conv1	32x32x128	3x3
ReLU	32x32x128	-
HLSupervision1	10	-
Conv2	32x32x128	3x3
ReLU	32x32x128	-
HLSupervision2	10	-
Mlpconv1	32x32x128	1x1
Pool1	16x16x128	3x3
Conv3	16x16x192	3x3
ReLU	16x16x192	-
HLSupervision3	10	-
Conv4	16x16x192	3x3
ReLU	16x16x192	-
HLSupervision4	10	-
Mlpconv2	16x16x192	1x1
Pool2	8x8x192	3x3
Conv5	8x8x256	3x3
ReLU	8x8x256	-
HLSupervision5	10	-
Conv6	8x8x256	3x3
ReLU	8x8x256	-
HLSupervision6	10	-
Mlpconv3	8x8x256	1x1
Mlpconv4	8x8x10	1x1
GlobalAvgPool	10	8x8

TABLE 5.2: Descripción de la arquitectura 2.

Tipo de capa		Tamaño de salida	Kernel
Initial block	Conv	32x32x2k	3x3
	BatchNorm	32x32x2k	-
	ReLU	32x32x2k	-
Dense block 1 (x16)	BatchNorm	32x32xG	-
	ReLU	32x32xG	-
	Conv	32x32x4k	1x1
	BatchNorm	32x32x4k	-
	ReLU	32x32x4k	-
	Conv	32x32xk	3x3
Transition block 1	BatchNorm	32x32xG	
	ReLU	32x32xG	
	Conv	32x32x$G/2$	
	Pool	16x16x$G/2$	
Dense block 2 (x16)	BatchNorm	16x16xG	-
	ReLU	16x16xG	-
	Conv	16x16x4k	1x1
	BatchNorm	16x16x4k	-
	ReLU	16x16x4k	-
	Conv	16x16xk	3x3
Transition block 2	BatchNorm	16x16xG	
	ReLU	16x16xG	
	Conv	16x16x$G/2$	
	Pool	8x8x$G/2$	
Dense block 3 (x16)	BatchNorm	8x8xG	-
	ReLU	8x8xG	-
	Conv	8x8x4k	1x1
	BatchNorm	8x8x4k	-
	ReLU	8x8x4k	-
	Conv	8x8xk	3x3
GlobalAvgPool		G	8x8
MLP		10	-

TABLE 5.3: Descripción de la arquitectura 3. k indica la tasa de crecimiento de la red, que ha sido fijada a 12. G indica el número de canales del conocimiento global de la red, a la altura de cada capa, y aumentará en k filtros tras cada convolución dentro de un bloque denso. Hemos hecho explícita la compresión de este conocimiento global, indicando en la salida de la capa de convolución que aplica la compresión, que el número de imágenes de características pasa a ser $G/2$. No obstante, retomamos la convención G en las capas sucesivas (incluso cuando esta acabe de ser comprimida por la capa previa), por claridad de notación.

	Arquitectura 1	Arquitectura 2
Media	77.082	87.652
Máx	**77.392**	**87.854**
Mín	70.244	87.606
Mediana	70.616	87.072
Sugeno	72.47	86.786

TABLE 5.4: Tasa de acierto por arquitectura para funciones individuales.

UNIX, que facilita la paralelización de tareas y que hemos utilizado para entrenar en paralelo múltiples modelos, lo que ha agilizado mucho las ejecuciones.

5.2.2 Pruebas individuales

Inicialmente probamos el efecto de utilizar distintas funciones de agregación en las arquitecturas descritas. La Tabla 5.4 resume los resultados de aplicar dichas funciones como función única de pooling.

Como queda claro, ninguna de las funciones por sí solas logra superar el rendimiento obtenido por el pooling máximo. Además, los segundos mejores resultados se obtienen en todo momento para la media aritmética, la otra función de pooling históricamente empleada con éxito en las redes convolucionales.

No obstante, a continuación tratamos de comprobar el impacto que tiene combinar estas funciones siguiendo la teoría analizada en la sección 2, y demostramos que ciertas combinaciones pueden obtener un rendimiento superior al de las funciones individuales con un sobre coste muy pequeño.

5.2.3 Combinación de funciones

El proceso a seguir a la hora de combinar distintas funciones de agregación es muy sencillo, y queda reflejado en la Figura 5.1.

En primer lugar, escogemos n funciones crecientes y generamos, para cada capa de pooling, n coeficientes $\alpha_1, \alpha_2, ..., \alpha_n \in \mathbb{R}, \alpha_i \geq 0 \; \forall i = 1, ..., n$. A los valores de cada ventana de pooling, $X^{\alpha\beta}$ les aplicamos las n funciones escogidas, con lo que obtenemos n salidas $y_1^{\alpha\beta}, y_2^{\alpha\beta}, ..., y_n^{\alpha\beta}$. La salida combinada se calcula en base a la siguiente suma ponderada:

$$y^{\alpha\beta} = \sum_{i=1}^{n} \alpha_i y_i^{\alpha\beta} \tag{5.1}$$

El número de parámetros α a aprender es variable, y depende en última instancia de la decisión tomada por el programador. Hemos seguido el modelo presentado en [43] a la hora de probar distintos números de parámetros, lo que significa que para cada función escogida, tenemos las siguientes opciones:

- 1 único parámetro por capa
- 1 parámetro para cada región de pooling (compartido por todos los canales)
- 1 parámetro por cada canal (compartido por todas las regiones de pooling)
- 1 parámetro por cada región de pooling y canal

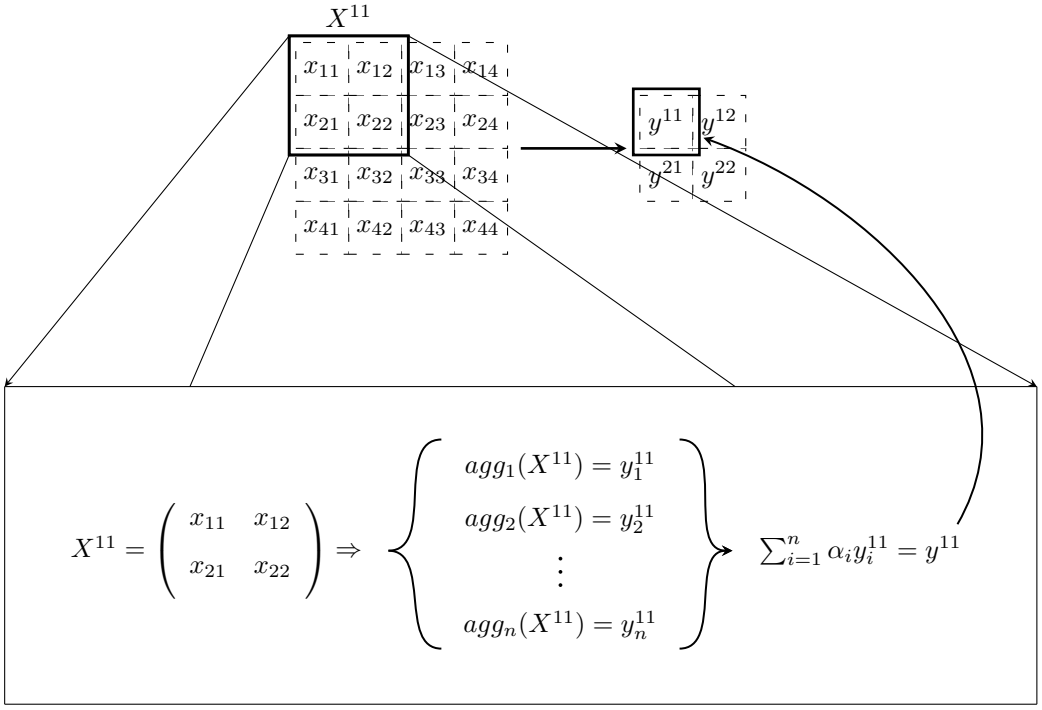

FIGURE 5.1: Ejemplo de la combinación de funciones de pooling. Para cada ventana de pooling calculamos n reducciones, que posteriormente ponderamos mediante n parámetros $\alpha \in \mathbb{R}$ positivos.

A pesar de que hemos probado varias de las estrategias anteriores, finalmente hemos optado por reportar los resultados obtenidos cuando aprendemos 1 parámetro distinto por cada canal. Las diferencias observadas en los experimentos preliminares para las distintas variantes no parecen significativas como para justificar el estudio en profundidad de todas las posibles combinaciones, en especial de aquellas que incurren en un mayor número de parámetro aprendibles, y la opción escogida parece un buen punto medio entre número de parámetros extra y capacidad de adaptación de la red.

Hemos ejecutado las combinaciones de manera gradual, en base a la teoría desarrollada en la parte 2. Para las pruebas de las dos primeros arquitecturas, y dada la naturaleza aleatoria de la inicialización de los pesos de la red, que puede impactar notablemente el aspecto final de cada modelo, hemos repetido cada prueba 5 veces, y reportamos la tasa de acierto media a lo largo de dichas ejecuciones. En el caso de la arquitectura 3, y debido al gran número de parámetros de dicho modelo y limitaciones temporales, únicamente hemos entrenado cada modelo una sola vez.

Prueba 1 - Combinación de órdenes estadísticos

Según la Proposición 4.4.2, para asegurar que una combinación $\mathcal{I}(\mathbf{OS_1}, ..., \mathbf{OS_n})$ de n órdenes estadísticos es una función creciente, basta con asegurar que $\alpha_i \geq 0 \; \forall i \in 1, ..., n$. Además, para poder asegurar que dicha función sea además idempotente, y en base a la Proposición 4.3.3, se debe cumplir que $\sum_{i=1}^{n} \alpha_i = 1$.

a) Función creciente: La Tabla 5.5 muestra los resultados obtenidos cuando la combinación resultante es creciente, aunque no necesariamente idempotente. Para asegurar que los coeficientes sean positivos, los elevamos al cuadrado. Hemos probado otras alternativas, como limitar el rango de valores de los parámetros al intervalo $(0, 1)$, tanto mediante la función logística $f(x) = \frac{1}{1+e^{-t}}$ como la función $f(x) =$

	Arquitectura 1	Arquitectura 2
Mín + Máx	**77.018**	87.416
Mín + Máx + Mediana	76.91	**87.432**

TABLE 5.5: Tasa de acierto combinando órdenes estadísticos, con $\alpha_i \geq 0 \; \forall i = 1, ..., n$, dando lugar a una función creciente.

	Arquitectura 1	Arquitectura 2
Mín + Máx	**77.174**	87.07
Mín + Máx + Mediana	76.74	**87.262**

TABLE 5.6: Tasa de acierto combinando órdenes estadísticos, con $\sum_{i=1}^{n} \alpha_i = 1$, dando lugar a una función idempotente

$max(min(x, 1), 0)$, pero estas opciones limitan el espacio de valores de manera innecesaria, y no parecen ofrecer mejores resultados. El estudio de otras alternativas se deja para estudios posteriores.

Como se puede apreciar, la diferencia en cuanto a los resultados obtenidos al añadir la mediana como combinación extra apenas difieren, aunque sí que parecen mejorar ligeramente al trabajar con la segunda arquitectura. No obstante, ninguna de las opciones obtiene un mejor resultado que el pooling máximo en el caso individual.

b) Función idempotente: Se repite el comportamiento con respecto a las mejoras a la hora de añadir la mediana, aunque la precisión en la mayoría de casos es algo inferior al caso no idempotente. No obstante, las diferencias son tan pequeñas que es difícil afirmar que la idempotencia no sea una característica deseable a la hora de agregar los resultados, pese a que no parece indispensable.

Prueba 2 - Combinación de órdenes estadísticos y media aritmética

Según la Proposición 4.4.3, añadir a una combinación de n órdenes estadísticos la media aritmética, da como resultado una función creciente siempre y cuando $\frac{\alpha}{n} \geq 0$ y $\frac{\alpha}{n} + \beta_i \geq 0 \; \forall i \in 1, ..., n$, siendo α el coeficiente asociado a la media aritmética, y β_i el coeficiente asociado al i-ésimo estadístico de orden. Por lo tanto, basta con elevar al cuadrado todos los coeficientes para que la función resultante siga siendo una función creciente.

En la Tabla 5.7 se reflejan los resultados obtenidos cuando combinamos distintos estadísticos de orden con la media aritmética de la forma descrita. Como cabría esperar, para ambas arquitecturas los mejores resultados se obtienen al combinar la media aritmética y el máximo, aproximándose mucho a los resultados ofrecidos exclusivamente por el máximo.

	Arquitectura 1	Arquitectura 2
Media aritmética + Mín	75.042	87.23
Media aritmética + Máx	**77.23**	**87.778**
Media aritmética + Mín + Máx	77.166	87.252
Media aritmética + Mín + Máx + Mediana	77.044	87.566

TABLE 5.7: Tasa de acierto combinando la media aritmética y ciertos órdenes estadísticos

	Arquitectura 1	Arquitectura 2
Mín + Sugeno	72.408	87.458
Máx + Sugeno	77.09	**87.602**
Mín + Máx + Sugeno	**77.298**	87.014
Mín + Máx + Mediana + Sugeno	77.028	87.294

TABLE 5.8: Tasa de acierto combinando la integral de sugeno y ciertos órdenes estadísticos

	Arquitectura 1	Arquitectura 2
Media aritmética + Sugeno	76.932	**88.228**

TABLE 5.9: Tasa de acierto combinando la media aritmética y la integral de Sugeno

Prueba 3: Órdenes estadísticos y Sugeno

Por la Proposición 4.5.5, tenemos que, si añadimos a una colección de órdenes estadísticos la integral de sugeno, la función resultante sigue siendo creciente siempre y cuando todos los coeficientes empleados sean positivos. Una vez más, elevamos al cuadrado los coeficientes para asegurar que sea el caso. Los resultados pueden consultarse en la Tabla 5.8. A pesar de los malos resultados individuales obtenidos por la integral de sugeno, es curioso observar como al combinarla con otras funciones los resultados mejoran notablemente. Nuestra hipótesis es que las activaciones generadas por la integral de sugeno son lo suficientemente distintas a las obtenidas por el máximo o la media y ofrecen información que se pierde en el resto de casos.

Prueba 4: Media aritmética y Sugeno

La Proposición 4.5.9 indica que la combinación de la media aritmética y la integral de sugeno es una función creciente, siempre y cuando $\alpha \geq 0$ y $\alpha + n\beta \geq 0$, siendo α el coeficiente de la media aritmética, β el coeficiente de la integral de sugeno, y n el número de elementos a reducir. Por tanto, de nuevo es suficiente con elevar al cuadrado ambos coeficientes para asegurar que la combinación resultante sea creciente.

Los resultados obtenidos para esta función están representados en la Tabla 5.9. Curiosamente, para la arquitectura 2 es esta combinación la que obtiene la mejor tasa de acierto medio, lo que parece respaldar la idea de que la integral de Sugeno preserva información que no se mantiene con ninguna de las demás funciones de agregación.

5.2.4 Test de robustez

Con el fin de estudiar más en profundidad la respuesta ofrecida por los mejores modelos encontrados, hemos realizado unos test de robustez similares a los propuestos en [43]. Ante la falta de una definición estándar del concepto de robustez para el procesamiento de imagen [46], parece necesario indicar que por robustez entendemos en este caso la capacidad de un modelo para predecir correctamente ejemplos de entrada que presenten ruido o provengan de una distribución estadística ligeramente distinta de aquella en base a la cual fue entrenado.

En concreto, hemos analizado la respuesta del modelo al aplicar tres tipos de transformaciones a las imágenes de entrada: rotación, escala y cambio del brillo.

	Arquitectura 3
Media	**89.25**
Mediana	88.76
Máx	87.99
Mín	88.28
Media + Máx	86.99
Media + Sugeno	86.38
Máx + Sugeno	87.79
Mín + Máx + Media	88.52

TABLE 5.10: Tasa de acierto para distintas pruebas utilizando la arquitectura 3

Para ello, hemos tomado el conjunto de test del dataset CIFAR10 y hemos aplicado la misma transformación a todas las imágenes, reportando después la tasa de acierto obtenida al calcular la predicción para las imágenes modificadas. La Figura 5.2 muestra los distintos resultados obtenidos para la arquitectura 2 cuando empleamos la combinación de integral de Sugeno con media aritmética como pooling, y los compara con los que ofrecen los modelos que utilizan el máximo o la media individualmente.

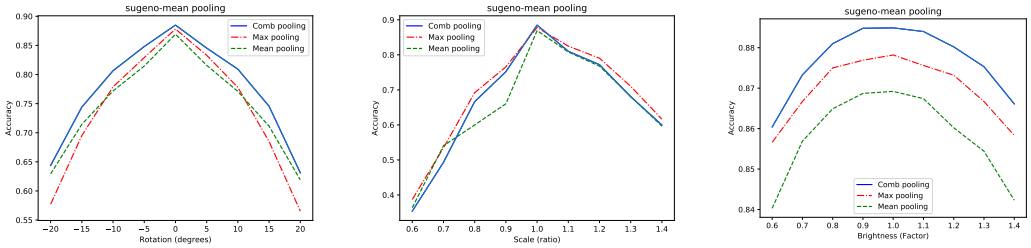

FIGURE 5.2: Respuesta a distintos tipos de ruido sobre el conjunto de test del dataset CIFAR10. El valor central del eje x representa el aspecto por defecto del dataset. Los valores de ruido aplicados son mayores cuanto más alejados se encuentran del punto central, lo que explica el aspecto de las curvas resultantes.

Como se puede observar, el modelo que utiliza una combinación de funciones supera a los pooling máximo y media tanto cuando introducimos cambios en la rotación de las imágenes como cuando modificamos su brillo. Además, en el caso de reescalar las imágenes, el desempeño, aunque algo inferior al del pooling máximo, se mantiene muy similar. Por lo tanto, vemos que la capacidad de generalización del modelo aumenta cuando utilizamos la combinación estudiada.

5.3 Pruebas arquitectura 3

Por limitaciones temporales, el estudio sobre la arquitectura 3 no ha sido tan extenso como para las otras dos arquitecturas, y por tanto, hemos centrado la experimentación en el estudio de las combinaciones que mejores clasificaciones han reportado en el resto de modelos. Los resultados se reflejan en la tabla 5.10

Como puede observarse, en este caso es la media aritmética la que consigue los mejores resultados, seguida de cerca por la mediana. En el campo de las combinaciones, es la combinación de mínimo, máximo y media la que obtiene el mejor resultado, aunque por detrás de los métodos individuales.

Para cada uno de los tres modelos presentados, hemos obtenido los mejores resultados con una función distinta. Esto parece indicar que la elección de una función de pooling no es un proceso trivial como pueda sospecharse, sino que puede resultar interesante analizar distintas opciones para una misma arquitectura dada.

5.4 Toma de decisión multimodelo

A continuación, exploramos la posibilidad de llevar a cabo un proceso de toma de decisión que agregue las predicciones obtenidas por los distintos modelos entrenados. Dada la cantidad de modelos, parece interesante dividir en dos fases el proceso: la primera se encargará de agregar, para cada arquitectura distinta, las predicciones de todos los modelos que utilizan distintas funciones de pooling, por ser las más similares entre sí; la segunda trabajará sobre las predicciones agregadas de cada arquitectura, generando el consenso final. La Figura 5.3 describe visualmente el proceso seguido.

Para ser más precisos, suponga que contamos con M arquitecturas distintas. Hemos entrenado cada arquitectura con N funciones de pooling diferentes, con lo que tenemos N variantes para cada arquitectura. Sea $\mathbf{X}$ una imagen de entrada que queremos clasificar en una de C posibles clases, fijamos la arquitectura k, y comenzamos por obtener la predicción de cada una de sus N variantes para $\mathbf{X}$. A partir de estas predicciones, generamos la matriz $\mathbf{Y}^k$, donde la fila i-ésima contiene las probabilidades de pertenencia a cada una de las C clases, devuelta por la variante i-ésima de la arquitectura k:

$$\mathbf{Y^k} = \begin{pmatrix} Y^k_{1,1} & Y^k_{1,2} & \dots & Y^k_{1,C} \\ Y^k_{2,1} & Y^k_{2,2} & \dots & Y^k_{2,C} \\ \vdots & \vdots & \ddots & \vdots \\ Y^k_{N,1} & Y^k_{N,2} & \dots & Y^k_{N,C} \end{pmatrix}$$

El objetivo entonces, es agregar las distintas columnas y obtener un vector final $\mathbf{Y}^{k*}$, con las probabilidades de que $\mathbf{X}$ pertenezca a cada una de las C clases, según el consenso entre las variantes de la arquitectura k.

Tras esta primera fase, contaremos con M vectores de probabilidad, uno por cada una de las M arquitecturas distintas. Con todos estos valores, podemos generar una nueva matriz $\mathbf{Y}$, donde la fila k-ésima representará, en esta ocasión, las probabilidades de pertenencia a cada clase, según el consenso entre las variantes de la arquitectura k. Para concluir, repetiremos la toma de decisión, agregando las columnas de $\mathbf{Y}$. El vector de probabilidades $\mathbf{Y}^*$, resultado de esta operación, será el que empleemos finalmente a la hora de clasificar el ejemplo $\mathbf{X}$.

Es importante anotar que, aunque hablamos todo el tiempo de vectores de probabilidades, dependiendo de la función de agregación escogida, es posible que los resultados de los consensos en alguna de las fases no estén normalizados. En dicho caso, normalizaremos estos vectores utilizando la función exponencial normalizada.

A la hora de agregar los resultados obtenidos, tanto en la primera como en la segunda fases de la toma de decisión, hemos probado las siguientes funciones de agregación:

- Media aritmética
- Mediana
- Mínimo
- Máximo
- Integral de Sugeno
- Integral de Sugeno (reemplazando el producto por la t-norma de Hamacker)
- Integral Choquet
- Integral CF12 (usando el mínimo)
- Integral CF12 (usando la raíz cuadrada y t-norma de Lukasiewicz)
- Integral CF (escogiendo la t-norma de Hamacher como función F)
- OWA1 (a=0.1, b=0.5)
- OWA2 (a=0.5, b=1)
- OWA3 (a=0.3, b=0.8)
- Media geométrica
- Media harmónica
- Overlap del seno

Puede consultarse más información acerca de dichas funciones en citebeliakov2016averaging.

La Figura 5.4 representa un ejemplo sencillo para un proceso de toma de decisión en el que utilizamos el error cuadrático medio como función de consenso.

Aunque hemos considerado la posibilidad de utilizar combinaciones de estas funciones, de modo similar al empleado en las capas de pooling, hemos descartado la idea dada la dificultad de encontrar los mejores valores para los coeficientes de cada función. Esto no resultaba un problema en las funciones de pooling dado que los valores se aprendían como otro parámetro más de la red, mediante el mismo de aprendizaje utilizado para ajustar los pesos de las capas de convolución.

5.4.1 Modelos individuales

En esta sección reportamos los resultados obtenidos al llevar a cabo únicamente el primero de los pasos descritos anteriormente. Es decir, realizamos una toma de decisión entre los modelos con arquitectura común, pero distinta función de pooling.

Los resultados obtenidos para cada arquitectura quedan reflejados en las Tablas 5.11–5.13. Como se puede observar, independientemente de la función de agregación escogida, todos los resultados mejoran los obtenidos por el total de modelos individuales, lo que parece justificar la utilización de esta estrategia. En concreto, son la media geométrica y el overlap de la función seno las que consiguen los mejores resultados, tanto para la arquitectura 2 como la 3.

Es importante tener en cuenta que el método escogido ha sido el de agregar todas las redes entrenadas sin ninguna fase de selección anterior, por lo que probablemente los resultados podrían mejorar si filtráramos aquellos modelos que aportan información redundante al “ensemble".

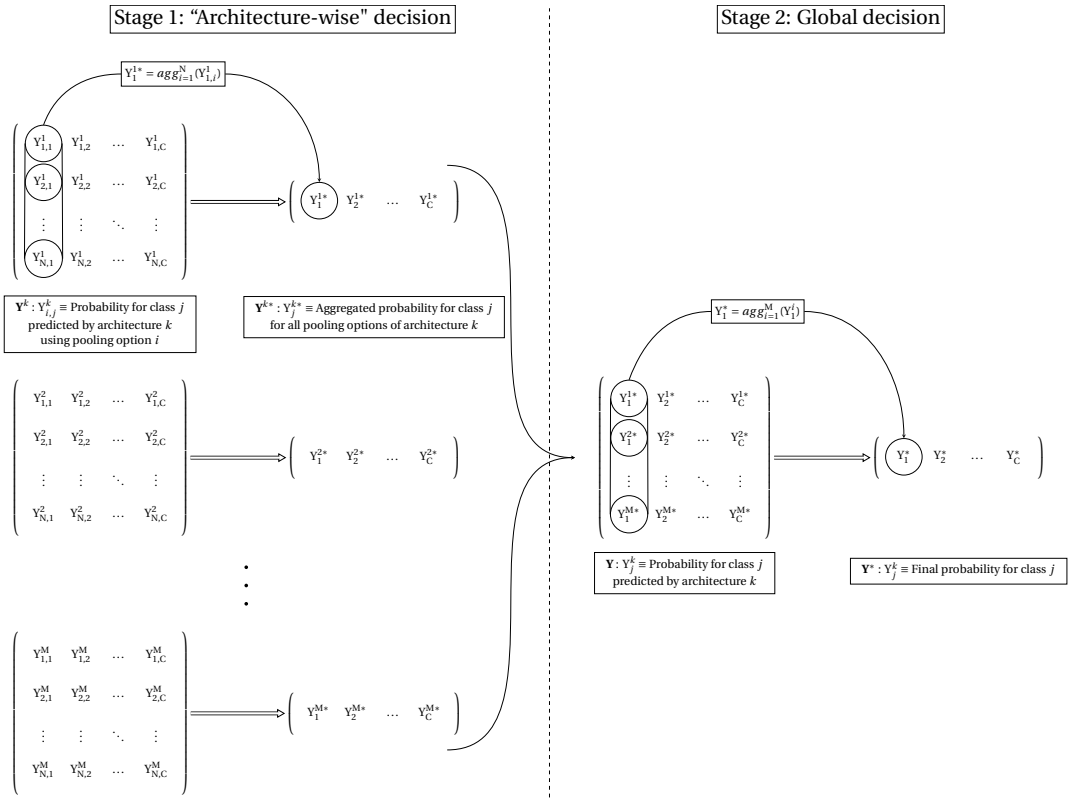

FIGURE 5.3: Representación gráfica del proceso de toma de decisión, dividido en dos fases, para un problema de clasificación con C clases posibles. Contamos con M arquitecturas distintas, para cada una de las cuales hemos entrenado N variantes, diferenciadas exclusivamente por la función de pooling empleada. A la izquierda de la línea moteada mostramos la toma de decisión de los modelos que comparten la misma arquitectura, pero para los que hemos escogido una función de pooling distinta. A la derecha de la línea mostramos la toma de decisión final, que agrega los resultados generados por esta primera fase. **Nota**: Dependiendo de la función empleada, puede ser necesario normalizar el vector obtenido como resultado en cualquiera de las dos fases, de modo que siga pudiendo interpretarse como un vector de probabilidades.

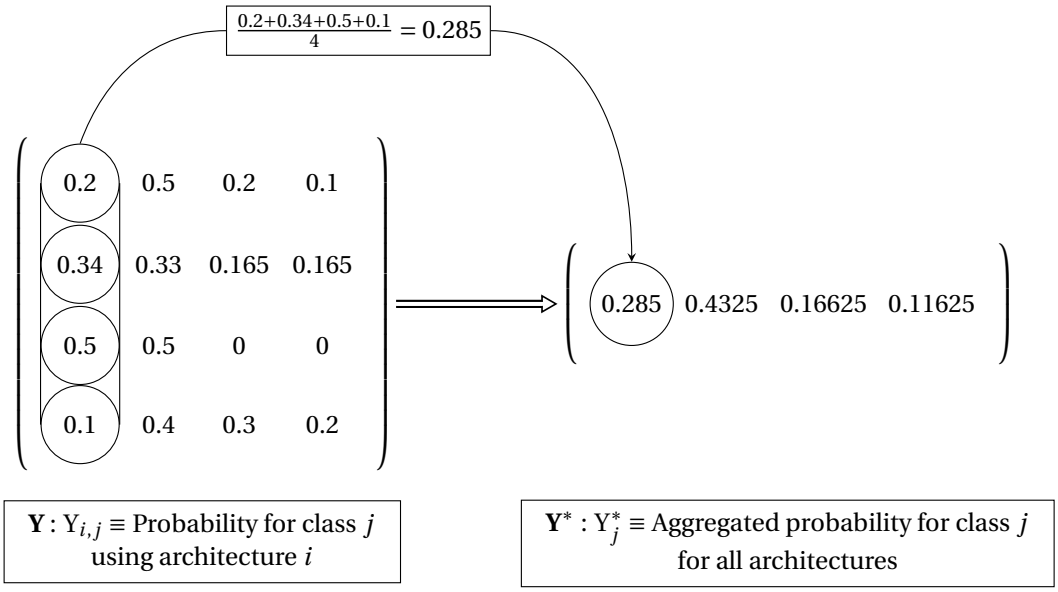

FIGURE 5.4: Ejemplo sencillo en el que se ejemplifica el proceso seguido en la segunda fase del proceso de toma de decisión, cuando empleamos la media aritmética como función de agregación. En este caso existen 4 clases posibles (columnas) y 4 arquitecturas distintas (filas). Si se tratase de un ejemplo de la primera fase, las filas representarían 4 variantes de una arquitectura concreta.

Función de agregación	Tasa de acierto
Media aritmética	**80.61**
Mediana	80.60
Mínimo	75.59
Máximo	76.85
Sugeno	80.31
Sugeno (Hamacher)	80.44
Choquet	80.57
CF12 (Min + Min)	80.32
CF12 (Raíz + Lukasiewicz)	80.59
CF (Hamacher)	80.36
OWA1	80.55
OWA2	80.58
OWA3	80.52
Media geométrica	80.17
Overlap seno	80.17

TABLE 5.11: Tasa de acierto obtenida por la toma de decisión aplicada a las variantes de la arquitectura 1, en base a la función de agregación escogida.

Función de agregación	Tasa de acierto
Media aritmética	90.57
Mediana	90.47
Mínimo	90.07
Máximo	89.84
Sugeno	90.49
Sugeno (Hamacher)	90.49
Choquet	90.56
CF12 (Min + Min)	90.49
CF12 (Raíz + Lukasiewicz)	90.57
CF (Hamacher)	90.55
OWA1	90.57
OWA2	90.54
OWA3	90.58
Media geométrica	**90.69**
Overlap seno	**90.69**

TABLE 5.12: Tasa de acierto obtenida por la toma de decisión aplicada a las variantes de la arquitectura 2, en base a la función de agregación escogida.

Función de agregación	Tasa de acierto
Media aritmética	93.49
Mediana	93.39
Mínimo	92.74
Máximo	92.53
Sugeno	93.38
Sugeno (Hamacher)	93.37
Choquet	93.44
CF12 (Min + Min)	93.38
CF12 (Raíz + Lukasiewicz)	93.47
CF (Hamacher)	93.41
OWA1	93.46
OWA2	93.44
OWA3	93.36
Media geométrica	**93.65**
Overlap seno	**93.65**

TABLE 5.13: Tasa de acierto obtenida por la toma de decisión aplicada a las variantes de la arquitectura 3, en base a la función de agregación escogida.

5.5 Toma de decisión multimodelo

A continuación llevamos a cabo el proceso descrito en la Figura 5.3. Probamos a agregar todos los grupos de la primera fase con cada una de las funciones disponibles, y para cada matriz resultante de entrada de la segunda fase, repetimos el proceso y agregamos sus valores con cada una de las funciones disponibles. Esto supone que, si disponemos de n funciones de agregación distintas, el número de combinaciones a probar sea n^2, motivo por el cual no reportamos los resultados de cada una de las combinaciones utilizadas, sino únicamente el mejor.

Nótese que, aunque pueda parecer directo escoger como agregación para la primera fase aquella que reportó mejores resultados tras evaluar la clasificación de las predicciones practicada en 5.4.1, esta no tiene por qué ser la función de agregación óptima en el contexto del proceso global. De hecho, este es un hecho que hemos comprobado empíricamente a lo largo de los experimentos.

Cuando agregamos los resultados de las tres arquitecturas de este modo, los mejores resultados obtenido son de 92.46% de acierto, cuando usamos el overlap seno como agregación para la primera fase y el OWA1 para la segunda. Nótese que el valor es menor al que obtenemos si únicamente agregamos las distintas variantes de la arquitectura 3.

Esto puede verse justificado por el hecho de que la diferencia en el número de parámetros y la precisión de cada tipo de modelos es muy grande. Más concretamente, la arquitectura 1 únicamente alcanza un 80.61% de acierto tras la agregación de la fase 1, en contraste con el 90.69 y 93.65 obtenidos para las arquitecturas 2 y 3 respectivamente.

Es por esto que repetimos el proceso, ignorando en este caso los resultados ofrecidos por la arquitectura 1. Cuando analizamos la toma de decisión con las arquitecturas 2 y 3 exclusivamente, el mejor resultado conseguido para la combinación de CF12 y mínimo, con un 93.33% de acierto. Una vez más, y a pesar de que, efectivamente, omitir los resultados de la arquitectura 1 mejoran el resultado final, todo parece indicar que lo más eficiente es centrarse en el uso exclusivo de la arquitectura 3.

Esto no resulta del todo sorprendente si tenemos en cuenta el número de parámetros y capas de cada uno de los modelos, y está en la línea de lo que los últimos trabajos del campo parecen probar: que para lograr una mejor capacidad de clasificación es conveniente emplear modelos cada vez más profundos. En un futuro podría repetirse esta última experimentación para varias arquitecturas con un número de parámetros similar, para comprobar si esta estrategia de toma de decisión puede suponer un beneficio real.

Sección 6

Conclusiones

En este trabo hemos presentado los resultados obtenidos al sustituir las funciones de pooling clásicas en el proceso de una red convolucional por funciones más avanzadas, así como combinaciones crecientes de estas funciones. Los resultados muestran que, pese a que no hay una regla universal que nos permita escoger la mejor función a priori, esta sí que cambia de uno a otro modelo, no resultando siempre las más eficientes el máximo o la media aritmética.

Por tanto, la elección de una función de pooling a la hora de diseñar una nueva arquitectura debería ser un proceso que se estudiara con detenimiento, pues es sencillo mejorar el rendimiento de estos modelos con poco trabajo y sin un sobre coste computacional demasiado elevado.

Además, también hemos presentado un método de toma de decisión para agregar las salidas ofrecidas por las distintas variantes entrenadas, que ha demostrado ser una herramienta muy útil a la hora de mejorar los resultados ofrecidos por todas las arquitecturas, superándolas por varios puntos en su tasa de acierto obtenida.

No obstante, esta estrategia también ha demostrado la necesidad de combinar arquitecturas con un número de parámetros y complejidad similar, si se quiere sacar el máximo partido al ensemble de dichos métodos. La prueba de esta técnica en otras arquitecturas queda como trabajo futuro.

Del mismo modo, se experimentará con otras funciones de agregación, así como con sus respectivas combinaciones crecientes, y más aún, se tratará de comprobar si existen otras propiedades matemáticas que beneficien el desempeño de la red y que permitan facilitar la selección de la función de pooling a utilizar.

Bibliografía

[1] W. S. Mcculloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity.," *Bulletin of Mathematical Biophysics*, no. 5, pp. 115–133, 1943. DOI: 10.1007/BF02478259. [Online]. Available: https://doi.org/10.1007/BF02478259.

[2] F. Rosenblatt, "The perceptron—a perceiving and recognizing automaton," Cornell Aeronautical Laboratory, Tech. Rep., 1957.

[3] B. Widrow, "An adaptive 'adaline' neuron using chemical 'memistors'," Stanford Electronics Laboratories, Tech. Rep., 1960.

[4] M. L. Minsky and S. A. Papert, *Perceptrons*. MIT Press, 1969.

[5] *Proceedings of the Fourth National Conference on Artificial Intelligence*, 1984.

[6] P. J. Werbos, "Backwards differentiation in ad and neural nets: Past links and new opportunities," in *Automatic Differentiation: Applications, Theory, and Implementations*, Springer Berlin Heidelberg, 2006, pp. 15–34, ISBN: 978-3-540-28438-3.

[7] S. Linnainmaa, "Taylor expansion of the accumulated rounding error," *BIT Numerical Mathematics*, no. 16, pp. 146–160, 1976. DOI: 10.1007/BF01931367. [Online]. Available: https://doi.org/10.1007/BF01931367.

[8] P. Werbos and P. John, "Beyond regression: New tools for prediction and analysis in the behavioral sciences.," Jan. 1974.

[9] D. E. Rumelhart, H. G. E., and R. J. Williams, *Nature*, no. 323, pp. 533–536, 1986. DOI: 10.1038/323533a0. [Online]. Available: https://doi.org/10.1038/323533a0.

[10] D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture of monkey striate cortex," *The Journal of Physiology*, vol. 195, pp. 215–243, Mar. 1968. DOI: 10.1113/jphysiol.1968.sp008455.

[11] K. Fukushima, "Neocognitron: A hierarchical neural network capable of visual pattern recognition," *Neural Networks*, vol. 1, no. 2, pp. 119 –130, 1988, ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(88)90014-7. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0893608088900147.

[12] Y. Lecun, L. Botou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, Dec. 1998. DOI: 10.1109/5.726791. [Online]. Available: https://doi.org/10.1109/5.726791.

[13] S. Hochreiter and J. Schmidhuber, *Neural Computation*, vol. 9, pp. 1735–1780, 8 Nov. 1997.

[14] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, Ieee, 2009, pp. 248–255.

[15] H. Mark, "Many-core gpu computing with nvidia cuda," in *Proceedings of the 22nd Annual International Conference on Supercomputing*, Jun. 2008.

[16] R. Raina, A. Madhavan, and A. Y. Ng, "Large-scale deep unsupervised learning using graphics processors," in *Proceedings of the 26th Annual International Conference on Machine Learning*, 2009.

[17] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," *Neural Information Processing Systems*, vol. 25, Jan. 2012. DOI: 10.1145/3065386. [Online]. Available: https://doi.org/10.1145/3065386.

[18] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," Jun. 2010, pp. 807–814.

[19] I. Goodfellow, Y. Bengio, and C. Aaron, "Deep learning," in. MIT Press, 2016, ch. 6, p. 188.

[20] ——, "Deep learning," in. MIT Press, 2016, ch. 6, p. 191.

[21] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," vol. 15, Jan. 2010.

[22] I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, "Maxout networks," *30th International Conference on Machine Learning, ICML 2013*, vol. 1302, Feb. 2013.

[23] C. Bishop, *Pattern Recognition and Machine Learning*. Springer-Verlag New York, 2006.

[24] S. Ruder, "An overview of gradient descent optimization algorithms," Sep. 2016.

[25] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural Networks: Tricks of the Trade: Second Edition*, G. Montavon, G. B. Orr, and K.-R. Müller, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 9–48, ISBN: 978-3-642-35289-8. DOI: 10.1007/978-3-642-35289-8_3. [Online]. Available: https://doi.org/10.1007/978-3-642-35289-8_3.

[26] T. M. Heskes and B. Kappen, "On-line learning processes in artificial neural networks," in, ser. North-Holland Mathematical Library, J. Taylor, Ed., vol. 51, Elsevier, 1993, pp. 199 –233. DOI: 10.1016/S0924-6509(08)70038-2. [Online]. Available: https://doi.org/10.1016/S0924-6509(08)70038-2.

[27] H. Shimodaira, "Improving predictive inference under covariate shift by weighting the log-likelihood function," *Journal of Statistical Planning and Inference*, vol. 90, no. 2, pp. 227–244, 2000, ISSN: 0378-3758. DOI: 10.1016/S0378-3758(00)00115-4. [Online]. Available: https://doi.org/10.1016/S0378-3758(00)00115-4.

[28] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15, Lille, France: JMLR.org, 2015, 448–456.

[29] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, *How does batch normalization help optimization? (no, it is not about internal covariate shift)*, May 2018.

[30] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations*, 2015.

[31] A. Krogh and J. A. Hertz, "A simple weight decay can improve generalization," in *Advances in Neural Information Processing Systems 4*, J. E. Moody, S. J. Hanson, and R. P. Lippmann, Eds., Morgan-Kaufmann, 1992, pp. 950–957. [Online]. Available: http://papers.nips.cc/paper/563-a-simple-weight-decay-can-improve-generalization.pdf.

[32] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html.

[33] Y. Gal and Z. Ghahramani, "Bayesian convolutional neural networks with bernoulli approximate variational inference," Jun. 2015.

[34] M. Lin, Q. Chen, and S. Yan, "Network in network," Dec. 2013.

[35] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu, "Deeply-supervised nets," in *Proceedings of Machine Learning Research*, vol. 38, May 2015, pp. 562 –570.

[36] T. Evgeniou and M. Pontil, "Support vector machines: Theory and applications," in *Machine Learning and Its Applications: Advanced Lectures*. Springer-Verlag Berlin Heidelberg, Jan. 2001, ch. 11, pp. 249–257. DOI: 10.1007/3-540-44673-7_12.

[37] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 2261–2269.

[38] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[39] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Weinberger, "Deep networks with stochastic depth," vol. 9908, Oct. 2016, pp. 646–661, ISBN: 978-3-319-46492-3. DOI: 10.1007/978-3-319-46493-0_39.

[40] D. Paternain, J. Fernandez, H. Bustince, R. Mesiar, and G. Beliakov, "Construction of image reduction operators using averaging aggregation functions," *Fuzzy Sets and Systems*, vol. 261, pp. 87 –111, 2015, Theme: Aggregation operators, ISSN: 0165-0114. DOI: https://doi.org/10.1016/j.fss.2014.03.008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0165011414001122.

[41] C. Lee, P. Gallagher, and Z. Tu, "Generalizing pooling functions in cnns: Mixed, gated, and tree," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 4, pp. 863–875, 2018.

[42] A. Krizhevsky, *Learning multiple layers of features from tiny images*, 2009.

[43] C. Lee, P. Gallagher, and Z. Tu, "Generalizing pooling functions in cnns: Mixed, gated, and tree," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 4, pp. 863–875, 2018.

[44] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.

cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[45] O. Tange, "Gnu parallel - the command-line power tool," *;login: The USENIX Magazine*, vol. 36, no. 1, pp. 42–47, 2011. DOI: http://dx.doi.org/10.5281/zenodo.16303. [Online]. Available: http://www.gnu.org/s/parallel.

[46] A. Vacavant, "A novel definition of robustness for image processing algorithms," in *Reproducible Research in Pattern Recognition: First International Workshop, RRPR 2016, Cancun, Mexico, December 4, 2016, Revised Selected Papers*, Apr. 2017, pp. 75–87, ISBN: 978-3-319-56413-5.