MÁSTER UNIVERSITARIO DE INVESTIGACIÓN EN INGENIERÍA DE SOFTWARE Y SISTEMAS INFORMÁTICOS

TRABAJO FINAL DE MÁSTER

Estrategias de poda en redes neuronales convolucionales

INGENIERÍA DE SOFTWARE - 31105109

Autor:
Sabato Ceruso

Director:
Carlos Cerrada Somolinos

Junio de 2018

MÁSTER UNIVERSITARIO DE INVESTIGACIÓN EN INGENIERÍA DE SOFTWARE Y SISTEMAS INFORMÁTICOS

INGENIERÍA DE SOFTWARE - 31105109

Estrategias de poda en redes neuronales convolucionales

TRABAJO TIPO B: TRABAJO ESPECÍFICO PROPUESTO POR EL ALUMNO

Autor:
Sabato Ceruso

Director:
Carlos Cerrada Somolinos

Declaración jurada de autoría

Yo, Sabato Ceruso, con N.I.E X-8912496-L, hace constar que es el autor del trabajo: «Estrategias de poda en redes neuronales convolucionales».

En tal sentido, manifiesto la originalidad de la conceptualización del trabajo, interpretación de datos y la elaboración de las conclusiones, dejando establecido que aquellos aportes intelectuales de otros autores, se han referenciado debidamente en el texto de dicho trabajo

- Garantizo que el trabajo que remito es un documento original y no ha sido publicado, total ni parcialmente por otros autores, en soporte papel ni en formato digital
- Certifico que he contribuido directamente al contenido intelectual de este manuscrito, a la génesis y análisis de sus datos, por lo cual estoy en condiciones de hacerme públicamente responsable de él.
- No he incurrido en fraude científico, plagio o vicios de autoría; en caso contrario, aceptaré las medidas disciplinarias sancionadoras que correspondan.

Firmado:

Fecha: 6 de junio de 2018

Autorización

Autorizo a la Universidad Nacional de Educación a Distancia a difundir y utilizar, con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria de este Trabajo Fin de Máster, como el código, la documentación y/o el prototipo desarrollado.

Firmado:

Resumen

Estrategias de poda en redes neuronales convolucionales

por Sabato Ceruso

En las últimas décadas las técnicas basadas en *deep learning* han demostrado conseguir un gran desempeño en variedad de problemas, tales como la clasificación de imágenes, detección, segmentación, procesamiento de lenguaje natural, etc. Este éxito se debe al desarrollo de modelos con arquitecturas complejas capaces de afrontar eficazmente dichos problemas. Dada esta complejidad, la gran mayoría de estas estructuras son incompatibles con implementaciones en aplicaciones en tiempo real o en arquitecturas de limitados recursos tales como teléfonos móviles. Para afrontar esta problemática diversas técnicas han sido desarrolladas en los últimos años, desde métodos que modifican o comprimen los modelos ya existentes hasta nuevas estructuras y técnicas para la creación de modelos eficientes. El presente trabajo pretende realizar una revisión de estas técnicas, para luego centrarse específicamente en la técnica de poda de conexiones. Tras la investigación de los diversos métodos existentes en la literatura, se experimentará con ellos sobre un modelo específico para así analizar su comportamiento, qué características influyen en su eficacia y qué medida de calidad cabe esperar de un algoritmo de estas características. Los resultados del presente trabajo se presentan mediante los experimentos realizados, demostrando conseguir reducir exitosamente la complejidad del modelo propuesto hasta lograr un consumo de memoria 80 veces menor y una cantidad de operaciones 7 veces menor.

Palabras clave: *Deep learning*, redes neuronales convolucionales, poda de conexiones

Índice general

Índice de figuras

Índice de tablas

Abreviaciones

BN	**B**atch **N**ormalization
Conv	**C**onvolución
Deconv	**D**econvolución
ELU	**E**xponential **L**inear **U**nit
FC	**F**ully **C**onnected layer
FLOP	**Fl**oating **Op**eration
GPGPU	**G**eneral **P**urpose **G**raphics **P**rocessing **U**nit
GPU	**G**raphics **P**rocessing **U**nit
ILSVRC	**I**magenet **L**arge **S**cale **V**isual **R**econgition **C**hallenge
MACC	**M**ultiply and **Acc**umulate
ReLU	**Re**ctified **L**inear **U**nit (Rectificación lineal)
SGD	**S**tochastic **G**radient **D**escent

Lista de símbolos

$*$	Operación de convolución

Capítulo 1

Introducción y justificación

1.1. Introducción

En las últimas décadas las técnicas basadas en *deep learning* han demostrado conseguir un buen desempeño en variedad de problemas, tales como la clasificación de imágenes, detección, segmentación, procesamiento de lenguaje natural, etc. De entre los diversos tipos de redes neuronales, las redes neuronales convolucionales han sido desarrolladas más extensamente en los últimos años, siendo estas las estudiadas en el presente trabajo.

Gran parte de este desarrollo reciente se debe a la gran cantidad de datos etiquetados cada vez más fácilmente accesibles y al creciente número de competiciones que promocionan la investigación. Ejemplos de ello se tiene en [9, 10] con imágenes clasificadas por etiquetas o en *datasets* más complejos como [11, 12] que cuentan además con clasificaciones a nivel de pixel (segmentación).

Otro factor que obstaculizaba el desarrollo es la potencia de cómputo necesaria para poder entrenar incluso las redes neuronales más ligeras. Con el avance reciente de la potencia de cómputo de las GPGPU se han podido desarrollar redes neuronales convolucionales capaces de alcanzar resultados del estado del arte en varias aplicaciones diferentes. De la mano de este desarrollo, con el fin de facilitar la implementación han surgido diversos *frameworks* como *Caffe* [13], *Tensorflow* [14] o *PyTorch* [15].

A pesar de todos estos avances, tanto en eficacia de resultados como en eficiencia gracias al cómputo en GPU, las redes neuronales convolucionales siguen teniendo problemas de eficiencia tanto en tiempo de entrenamiento como en inferencia. En tiempo de entrenamiento es claro el coste elevado de

recursos, en inferencia es necesario gran cantidad de espacio de almacenamiento para guardar los pesos aprendidos y potencia de cómputo para realizarla, resultando así, imposible las implementaciones con elevados requisitos de eficiencia como puede ser el caso de aplicaciones a ejecutar en tiempo real o la implementación en dispositivos de limitados recursos.

Diversas técnicas se han desarrollado con el objetivo de afrontar este problema. El presente trabajo se centrará concretamente en la técnica de poda de conexiones sobre redes neuronales ya entrenadas. A lo largo de este capítulo se presentará la arquitectura general de una red neuronal así como sus métodos de entrenamiento y se explicará en más detalle la problemática expuesta. En el capítulo 2 se explicará las diferentes técnicas presentes en el estado del arte para afrontar este problema. En el capítulo 3 se explicará la investigación realizada y los objetivos perseguidos. En el capítulo 4 se presentarán y discutirán los resultados obtenidos para cada experimento propuesto. Finalmente, en el capítulo 5 se expondrán las conclusiones y se plantearán posibles líneas futuras de investigación.

1.2. Redes neuronales convolucionales

1.2.1. Red neuronal artificial

Una red neuronal artificial es un modelo de cómputo basado en la composición de un gran conjunto de unidades, llamadas neuronas, organizadas de forma análoga al comportamiento observado en un sistema biológico; de ahí recibe el nombre este modelo. Este se puede representar como un grafo dirigido acíclico, donde cada nodo representa una neurona y cada arco una interconección entre dos neuronas con un peso asociado al mismo que será aprendido durante el entrenamiento.

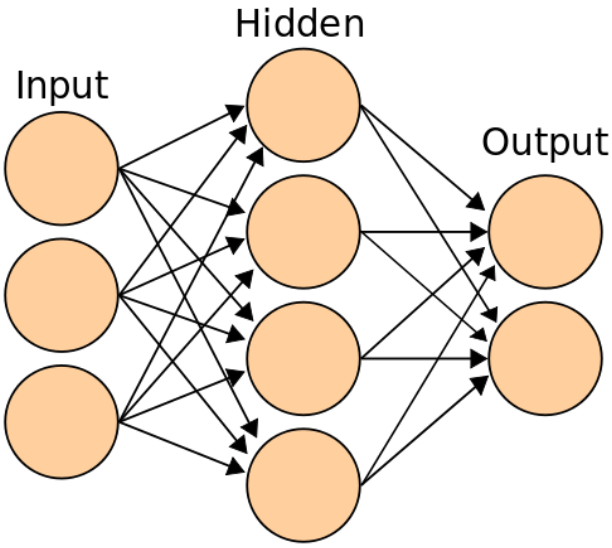


FIGURA 1.1: Representación de una red neuronal artificial.

De forma general, el grafo está organizado en capas tal y como se muestra en la figura 1.1. Estas capas se dividen en 3 tipos: entrada, ocultas y salida. Este grafo representa el modo en que se realizará las operaciones, dando una salida de tantas dimensiones como número de nodos tenga la última capa para una entrada de dimensión igual al número de nodos en la primera capa. En el ejemplo de la figura 1.1 se muestra una red neuronal que tiene como entrada un vector de dimensión 3 y salida 2.

Para calcular la salida de la red se calcula la salida de cada nodo siguiendo la fórmula:

$$z_j = \sum_{i \in I_j} w_{ij} x_i \tag{1.1}$$

Donde z_j es la salida del nodo j, x_i es la salida del nodo i, I_j es el conjunto de nodos que tienen un arco con destino j y w_{ij} es el peso del arco del nodo i al j. En la figura 1.2 se muestra el ejemplo gráfico para una neurona.

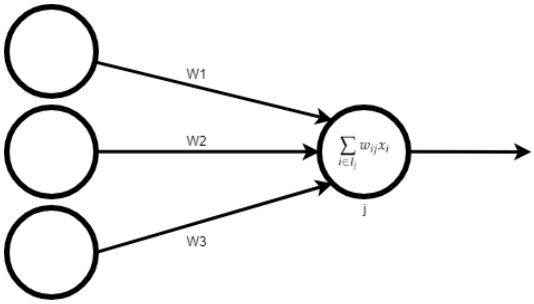


FIGURA 1.2: Cálculo de la salida de una neurona.

El número de capas ocultas puede variar entre arquitecturas diferentes así como las operaciones realizadas en cada neurona o la cantidad y complejidad de las conexiones. Para mejorar la eficacia se agregan a cada nodo sesgos y a cada salida una función de activación que agrege elementos no lineales al sistema, de modo que, el cálculo de salida para cada nodo, siguiendo la notación anterior, resulta:

$$z_j = h(\sum_{i \in I_j} w_{ij} x_i + b_j) \tag{1.2}$$

Donde b_j es el sesgo del nodo j y $h(.)$ es la función de activación. Esta es una función que modifica la salida, normalmente para agregar elementos no lineales permitiendo así generalizar comportamientos más complejos. Algunas de estas funciones se detallarán en 1.2.2.

1.2.2. Red neuronal convolucional

El modelo base de red neuronal artificial se extiende al concepto de red neuronal convolucional. Estas, al igual que las vistas en la sección 1.2.1 están organizadas en capas, cada una recibe las entradas de la capa anterior, las procesa con unos pesos aprendidos y da una salida a ser utilizada por la siguiente capa.

Este modelo se diferencia del anterior al trabajar principalmente con imágenes y de sustituir la operación de suma por la operación de convolución, de ahí su nombre.

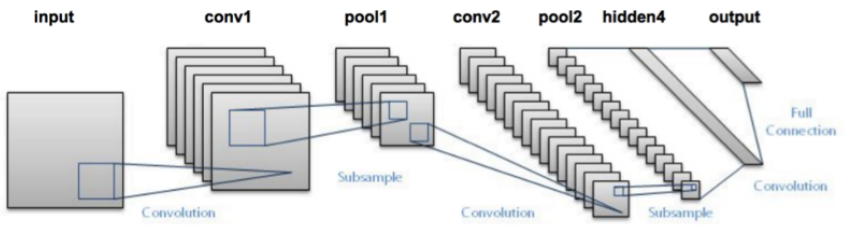


FIGURA 1.3: Representación de una red neuronal convolucional.

En la figura 1.3 se muestra la arquitectura de una red neuronal convolucional, en concreto la arquitectura de la red clasificadora LeNet [16]. Como puede observarse, cada capa se organiza en forma de un volumen tridimensional que varía sus dimensiones a lo largo de la red. Cada volumen de dimensiones (W, H, D) está compuesto por D *feature maps* o mapas de características de dimensiones (W, H), correspondientes con coordenadas espaciales de la imagen de entrada.

Para una red neuronal de L capas, denótese el conjunto de mapas de características $z_l \in \mathbb{R}^{W_l \times H_l \times D_l}$ con dimensión $W_l \times H_l$ y profundidad (o número de mapas de características) D_l. Estos conjuntos de mapas de características, z_l, se corresponden a cada una de las capas de la red pudiendo ser o bien su entrada, z_0, o bien la salida de algunas de las mismas cuando $l \in (1, ..., L)$. Cada mapa de características individual se denotará como $z_l^{(k)}$ con $k \in (0, ..., D_l)$, siendo este el k-ésimo mapa de características de la l-ésima capa.

Con esta notación se explicará las operaciones que realizan las distintas capas de una red neuronal convolucional. Estas se dividen en 4 grandes grupos:

- **Convoluciones**: Estas capas dan nombre a este tipo de redes neuronales, realizan distintos tipos de convolución con los mapas de característica de entrada para generar su salida.
- **Redimensión**: Se ocupan de redimensionar la entrada utilizando diferentes tipos de operaciones con el fin de aumentar el campo receptivo de cada neurona.
- **Activación**: Estas capas aplican funciones que añaden elementos no lineales al cálculo con el fin de mejorar el potencial de generalización del error. También pueden ser utilizadas como salida de una red clasificadora de dos clases.

- **Otras capas**: Existen numerosas otras capas que han demostrado ser vitales en muchos casos. En este grupo se presentarán las más importantes.

Capas convolucionales

Las capas de convolución aplican un *kernel* de convolución al conjunto de mapas de características de entrada. Las dimensiones del *kernel* de convolución viene definido por la arquitectura de la red, sin embargo, los valores del mismo son aprendidos durante el entrenamiento. Recordando la operación de convolución, esta se define, para cada pixel (i, j) de cada mapa de características:

$$z_l^{'(i,j,k)} = \sum_{p=0}^{m} \sum_{q=0}^{n} w_{p,q} z_l^{((i+p-m/2),(j+q-n/2),k)} \quad \forall (i,j) \in (W_l, H_l) \tag{1.3}$$

Donde $z_l^{'(k)}$ es el mapa de características resultante de convolucionar $z_l^{(k)}$ con el *kernel* w de tamaño $m \times n$. De ahora en adelante, por simplicidad de notación, se denotará esta operación con el símbolo $*$, siendo $z_l^{'(k)} = w * z_l^{(k)}$ equivalente a 1.3.

El objetivo de utilizar convoluciones en redes neuronales es para filtrar la entrada y así identificar patrones locales en cada parte de la imagen. Esto se consigue gracias a que operan a nivel local, moviendo una «ventana deslizante» en toda la imagen para identificar estos patrones, siendo esta ventana el propio *kernel* de convolución.

A continuación se presentarán algunas de las capas de convolución más utilizadas.

Convolución 2D

Esta es la capa más utilizada, una convolución de todos los mapas de característica de entrada. Dada una capa de entrada z_{l-1} de dimensiones $(W_{l-1}, H_{l-1}, D_{l-1})$ y unos pesos $w_l \in \mathbb{R}^{m \times n \times D_{l-1} \times D_l}$ con $m \times n$ siendo el tamaño del *kernel*, para obtener la capa z_l se aplica la operación:

$$z_l^{(k)} = \sum_{i=0}^{D_{l-1}} w_l^{(i,k)} * z_{l-1}^{(i)} \qquad \forall k \in [0, ..., D_l] \tag{1.4}$$

Realizando así $D_{l-1} \times D_l$ convoluciones para así obtener la salida. Como es de observar, esta operación permite cambiar el número de mapas de características de la siguiente capa independientemente de la cantidad que hubiera en la capa anterior.

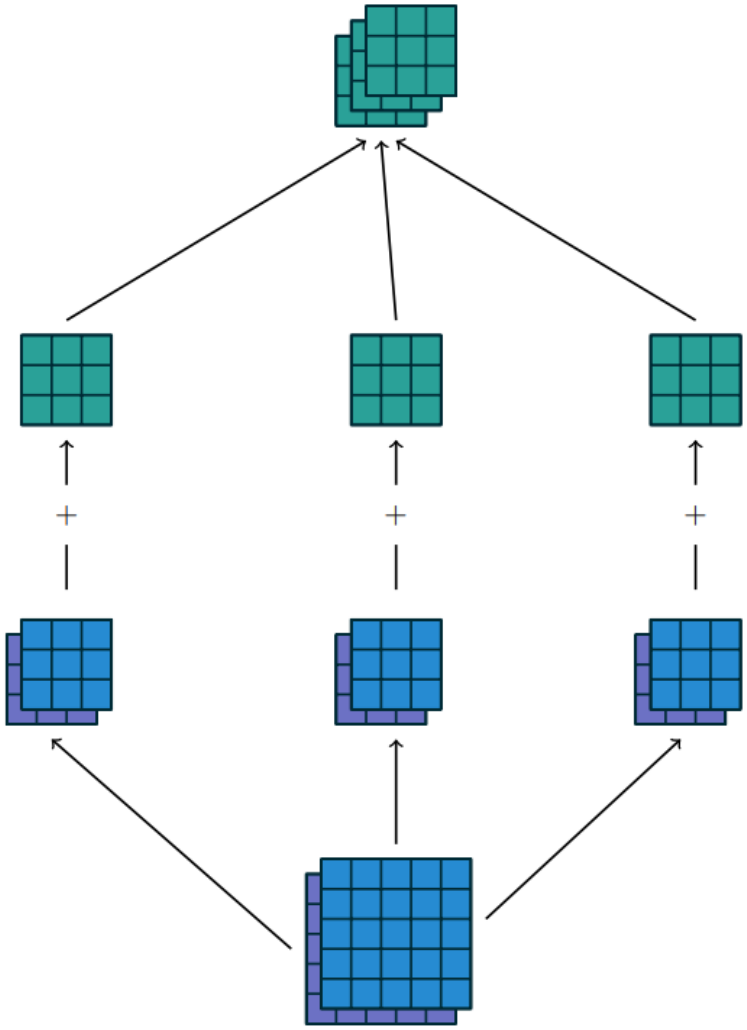


FIGURA 1.4: Convolución 2D. Imagen tomada de [1].

En la figura 1.4 se muestra gráficamente una convolución con 2 mapas de características de entrada y 3 de salida.

Un caso particular de convolución 2D es el caso de las capas *fully connected*, FC de ahora en más. Estas suelen ser las últimas capas presentes en las redes clasificadoras, véase [17, 16], siendo utilizadas para agregar información global de toda la imagen. Estas consisten en multiplicar cada pixel del volumen de la entrada por un valor diferente y reducir mediante suma el resultado, repitiendo el proceso tantas veces como dimensiones de salida tenga la capa.

Las FC se pueden implementar como una convolución al redimensionar la entrada z_{l-1} a $(1 \times 1 \times W_{l-1}H_{l-1}D_{l-1})$ y realizando la convolución con $w_l \in \mathbb{R}^{1\times1\times W_{l-1}H_{l-1}D_{l-1}\times D_l}$.

Convolución 2D con *stride*

Como se acaba de exponer, el resultado de una convolución 2D, suponiendo que se ha realizado el *padding* necesario correctamente, dará como resultado imágenes de igual ancho y alto que la entrada. En algunos casos es deseable aplicar una reducción de estas dimensiones de forma implícita durante la convolución, esto se logra aplicando un *stride* mayor a 1.

El *stride* indica el salto que se aplicará entre cada aplicación del *kernel* de convolución, de tal forma, que para una entrada z_{l-1} de dimensiones $(W_{l-1}, H_{l-1}, D_{l-1})$, la salida z_l será de dimensiones $(W_{l-1}/s, H_{l-1}/s, D_l)$, siendo s el tamaño del *stride*.

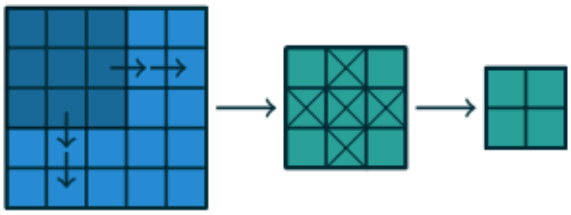
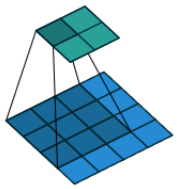


FIGURA 1.5: Convolución 2D con un *kernel* 3x3 con *stride* de 2.
Imagen tomada de [1].

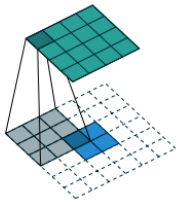
En la figura 1.5 se muestra el ejemplo de una convolución con *stride* de tamaño 2. Obviando los bordes, el resultado sin *stride* sería de dimensiones 3x3, sin embargo, al haberse aplicado un salto de 2, las posiciones marcadas con cruces no se calcularon, con lo que el resultado final es de dimensiones 2x2.

Convolución 2D traspuesta

Una convolución traspuesta, también llamada deconvolución aunque no sea una deconvolución matemática, recibe el nombre del hecho de que, si la convolución se calcula como una multiplicación de matrices, esta puede ser calculada multiplicando con la traspuesta de la matriz de convolución.



(A) Convolución 2D



(B) Deconvolución 2D

FIGURA 1.6: Convolución/deconvolución 2D con un *kernel* 3x3. Imagen tomada de [1]

Observando la figura 1.6a, una convolución sería el resultado de aplicar un *kernel* 3x3 a la imagen azul de 4x4, dando como resultado la imagen verde de 2x2. Véase la reducción de dimensiones dado el tratamiento de bordes.

La convolución traspuesta invierte las dimensiones, siendo la entrada la imagen de dimensiones 2x2 y la salida la imagen de 4x4. Esta puede ser implementada mediante una convolución de la entrada con el *padding* correspondiente tal y como se muestra en la figura 1.6b.

Convolución 2D dilatada

La idea de una convolución dilatada se basa en aplicar el concepto de *stride* sobre el *kernel* en lugar de sobre la imagen, consiguiendo así aumentar el campo receptivo de cada neurona sin necesidad de redimensionar la imagen.

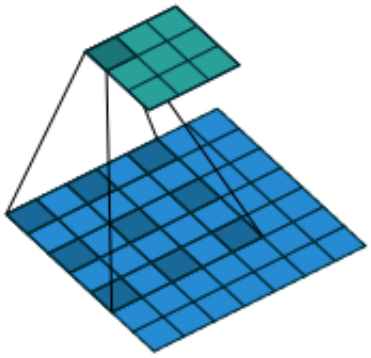


FIGURA 1.7: Convolución 2D de *kernel* 3x3 con dilatación 2. Imagen tomada de [1].

En la figura 1.7 se muestra un ejemplo de una convolución dilatada. La imagen azul es la de entrada, la verde la de salida y los pixeles sombreados sobre

la azul son los que intervinieron en la convolución para generar el pixel sombreado en la imagen verde. Un modo de ver esta operación es la de aplicar en lugar de un *kernel* 3x3 en este caso, uno de 5x5 con los pesos con alguna coordenada impar puestos a 0.

Convolución 2D *depthwise*

Por último, un tipo de convolución utilizada sobre todo en arquitecturas que se centran en aumentar su eficiencia, como se explicará en 2.3.1, es la convolución *depthwise*. Este tipo de convolución propone calcular la capa z_l utilizando los pesos $w \in \mathbb{R}^{m \times n \times D_{l-1}}$ siguiendo:

$$z_l^{(k)} = w * z_{l-1}^{(k)} \quad \forall k \in [0, .., D_{l-1}] \tag{1.5}$$

De modo que en lugar de aplicar para la capa l $D_{l-1} \times D_l$ convoluciones como sucede en una convolución 2D, se aplicarán únicamente tantas convoluciones como mapas de características tenga la capa de entrada. En 2.3.1 se verá como se utilizarán este tipo de operaciones para mejorar la eficiencia del modelo.

Capas de redimensión

Una forma de agregar información y aumentar el campo receptivo de cada neurona es reducir el tamaño de cada mapa de característica. De esta forma, se consigue agregar información contextual de localidades cada vez más grandes, hasta conseguir agregar la información de toda la imagen como suelen hacer las redes clasificadoras. Véase el caso de [16, 17, 18, 19] por ejemplo, todas estas son redes clasificadoras que reducen el tamaño de sus mapas de características hasta el tamaño de un pixel para luego proceder con operaciones de tipo FC.

En la literatura, las capas dedicadas a reducir el tamaño de la entrada son llamadas *pooling layers*. Estas tienen la característica de reducir el tamaño de las imágenes de entrada aplicando distintas operaciones, en algunos casos operaciones no lineales, sin modificar su profundidad.

Max pool

La operación de **max pool** consiste en obtener el máximo de una ventana de

tamaño dado. Al igual que en el caso de una convolución, el nivel de reducción de tamaño depende del *stride* aplicado, no obteniendo niguna redimensión al aplicar un *stride* de 1.

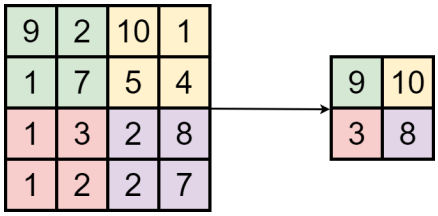


FIGURA 1.8: Operación de *Max pool* con una ventana de 2x2 y *stride* 2. Cada color muestra cada una de las aplicaciones de la ventana deslizante.

En la figura 1.8 se muestra el ejemplo de aplicación de un *max pool* de tamaño 2x2 con *stride* 2. Véase que en este ejemplo se han seleccionado los *strides* y tamaños de ventanas de tal modo que no se solapen.

Average pool

Esta operación cumple las mismas características que el **max pool**, salvo que para calcular el resultado aplica la operación de media de los píxeles dentro de su ventana. Nótese que este es un caso particular de una convolución 2D *depthwise* con *stride* con todos sus pesos igual a $\frac{1}{mn}$.

Capas de activación

Como se ha adelantado en la ecuación 1.2, en muchos casos se aplican funciones, llamadas de activación, a la salida de algunas capas con el fin de agregar elementos no lineales al modelo.

Algunas de las más utilizadas son las clásicas sigmoide y tangente hiperbólica, rectificaciones lineales (ReLU de ahora en más) [20], ELU (*exponential* ReLU) [21] o las *leaky* ReLu [22]. Su formulación matemática es la siguiente respectivamente:

$$h(x) = \frac{1}{1 + e^{-x}} \tag{1.6}$$

$$h(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{1.7}$$

$$h(x) = max(x, 0) \tag{1.8}$$

$$h(x) = \begin{cases} x & \text{si } x \geq 0 \\ \alpha(e^x - 1) & \text{de otro modo} \end{cases} \tag{1.9}$$

$$h(x) = \begin{cases} x & \text{si } x > 0 \\ \alpha x & \text{de otro modo} \end{cases} \tag{1.10}$$

Tanto la tangente hiperbólica como sigmoide han sido las más utilizadas clásicamente en las redes neuronales artificiales. Sin embargo, su cómputo es más complejo que otras.

Otras funciones se han desarrollado, como ReLU, estas han ido sustituyendo a las funciones de activación clásicas dadas las siguientes ventajas:

- Bajo coste computacional.
- Mejor propagación del gradiente para valores mayores a 1.
- Invariabilidad a la escala: $max(0, ax) = a\ max(0, x) \quad \forall a > 0$.

De especial interés es la propiedad de propagación del gradiente, tómese como ejemplo la función sigmoide $\sigma(x) = \frac{1}{1+e^{-x}}$, su derivada es: $\frac{\delta\sigma(x)}{\delta x} = \sigma(x)(1 - \sigma(x))$ mostradas en la figura 1.9.

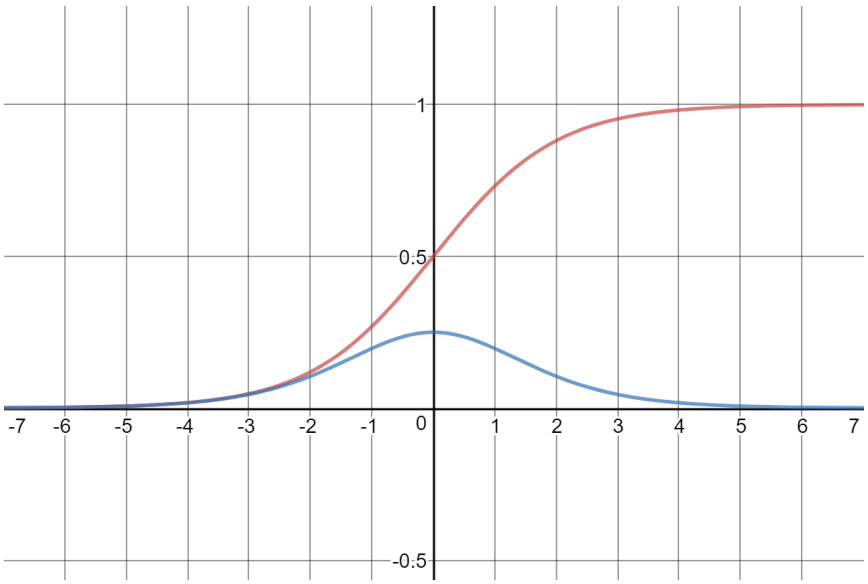


FIGURA 1.9: Función sigmoide en rojo y su derivada en azul.

Véase que tanto el dominio de la función como de su derivada es $(0,1)$. Al ser el valor de la derivada siempre menor a 1, este disminuirá el valor de las derivadas de las capas anteriores a esta, provocando así el efecto de *vanishing gradient*.

La derivada de las funciones ReLU es 1 para valores mayores a 0. Con lo que, contribuyen a no agravar el problema del *vanishing gradient*, por esa razón se dice que cuentan con una mejor propagación del gradiente. Nótese que matemáticamente la función no es derivable, por lo que, el cálculo de la derivada en 0 variará dependiendo de la implementación.

Con la utilización de la función ReLU, emerge el problema de las *dead neurons*. Esto es provocado a causa de no propagar el gradiente para valores menores a 0, pudiendo caer en situaciones en las que no se genera ninguna activación independientemente de la entrada; a estas neuronas que no generan ninguna activación se les llama *dead neurons*.

Para solventar este problema, se utilizan funciones como ELU o *leaky* ReLu que permiten una propagación del gradiente, aunque sea mínima, para valores menores a 0, manteniendo los beneficios de la función ReLU.

En [23] se realiza un estudio exhaustivo de diversas funciones de activación sobre redes neuronales convolucionales.

Otras capas

En este grupo se explicará brevemente otro tipo de capas que son extensamente utilizadas en la literatura por diferentes arquitecturas que han demostrado ser vitales para su éxito. En concreto se presentarán las capas de *batch normalization* (BN de ahora en más) [24] y *dropout* [2].

BN

El proceso de entrenamiento de una red neuronal resulta más complicado más profunda sea. En gran parte esto se debe a que la distribución de valores de cada capa varía durante el entrenamiento al variar los pesos de las capas anteriores. Una técnica utilizada frecuentemente es normalizar las entradas de la red, asegurándose que cada instancia de entrenamiento tenga media 0 y varianza 1, sin embargo esto no garantiza que los valores de entrada a cada capa intermedia de la red esten normalizados.

Partiendo de esta idea de normalización nace la técnica BN. Esta técnica pretende hacer formar parte de la arquitectura al proceso de normalización, consiguiendo que la entrada a cada capa de activación esté normalizada. De modo que, la normalización de cada capa se consigue, de la siguiente forma:

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathbb{E}(x^{(k)})}{\sigma^2(x^{(k)})} \tag{1.11}$$

Donde $x^{(k)}$ es el k-ésimo mapa de características de una capa arbitraria, $\mathbb{E}(x^{(k)})$ es la esperanza y $\sigma^2(x^{(k)})$ su varianza.

El cálculo de la esperanza y varianza real de todo el conjunto de entrenamiento para cada capa en que se quiera normalizar resulta no solo impráctico, sino imposible computacionalmente en casi todos los casos. Como se verá en 1.2.3, para el entrenamiento se utilizan *mini-batches*, con lo que, cada *mini-batches* produce una estimación de la media y varianza de cada activación. De esta forma se puede modificar el algoritmo de normalización para que utilice la media y varianza del *mini-batch*.

Para asegurarse de que la transformación a aplicar pueda representar la transformación identidad, se añaden dos parámetros más a la transformación, la escala y traslación, resultando el cálculo de la siguiente forma:

$$BN(x) = \gamma^{(k)}\hat{x}^{(k)} + \beta^{(k)} \tag{1.12}$$

Siendo los parametros de escala γ y traslación β aprendibles durante el entrenamiento.

Por último, la forma en que se utiliza esta operación en la literatura es utilizando la media y varianza de la población en lugar de las de los *mini-batches*. Esto se consigue considerando la media y varianza de cada capa a normalizar como un parámetro aprendible, que se actualiza con cada iteración de entrenamiento. De modo que, una vez finalizado el entrenamiento, se tendrá un valor fijo tanto para la media como varianza, siendo estos los utilizados para la normalización en tiempo de inferencia.

Dropout

Uno de los problemas de las redes neuronales más frecuentes es el sobreajuste. Esto ocurre principalmente cuando los datos de entrenamiento son limitados, logrando que el modelo aprenda relaciones complejas existentes

únicamente en este limitado conjunto, muchas veces ocasionadas por el ruido, que no están presentes en datos reales.

Un modo de prevenir estas co-adaptaciones complejas, es decir, situaciones en que algunas neuronas solo son útiles en conjunción con otras, es aplicando la técnica de *dropout*. Esta técnica consiste en omitir temporalmente algunas de las neuronas de alguna capa de la red neuronal durante el entrenamiento, consiguiendo así que cada una aprenda a identificar patrones útiles para la predicción.

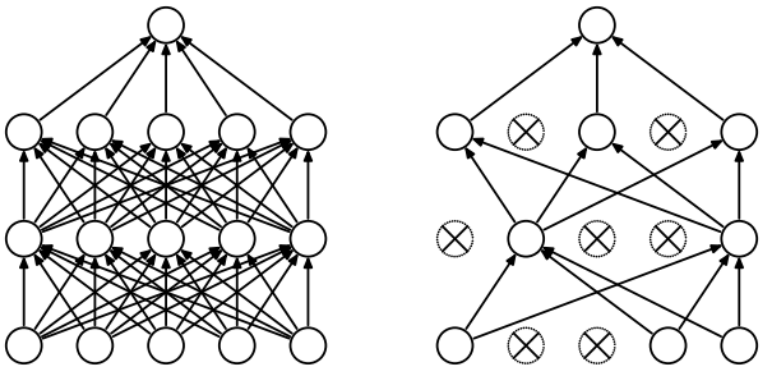


FIGURA 1.10: Ejemplo de aplicación de *dropout*. Imagen tomada de [2].

En la figura 1.10 se muestra un ejemplo de aplicación de *dropout* en una iteración de entrenamiento. En la fase de inferencia se utilizarán todas las neuronas, promediando así el resultado de cada uno de los sub-detectores entrenados en fase de aprendizaje.

1.2.3. Proceso de entrenamiento

El objetivo del entrenamiento es aprender los pesos de las variables que permitan mejorar el poder de predicción del modelo. Este aprendizaje se realiza optimizando una función de costes definida, permitiendo así encontrar qué combinación de parámetros han de tener las variables del modelo para incurrir en coste mínimo.

De los métodos existentes de optimización, el más usado es el método de gradiente descendiente. Este es un método que persigue minimizar la función de costes $L(\theta)$, parametrizada con θ, siendo este el conjunto de valores de las variables del modelo. Para encontrar el mínimo de la función, actualiza los valores de θ iterativamente en la dirección opuesta del gradiente de la

función objetivo $\Delta L(\theta)$ respecto a cada uno de los parámetros. El salto que se realiza en cada iteración dependerá del ratio de aprendizaje, α, que será utilizado para modificar cada uno de los parámetros hasta alcanzar un mínimo local.

Principalmente existen 3 tipos de optimización por gradiente descendiente: *batch gradient descent*, *stochastic gradient descent* (SGD de ahora en más) y *mini-batch gradient descent*.

Batch gradient descent

La técnica de *batch gradient descent* consiste en calcular el gradiente de la función de costes respecto a los parametros θ para cada una de las instancias de entrenamiento y actualizar iterativamente estos parámetros, siguiendo la fórmula, siendo t el número de iteración:

$$\theta_{t+1} = \theta_t - \alpha \Delta_{\theta_t} L(\theta_t) \tag{1.13}$$

Como es de observar, este método es muy lento al tener que calcular los gradientes de todo el *dataset* para realizar una única actualización. Este método garantiza el óptimo global para funciones convexas.

SGD

SGD en contraste con *Batch gradient descent* realiza las actualizaciones de los parámetros por cada instancia de entrenamiento vista.

$$\theta_{t+1} = \theta_t - \alpha \Delta_{\theta_t} L(\theta_t, x^i, y^i) \tag{1.14}$$

Donde x^i, y^i es el par instancia y predicción esperada del conjunto de entrenamiento T con $i \in (1, .., |T|)$. Este método permite actualizaciones mucho más rápidas y permite también re-aprovechar un modelo ya entrenado sobre un conjunto para realizar actualizaciones sobre el mismo con nuevas instancias sin necesidad de re-entrenarlo por completo.

El principal inconveniente de este método es la fluctuación que se produce en la función de costes al realizar actualizaciones en base a una única instancia.

Mini-batch gradient descent

Por último, este método combina lo mejor de los dos anteriores. Este realiza actualizaciones por *mini-batches* de tamaño n, de modo que, para n pares de instancias de entrenamiento x^i, y^i se calcula:

$$\theta_{t+1} = \theta_t - \alpha \Delta_{\theta_t} L(\theta_t, x^{i+j}, y^{i+j}) \quad \forall j \in (1, .., n) \tag{1.15}$$

De este modo, se tienen actualizaciones frecuentes como en SGD a la vez que se reduce la varianza de la función de costes gracias a la aplicación sobre un sub-conjunto completo de entradas de entrenamiento.

Este es el método base utilizado para entrenar redes neuronales en la literatura. Este método normalmente se le llama SGD, por lo que, de ahora en más se hará referencia al mismo con ese nombre. En[25] se muestra una revisión de los distintos algoritmos de optimización.

1.3. Problemática

Las problemáticas de las redes neuronales convolucionales se pueden dividir en dos categorias: aquellas relacionadas con el entrenamiento y las relacionadas con la inferencia. Este trabajo se centrará en las relacionadas con la inferencia, en concreto, con la eficiencia de cómputo y de utilización de memoria.

Desde que se generalizó el éxito de las redes neuronales, se fueron creando arquitecturas cada vez más profundas y complejas, llevando así a una ingente utilización de recursos.

	Parámetros	MACCs
LeNet [16]	431K	2.29M
VGG-16 [18]	138.3M	154.7G
ResNet50 [19]	25.56M	3.87G

TABLA 1.1: Número de parámetros y operaciones de diferentes arquitecturas.

En la tabla 1.1 se muestra el número de parámetros (cantidad de flotantes) y operaciones de multiplicación y suma de algunas de las arquitecturas más conocidas. Nótese la inmensa utilización de memoria y cómputo, solo la VGG-16 requiere más de 500 MB de memoria para almacenar sus pesos aprendidos.

Teniendo en cuenta estos requisitos computacionales, resulta imposible utilizar estos modelos en arquitecturas de bajo poder de cómputo como pueda ser dispositivos móviles u otros sistemas embebidos.

1.4. Objetivos

Como se ha visto, el tamaño y complejidad de muchos modelos imposibilita su implementación en ciertos dispositivos o para ciertas aplicaciones que requieran alto rendimiento. Diversas técnicas existen para afrontar este problema, tal y como se verá en el capítulo 2. El presente trabajo pretende estudiar la eficacia de la técnica de poda de conexiones en redes neuronales convolucionales, analizando el comportamiento del modelo resultante para diversas configuraciones de métodos con el fin de encontrar buenos protocolos de poda para abordar la problemática expuesta.

Capítulo 2

Estado del arte

El desarrollo en redes neuronales hasta hace pocos años se centraba en mejorar su desempeño en diferentes tareas. Esto se logró creando modelos cada vez más grandes tal y como se aprecia en las historia de los últimos ganadores de la *Imagenet Large Scale Visual Recognition Challenge* (ILSVRC de ahora en más) en la figura 2.1.

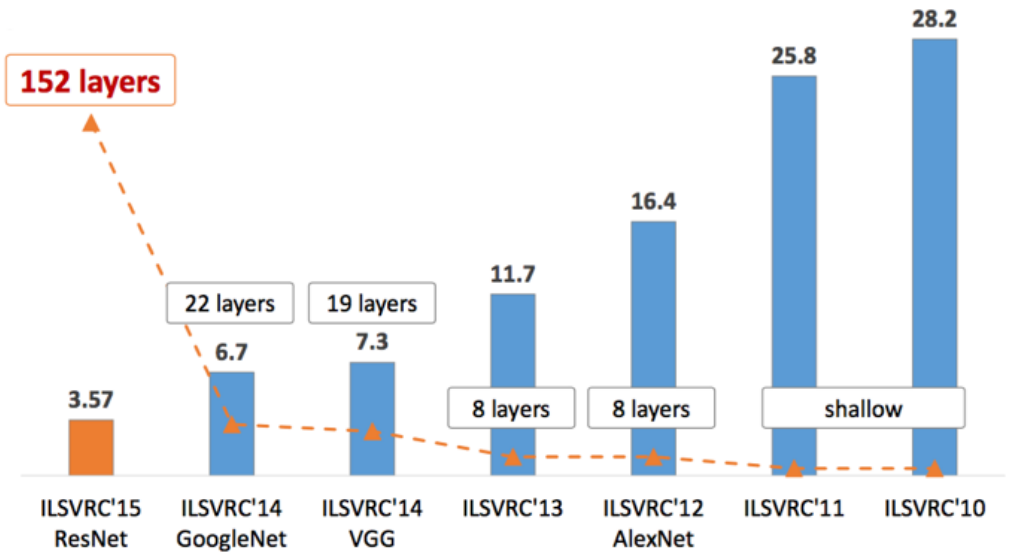


FIGURA 2.1: Evolución del error top-5 en ILSVRC. Imagen de Kaiming He.

Como puede verse, se ha llegado a utilizar modelos de hasta 152 capas, resultando imposible implementar dichos modelos en arquitecturas de limitados recursos.

Desde hace pocos años se ha desarrollado una línea de investigación centrada en mejorar la eficiencia de las arquitecturas. Con este fin, se han investigado diversas técnicas, desde modificación del modelo una vez entrenado hasta

arquitecturas orientadas a la eficiencia. En este capítulo se presentará algunas de estas líneas de investigación.

2.1. Compresión

Esta línea de investigación propone aplicar diversas técnicas con el principal objetivo de reducir el tamaño de los pesos de los modelos. Uno de los trabajos que presenta varias de estas técnicas es el que propone la técnica llamada *Deep compression* [3]. Esta puede resumirse en las 3 faces representadas en la figura 2.2.

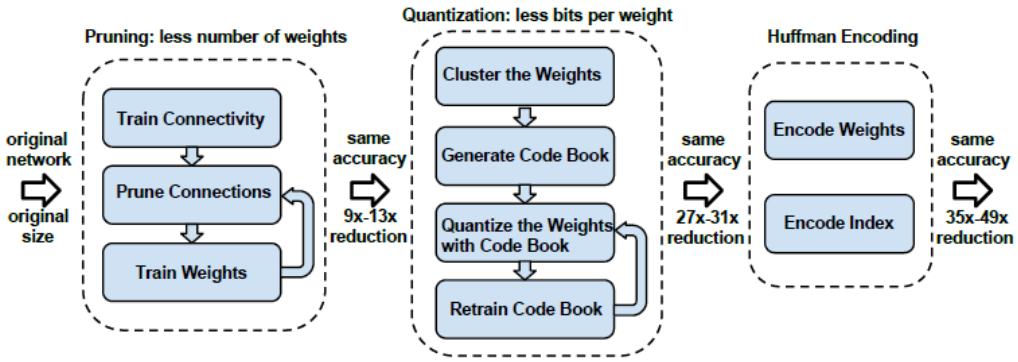


FIGURA 2.2: Proceso de compresión *Deep compression*. Imagen tomada de [3].

Estas 3 fases son: poda de conexiones, cuantificación y compresión. La primera fase consiste en aplicar un método de poda a una red ya entrenada, de los diferentes métodos de poda se discutirá en la sección 2.4.

La segunda fase consiste en cauntificar los pesos del modelo con el fin de utilizar menos bits para su representación. Una vez cuantificados los pesos en diferentes clústeres, se refina esta cuantificación realizando un entrenamiento sobre los mismos.

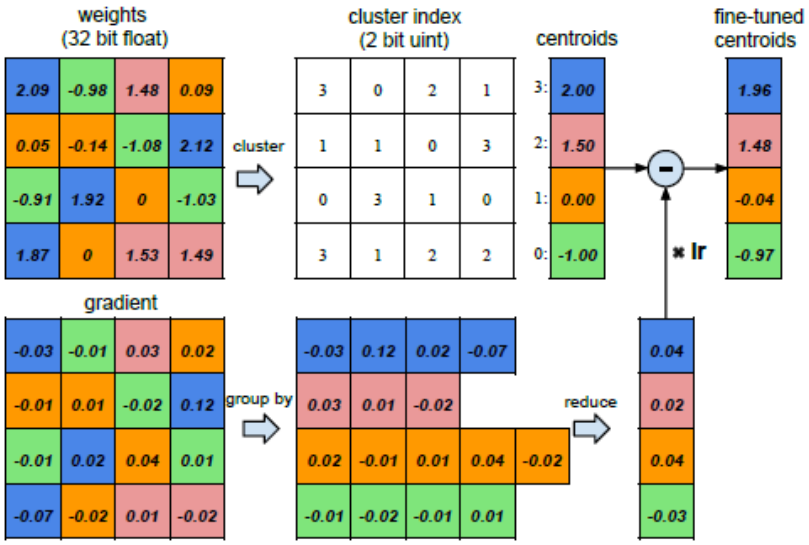


FIGURA 2.3: Esquema de cuantificación de pesos con 2 bits y proceso de actualización de centroides. Imagen tomada de [3].

En la figura 2.3 se muestra un ejemplo de cuantificación de pesos representados con flotantes de 32 bits a clústeres de 2 bits, permitiendo así compartir los pesos que caigan en el mismo clúster. También se muestra el esquema de actualización de centroides de los clústeres aplicada durante el proceso de entrenamiento de la cuantificación.

Por último, aplica la codificación de Huffman [26] para una mayor compresión.

2.2. Destilación de conocimiento

Esta línea de investigación pretende transferir el conocimiento de una red más grande, llamada «profesora», a una red mucho más compacta y eficiente, la «estudiante», de tal modo de conseguir que la red «estudiante» imite a su «profesora» pero de manera mucho más eficiente. A esta técnica se le llama destilación de conocimiento [27].

Esto se consigue en dos fases; en primer lugar, se entrena la red «profesora» en la tarea deseada. En segundo lugar, se entrena la «estudiante» utilizando tanto los datos como la propia red «profesora». Esta red «estudiante» se entrena utilizando una función de pérdida que contempla tanto el error respecto a la salida esperada como el error respecto a la salida de la red «profesora». Formalmente:

$$L_s = H(p_s(x), y) + \lambda H(p_s(x), p_t(x)) \tag{2.1}$$

Siendo L_s la función de costes de la red estudiante, $p_s(x)$ y $p_t(x)$ la probabilidad de clasificación de cada clase (salida de la operación *softmax*) dada una entrada x para la red «estudiante» y «profesora» respectivamente, «H(., .)» la función de entropía cruzada, y el valor esperado para la entrada «x» y λ un parámetro que permite modificar el peso del coste con la red profesora.

Para guiar aún más el entrenamiento distintos trabajos proponen diferentes métodos para mejorar el entrenamiento de la red «estudiante», como el caso de [4] que propone contemplar más funciones de costes a lo largo de la red dentro de las capas oclutas de la misma tal y como se muestra en la figura 2.4.

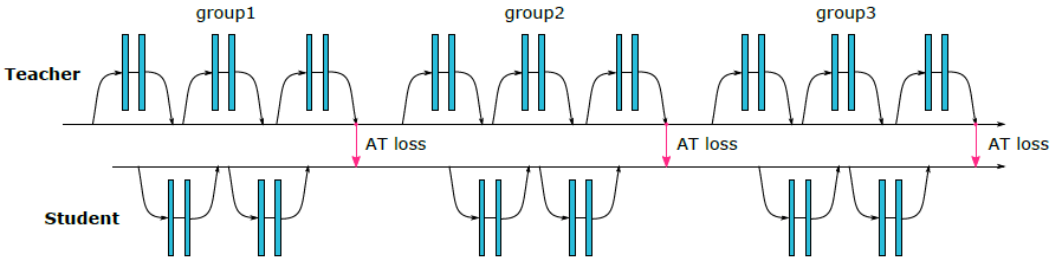


FIGURA 2.4: Esquema de funciones de costes adicionales en el entrenamiento de una red «estudiante». Imagen de [4].

2.3. Modificación de arquitectura

Esta línea de investigación propone utilizar arquitecturas ligeras para así mejorar la eficiencia del modelo. Estas se basan en la utilización de operaciones de bajo coste que sustituyen a los módulos más pesados de las arquitecturas tradicionales. Concretamente, se expondrán 3 formas diferentes de sustitur la operación de convolución: mediante las convoluciones separables de la *MobileNet* [5], con los módulos *fire* de la *SqueezeNet* [28] o con la alternativa de la operación de *shift* [6].

2.3.1. *MobileNet*

Esta arquitectura recibe el nombre de su propósito: el de ser implementada en dispositivos móviles. En la mayoría de arquitecturas que consiguen resultados del estado del arte la mayor parte de las operaciones son consumidas por las convoluciones, en particular, por aquellas con un *kernel* mayor a 1x1. Esta arquitectura propone sustituir estas convoluciones por una convolución *depthwise* como la vista en 1.2.2, seguida de una convolución 2D 1x1 tal y como se muestra en la figura 2.5. Estas dos operaciones toman el nombre de convolución separable.

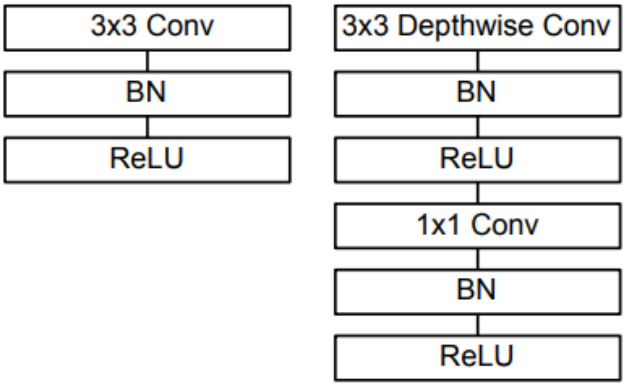


FIGURA 2.5: Ejemplo de sustitución de una convolución 2D por una *depthwise*. Imagen tomada de [5]

La motivación de esta modificación es la utilización de una operación capaz de simular una convolución 2D con la utilización de información local en cada mapa de característica con una convolución *depthwise* y la agregación de la misma con una convolución 1x1. Si bien matemáticamente no es una operación equivalente, ambas tienen la misma semántica, con lo que, es de esperar tener una ganancia en eficiencia asumiendo un coste en precisión. Estas dos operaciones combinadas son mucho más eficientes que una única operación de convolución 2D. El coste computacional en número de operaciones de una convolución normal en la capa l viene expresado con la ecuación:

$$C(*_l) = k_w k_h D_{l-1} D_l W_{l-1} H_{l-1} \tag{2.2}$$

Con $C(*_l)$ como el coste en operaciones de realizar una convolución en la capa l, (k_w, k_h) las dimensiones del *kernel*, $(W_{l-1}, H_{l-1}, D_{l-1})$ las dimensiones de la capa de entrada y D_l la profundidad de la capa de salida.

Mientras que las propuestas por la *MobileNet* tienen un coste en operaciones:

$$C(\text{sep}_l) = k_w k_h D_{l-1} W_{l-1} H_{l-1} + D_l D_{l-1} W_{l-1} H_{l-1} \tag{2.3}$$

Con $C(\text{sep}_l)$ como el coste en operaciones de realizar una convolución *depthwise* con un *kernel* de dimensión (k_w, k_h) seguida de una 1x1 en la capa l.

Por lo que, la ganancia en eficiencia puede expresarse como:

$$\frac{C(\text{sep}_l)}{C(*_l)} = \frac{k_w k_h D_{l-1} W_{l-1} H_{l-1} + D_l D_{l-1} W_{l-1} H_{l-1}}{k_w k_h D_{l-1} D_l W_{l-1} H_{l-1}} = \frac{1}{D_l} + \frac{1}{k_w k_h} \tag{2.4}$$

Por lo que, desde que la salida tenga que tener más de 1 mapa de característica ya se tendrá una ganancia en eficiencia.

2.3.2. *SqueezeNet*

Con la misma idea de acelerar las convoluciones, la arquitectura de *SqueezeNet* se centra en acelerar las convoluciones 3x3, sustituyendo estas por los que denomina módulos *fire*. Este módulo está compuesto por 3 convoluciones, una primera convolución 1x1 llamada «*squeeze*» y dos convoluciones «*expand*», una 1x1 y otra 3x3. Su arquitectura se muestra en la figura 2.6.

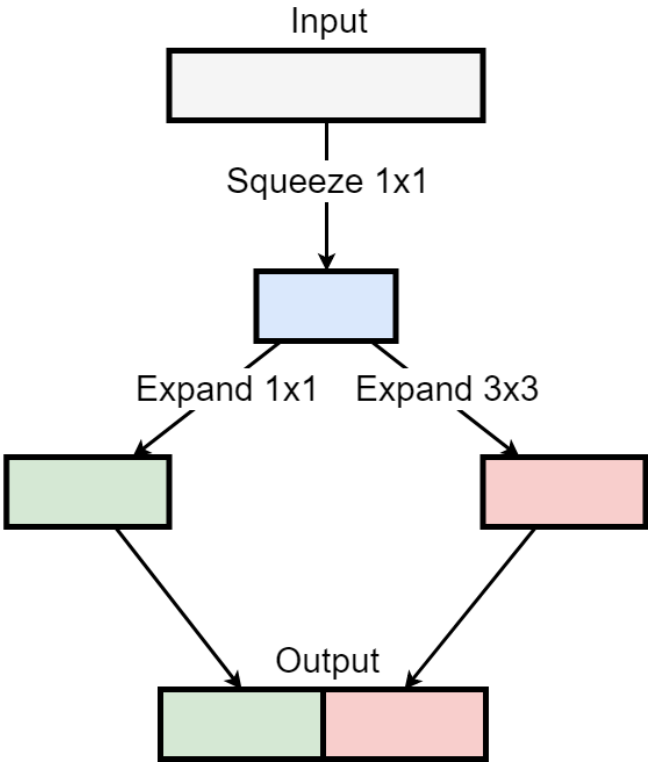


FIGURA 2.6: Arquitectura del módulo *fire*.

Como puede verse, a la entrada se le aplica una primera convolucón de 1x1 para así reducir el número de canales, luego en paralelo, se aplican convoluciones 1x1 y 3x3 para así expandir la cantidad de mapas de características a la mitad de mapas deseados en la salida. Por último, se concatena la salida de cada convolución *expand* para dar la salida del módulo. El número de operaciones de este módulo puede expresarse como:

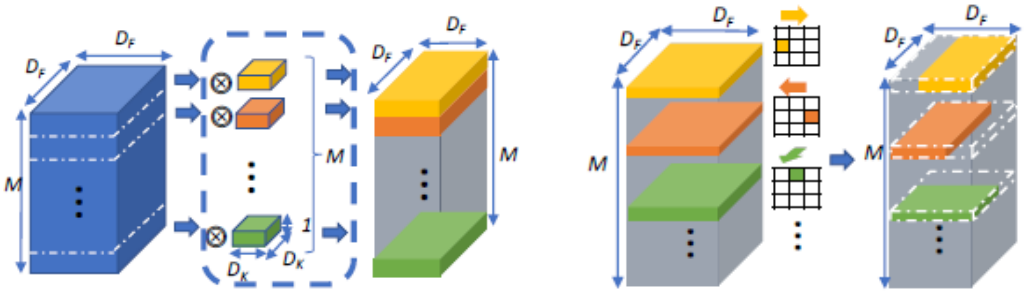
$$C(\text{fire}) = D_{l-1} b W_{l-1} H_{l-1} + b \frac{D_l}{2} W_{l-1} H_{l-1} + k_w k_h b \frac{D_l}{2} W_{l-1} H_{l-1} \qquad (2.5)$$

Con $C(\text{fire})$ como el número de operaciones necesarias para el módulo *fire*, b el número de capas de salida de la convolución *squeeze* y $(W_{l-1}, H_{l-1}, D_{l-1})$ las dimensiones de la entrada y D_l el número de mapas de características de la salida.

2.3.3. *Shift net*

Otro tipo de arquitectura que surgió con el fin de acelearar las convoluciones es la *ShiftNet*. Esta propone sustituir las convoluciones mayores a 1x1 por una

operación de *shift* seguida de una convolución 1x1; siendo esta arquitectura una forma de realizar convoluciones separables con menos operaciones aún.



(A) Convolución *depthwise*

(B) Operación de *shift*

FIGURA 2.7: Convolución *depthwise* y operación *shift*. Imágenes tomadas de [6]

En la figura 2.7 se muestra la arquitectura de la operación de *shift* frente a la convolución *depthwise*. Como puede verse, en lugar de aplicar una operación de convolución para cada mapa de características, la operación de *shift* mueve los mismos en alguna dirección. Esta operación de movimiento puede simularse con un *kernel* de convoulución con todos sus valores a 0, excepto aquellos en los que se quiere mover. En la figura 2.7 el valor en color muestra a modo de ejemplo qué peso a de ponerse a 1 para obtener el resultado que se muestra.

Si bien es una posibilidad implementar esta operación mediante una convolución, la verdadera ventaja de esta es la posibilidad de implementarla sin necesidad de aplicar ninguna operación, únicamente realizando movimientos de memoria; de modo que, el coste de la primera convolución de una convolución separable sería prácticamente nulo.

2.4. Poda de conexiones

Esta línea de investigación propone eliminar algunas conexiones del modelo con el fin de reducir tanto su tamaño como coste computacional. A esta eliminación de conexiones se le llama poda, y sobre esta línea en particular versará la investigación del presente trabajo.

En los últimos años se han desarrollado divesrso métodos de poda que actúan a diferentes niveles. [7] explica los diferentes niveles a los que se puede

podar una red neuronal. A nivel más alto, se puede podar los mapas de activación o canales de una capa de la red; en el siguiente nivel, se puede actuar a nivel de kernel, eliminando aquellos que se consideran innecesarios; y, a nivel más fino, se puede realizar la poda a nivel intra-kernel, que obliga a ser cero algunos valores de los kernels, produciendo así kernels dispersos.

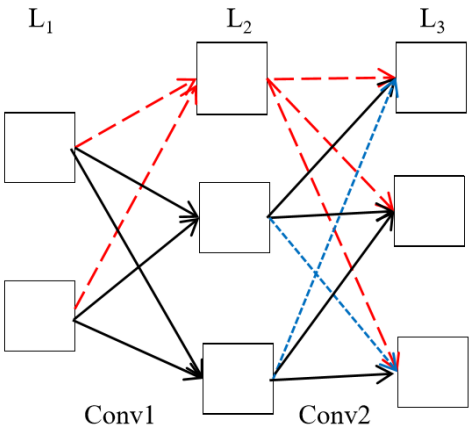


FIGURA 2.8: Poda a nivel de canal y kernel. Imágen tomada de [7]

La figura 2.8 muestra gráficamente los dos primeros niveles. Las líneas rojas muestran los kernels a eliminar si se poda el primer mapa de características de la capa L_2 y las líneas azules muestran el resultado de eliminar algunos kernels individuales.

Basados en la poda a nivel de mapas de características existen diversos trabajos como [29] o [30]. Ambos plantean una estrategia similar, esta es relizar iteraciones de poda seguidas de un entrenamiento y repetir el proceso hasta alcanzar el objetivo.

[30] plantea el problema como uno de optimización combinatoria. Siguiendo su notación, el problema se plantea de la siguiente manera:

$$\begin{aligned} &\text{MIN } |C(D|W') - C(D|W)| \\ &\text{s.a} \\ &\quad ||W'||_0 \leq B \end{aligned} \tag{2.6}$$

Donde D es el set de datos, W es el conjunto de pesos en los que convergió la red, W' es un conjunto de pesos de menor cardinalidad que W, $C(D|W)$ es el coste que se tiene al aplicar la red con los pesos W sobre los datos D y B es la

cota que limita el número de parámetros distintos de cero. En otras palabras, la función objetivo trata de encontrar el conjunto de pesos W' que a lo sumo tiene B parámetros que mejor mantiene la calidad de la red. Como se puede observar, el problema requiere $2^{|W|}$ evaluaciones para un determinado set de datos, resultando irresoluble mediante fuerza bruta para prácticamente cualquier red del estado del arte. Por esta razón, [30] propone un método *greedy* de poda iterativa seguido de un entrenamiento.

Al ser un método *greedy*, requiere de una herurística para determinar qué pesos podar en cada iteración. Es aquí donde [30] propone un criterio basado en la expansión del polinomio de Taylor y lo compara con otros métodos propuestos como poda por magnitud de pesos como propone [29], por valor de las activaciones o *Optimal Brain Damage (OBD)* [31].

Al resolver el problema de forma heurística, la calidad de la solución al problema de optimización dependerá principalmente de dos factores: el criterio de poda y estrategia de búsqueda. Por lo que respecta al criterio de poda, [30] además de utilizar una aproximación basada en la expansión de Taylor, propone ciertas modificaciones al criterio como una regularización basada en el coste computacional de calcular cada mapa de característica, siendo esta una forma de guiar el algoritmo hacia movimientos que reduzcan más rápidamente el coste computacional.

Capítulo 3

Implementación realizada

Como se ha dicho en 1.4, el objetivo principal del presente trabajo es analizar las diferentes estrategias de poda del estado del arte. Para ello, en primer lugar, se ha establecido el entorno de trabajo para poder abordar la implementación y comparación de los diferentes métodos. Seguidamente, se ha hecho una implementación y entrenamiento hasta su convergencia de la red neuronal LeNet [16] con el fin de utilizar el modelo resultante como base de comparación para las estrategias de poda a utilizar. Por último, se han diseñado una serie de experimentos con el fin de evaluar los diferentes métodos y combinaciones de los mismos para poder así analizar el comportamiento de cada uno de ellos.

3.1. Entorno de trabajo

Para la implementación se decide utilizar el *framework* Tensorflow [14] dadas las facilidades que proporciona a la hora de trabajar con redes neuronales. Sin embargo, este *framework* se basa en una estructura de grafo estático a diferencia de otros como puede ser Torch [15], con lo que aplicar algoritmos de poda presenta todo un reto dada la característica intrínseca de estos algoritmos de modificar la red. A pesar de esto, Tensorflow sigue siendo mejor opción gracias al resto de ventajas que presenta y se puede obtener el comportamiento deseado en los algoritmos de poda utilizando diferentes operaciones que obtengan el mismo resultado que el de modificar el grafo. Por estas razones, se decide seguir adelante con Tensorflow. Para faciltar la instalación se utiliza la tecnología de *Docker* [32].

Por último, para la realización de los entrenamientos se cuenta con una GPU Nvidia 1050; teniéndo presente la posibilidad de utilizar *Google Cloud Platform* en caso que fuera necesario.

3.2. Red neuronal LeNet

Para la realización de experimentos se entrena una variante de la red LeNet [16] sobre el *dataset MNIST* [8]. Este *dataset* está formado imágenes de 28x28 píxeles junto con una etiqueta para cada una de estas imágenes indicando el número escrito en la imagen. El conjunto de entrenamiento está formado por 60000 imágenes, el de *test* por 10000 y el de validación por 5000.

FIGURA 3.1: Ejemplo de varias imágenes del *dataset*. Imágenes tomada del *dataset MNIST* [8]

Para resolver el problema de clasificación de este *dataset* se plantea la arquitectura de red de la figura 3.2. Esta red está formada por dos convoluciones, dos capas de redimensión y dos capas *fully-connected*.

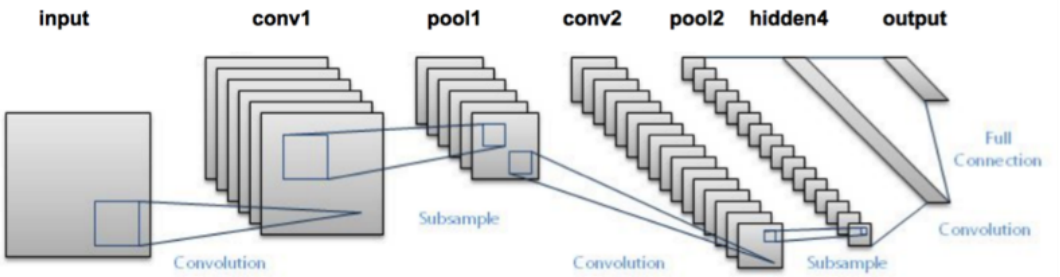


FIGURA 3.2: Arquitectura de LeNet.

	Kernel	N entradas	N salidas	Memoria	FLOPs
conv1	5x5	1	32	800	627200
conv2	5x5	32	64	51200	10035200
fc1	1x1	3136	1024	3211264	3211264
fc2	1x1	1024	10	10240	10240

TABLA 3.1: Arquitectura detallada de la red neuronal implementada.

En la tabla 3.1 se muestra los tamaños de kernel, canales de entrada, salida, elementos en memoria y FLOPs (número de operaciones de de tipo suma o MACC) de cada una de las operaciones de la red.

La red fue entrenada durante 10000 iteraciones con un *batch* de entrenamiento de 100 elementos, con un ratio de aprendizaje de 0.0001 utilizando el algoritmo de optimización *ADAM* [33] con $\beta_1 = 0{,}9$ y $\beta_2 = 0{,}999$ consiguiendo una precisión del 99.239 % sobre el conjunto de validación.

3.3. Proceso de poda

La configuración del proceso de poda puede dividirse en varias partes:

- Criterio de poda.
- Granularidad de la poda.
- Protocolo de poda.

El criterio de poda es la forma en que se decide qué conexiones se quiere podar en cada iteración de poda. La granularidad de la poda controla a qué nivel se quiere podar, ya sea a nivel de mapas de características, *kernel* o pesos individuales de cada *kernel*. Por último, el protocolo de poda es la forma en que se realiza todo el proceso, este decide en qué momento se realiza cada iteración de poda, a qué capas afecta o en qué momento se aplica un proceso de re-entrenamiento.

A su vez, el criterio de poda puede dividirse en dos:

- Criterio de evaluación.
- Criterio de selección.

El criterio de evaluación es el método que consigue evaluar cuantitativamente cada conexión, estimando de ese modo cuales son mejores candidatas a podar. El criterio de selección es aquel que decide qué conexión podar una vez que han sido evaluadas.

En cuanto a criterio de poda, en la literatura se utiliza prácticamente siempre un criterio de selección *greedy*, variando las diferentes propuestas principalmente en el criterio de evaluación. De los diferentes criterios de evaluación posibles, se estudiarán los siguientes:

- Magnitud de los pesos.
- Media de activación.
- Criterio de Taylor.

3.3.1. Diferentes criterios de evaluación

Magnitud de los pesos

Este criterio estudiará la magnitud de los pesos de los filtros tal y como propone [29]. Sea F_i^k el conjunto de pesos responsables de actuar sobre los mapas de características en la capa k - 1 para para producir el i-ésimo mapa de la capa k, entonces, la magnitud (M_i^k) del filtro F_i^k se obtendrá mediante la ecuación:

$$M_i^k = \sum_{w \in F_i^k} |w| \tag{3.1}$$

De este modo, se tiene una medida de la «importancia» del i-ésimo mapa de características de la capa k; a mayor magnitud, más importante el mapa. Si se tuviera una magnitud M_i^k de 0, entonces, el mapa i de la capa k podrá ser eliminado sin afectar a la precisión de la red.

Este criterio estudia únicamente los pesos, por lo que es independiente de los datos.

Media de activación

Otro aspecto que puede analizarse para determinar la importancia de un mapa de características es la media de las activaciones. Esto es, a la salida de

cada módulo ReLu [20] se calcula la media de los valores del mapa. Esto dará como resultado un valor indicativo de la importancia de dicho mapa de característica.

Criterio de Taylor

En el trabajo [30] se propone, siguiendo la notación del *paper*, el siguiente criterio:

$$\Theta_{TE}(z_l^k) = \left| \frac{1}{M} \sum_m \frac{\delta C}{\delta z_{l,m}^k} z_{l,m}^k \right| \tag{3.2}$$

Donde z_l^k es la activación del mapa de características k de la capa l, C es la función de costes y M es el número de elementos del mapa de características z_l^k. Esta ecuación indica que se deberá calcular la derivada de la funcion de costes respecto cada activación, multiplicar por la activación y promediar entre cada mapa de características; con lo que, aquellos mapas con una activación nula o gradiente plano obtendrán una valoración baja.

3.4. Experimentos realizados

Con el fin de analizar el comportamiento de las diferentes estrategias de poda se propone la realización de una serie de experimentos. Todos ellos se realizan utilizando la red neuronal LeNet expuesta en 3.2 ya entrenada como base.

Cada uno de estos experimentos tendrán una configuración diferente de criterio y protocolo de poda, manteniendo en común la granularidad, todos ellos se realizan a nivel de mapa de característica. La razón de utilizar únicamente esta granularidad es la traducción directa que conlleva la eliminación de un mapa de característica en eficiencia añadida, mientras que, para obtener una mayor eficiencia eliminando *kernels* individuales o incluso valores dentro de las convoluciones es necesario una implementación específica o incluso *hardware* específico que permita operaciones tales como convoluciones con *kernels* dispersos.

3.4.1. Experimentos realizados de poda sin re-entrenamiento

El primer experimento realizado es el de la aplicación de un protocolo de poda sin re-entrenamiento con diferentes criterios.

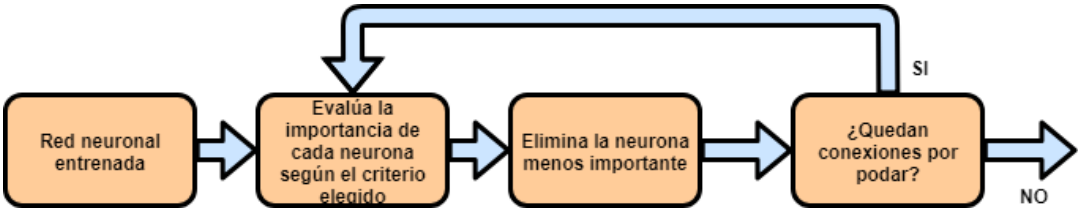


FIGURA 3.3: Diagrama del protocolo de poda sin re-entrenamiento.

En la figura 3.3 se muestra el diagrama de este protocolo. Este consiste en, partiendo del modelo entrenado, evaluar las conexiones de interés a podar, eliminar la menos importante y repetir el proceso hasta que se hayan eliminado todas las conexiones propuestas. En las siguientes sub-secciones se explicará en detalle cada experimento realizado bajo este protocolo.

Poda por capas

En primer lugar, se propone podar cada una de las capas de la red de forma individual, podando 1 mapa por iteración de poda siguiendo una estrategia *greedy* y monitorizando la precisión luego de cada iteración.

La red a utilizar contiene 1120 mapas de características, 32 de ellos resultado de la primera convolución (*conv1*), 64 resultado de la segunda (*conv2*) y 1024 resultado de la primera operación *fully-connected* (*fc1*), tal y como se observa en la tabla 3.1. Para cada una de estas capas, se realiza una poda utilizando los diferentes métodos expuestos en 3.3.1 junto con un método aleatorio a utilizar como referencia, limitando la poda a la capa seleccionada. En la sección 4.1.1 se muestran los resultados obtenidos.

Poda entre capas

Tras el análisis del comportamiento de los diferentes algoritmos para cada capa individual, se procede a aplicar los diversos criterios de poda a toda la red, de modo que, durante cada iteración de poda se podrán eliminar conexiones

de cualquiera de las capas de la red, excepto la última que corresponde con la clasificación final.

En primer lugar, se realizará una poda entre todas las capas de la red utilizando los criterios sin aplicar ningún tipo de normalización. Resultados de este experimento se presentan en 4.1.2.

Es de esperar que el desempeño de los métodos sea peor que en el caso de la aplicación a capas individuales dado que es posible que la magnitud de algunos criterios deba ser normalizada. Por lo que, se realizará una comparativa del desempeño de cada método sin normalizar y bajo la norma l_1 y l_2, para luego, una vez identificada la normalización requerida para obtener el mejor comportamiento, realizar una comparativa de cada método bajo su mejor normalización. Resultados de este experimento se presentan en 4.1.2.

3.4.2. Experimentos realizados de poda con re-entrenamiento

Tras la realización de la poda por capas individuales e identificación de la necesidad de una regularización para un correcto desempeño si se podan varias capas, se realizará una poda siguiendo el protocolo con re-entrenamiento.

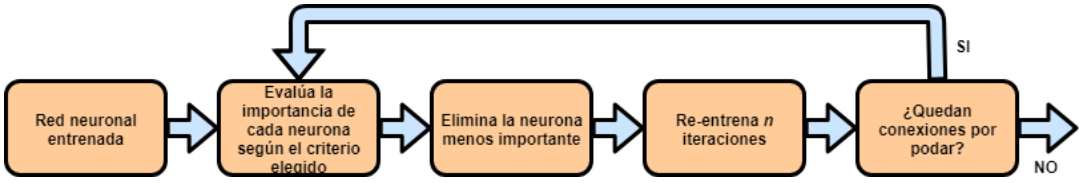


FIGURA 3.4: Diagrama del protocolo de poda con re-entrenamiento.

En la figura 3.4 se muestra el diagrama de este protocolo; partiendo del modelo ya entrenado, se evalúan las conexiones propuestas a podar, se poda la menos importante, se re-entrena las conexiones supervivientes durante un número determinado de iteraciones y se repite el proceso. En las siguientes sub-secciones se detallará cada uno de los experimentos realizados con este protocolo.

Estimación de número de iteraciones de re-entreno necesarias

En primer lugar, cabe definir el número óptimo de iteraciones de re-entreno necesarias para que este protocolo de poda sea efectivo; por lo que, se estudiará el comportamiento de cada método de poda al variar el número de iteraciones de re-entreno.

Este estudio se realizará analizando primero el efecto en una poda para cada capa individual, para luego repetir el estudio aplicado a todo el modelo en su conjunto. Para la realización de este experimento se aplicará la mejor regularización encontrada en en el experimento 3.4.1. Los resultados de esta estimación se encuentran en la sección 4.2.1.

Comparativa de los diferentes criterios bajo el mejor número de iteraciones de re-entreno encontrado

Tras encontrar el número óptimo de iteraciones de re-entreno a aplicar en un protocolo con re-entreno, se realizará la comparativa del desempeño de cada método de poda para así verificar si se sigue manteniendo como mejor criterio de poda que el encontrado bajo el protocolo 3.4.1.

En primer lugar se realizará la comparativa de poda por capas individuales y luego con una poda aplicada a toda la red. Resultados de esta comparativa se encuentran en 4.2.2.

Análisis del efecto de la regularización en el protocolo de poda con re-entreno

Hasta el momento, los experimentos de este protocolo aplicados a toda la red se han realizado utilizando la mejor regularización encontrada en el experimento 3.4.1. Para verificar si el hecho de realizar un re-entreno puede evitar la necesidad de aplicar una normalización, se realizará una comparativa del efecto de aplicar este protocolo con y sin normalización para cada uno de los criterios de poda. Esta comparativa se encuentra en la sección 4.2.3.

3.4.3. Experimentos realizados de poda con re-entrenamiento exclusivos a cada capa

Tras realizar los experimentos relativos a un protocolo con re-entreno, se propone modificar el mismo, aplicando dicho re-entrenamiento únicamente a la capa podada.

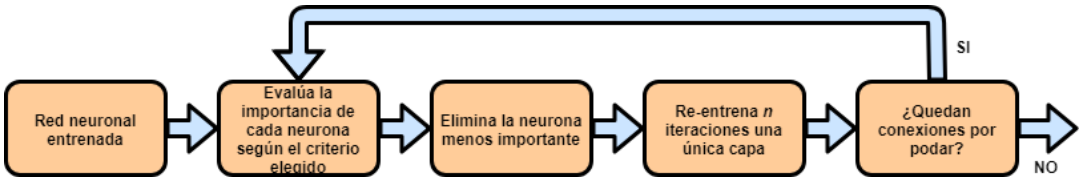


FIGURA 3.5: Diagrama del protocolo de poda con re-entrenamiento úna sola capa.

En la figura 3.5 se muestra el diagrama de dicho protocolo. Se realizarán experimentos de aplicar este protocolo a cada capa individual. Resultados de dichos experimentos así como comparativa para verificar si se sigue manteniendo el mismo criterio como mejor criterio de poda se muestran en la sección 4.3.

3.4.4. Experimentos realizados de poda con re-entrenamiento sin re-cálculo del criterio en cada iteración

Como último protocolo de poda analizado, se propone realizar una modificación al protocolo con re-entrenamiento del experimento 3.4.2 en pro de la eficiencia. Se propone no re-calcular el criterio de poda en cada iteración, salvo en la primera.

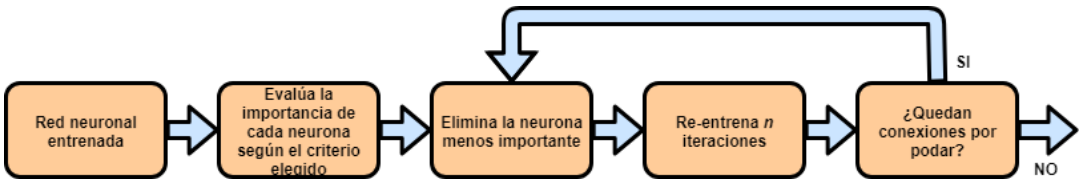


FIGURA 3.6: Diagrama del protocolo de poda con re-entrenamiento sin re-cálculo del criterio.

En la figura 3.6 se muestra el diagrama de este protocolo. Resultados de la aplicación de este protocolo a cada capa individual de la red así como a todo el modelo en su conjunto se muestran en la sección 4.4.

3.4.5. Comparación de los mejores métodos de cada protocolo de poda

Al finalizar los experimentos propuestos, se tomará de cada protocolo de poda el método con mejor desempeño para así comparar los diferentes protocolos. En primer lugar se realizará una comparación de los métodos aplicados a cada capa individual para luego comparar el desempeño al podar el modelo en en su conjunto. De este modo, se podrá tener una valoración final de cual es el mejor criterio de poda encontrado y bajo qué protocolo ha dado su mejor desempeño. Resultados de esta comparativa se encuentran en la sección 4.5.

3.4.6. Resultados finales

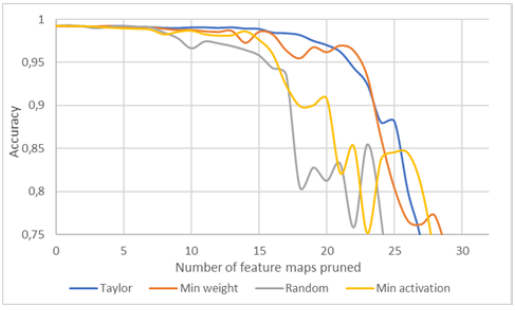
Por último, para finalizar, se agregará los resultados finales, comparando numéricamente el resultado los mejores algoritmos obtenidos con el modelo base, así como el resultado de entrenar un modelo reducido desde cero. Estos resultados se encuentran en la sección 4.6.

Capítulo 4

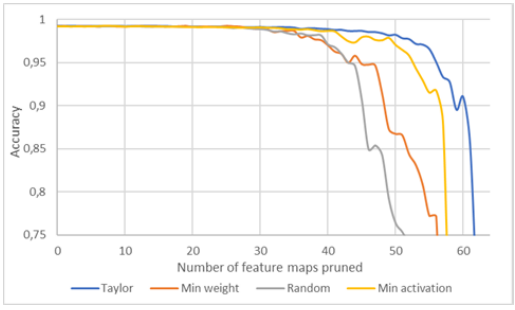
Resultados

4.1. Poda sin re-entrenamiento

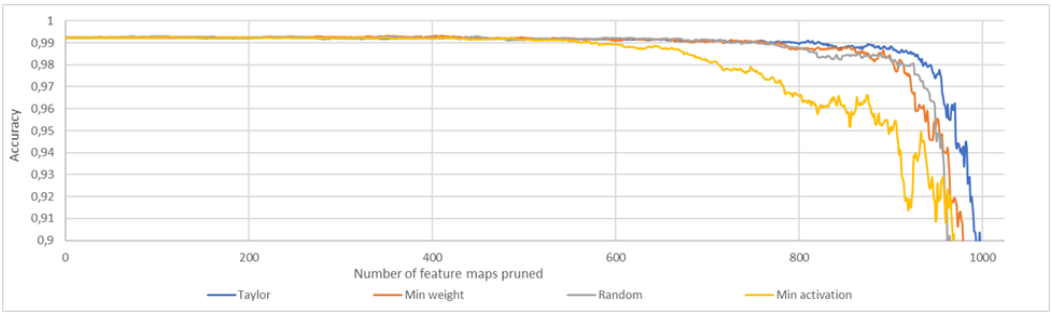
4.1.1. Poda sin re-entreno por capas



(A) Poda de la capa *conv1*



(B) Poda de la capa *conv2*



(C) Poda de la capa *fc1*

FIGURA 4.1: Poda de las capas *conv1*, *conv2* y *fc1* con diferentes métodos

En la figura 4.1 se muestra el resultado de eliminar una conexión por iteración aplicando la poda únicamente a las capas *conv1*, *conv2* y *fc1* por separado. Como se puede observar, una poda realizada en las primeras capas tendrá mayor repercusión negativa que una poda en las últimas. Esto concuerda con la teoría de que en las primeras capas se aprenden las estructuras

más básicas, con lo que, de no poder representar elementos tan simples como un borde es de esperar que el modelo no consiga una alta precisión.

De estos resultados se puede verificar que el criterio de evaluación con mejor desempeño ha sido el de *Taylor*, superado en apenas alguna ocasión en el momento de podar la capa *conv1*.

4.1.2. Poda sin re-entreno en toda la red

Poda sin re-entreno en toda la red sin regularización

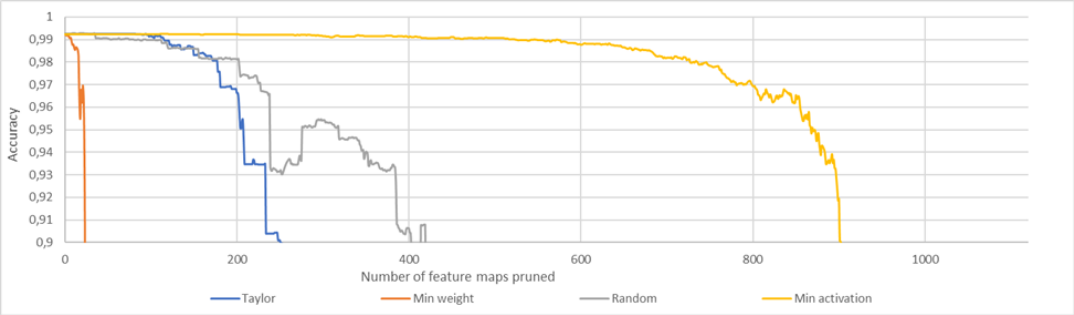
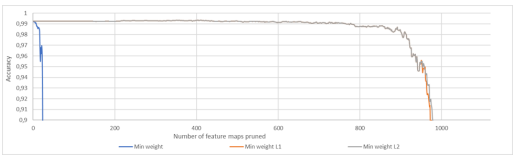


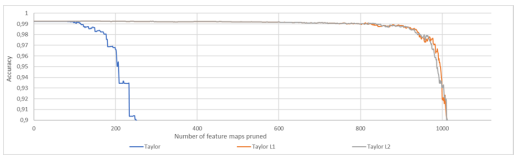
FIGURA 4.2: Poda de todas las capas de la red con diferentes métodos.

En la figura 4.2 se muestra el resultado de podar toda la red con los diferentes métodos. Como se puede observar, se ha tenido un desempeño mucho peor que en el caso de aplicar una poda a cada capa individual, teniendo un desempeño mucho peor incluso que una poda aleatoria. Esto se debe a la falta de una regularización en los criterios aplicados al tener cada capa una magnitud diferente tanto en sus activaciones como pesos.

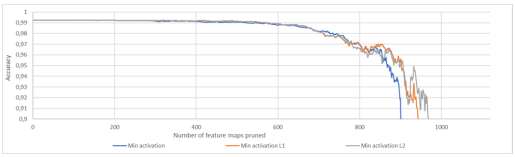
Poda sin re-entreno en toda la red con regularización



(A) Poda de todas las capas de la red con el método de mínimo peso.



(B) Poda de todas las capas de la red con el método de *Taylor*.



(C) Poda de todas las capas de la red con el método de mínima activación.

FIGURA 4.3: Comparativa del uso de diferentes regularizaciones para cada método.

En la figura 4.3 se muestra el efecto de aplicación de diferentes regularizaciones a cada método de poda. Como puede observarse, el desempeño de los distintos métodos de poda mejoró considerablemente, considerando como mejor regularización la l_2 para la normalización de las magnitudes entre las diferentes capas.

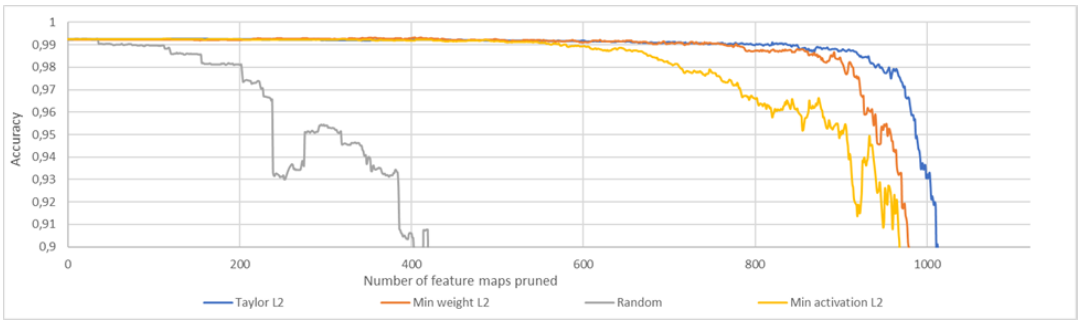


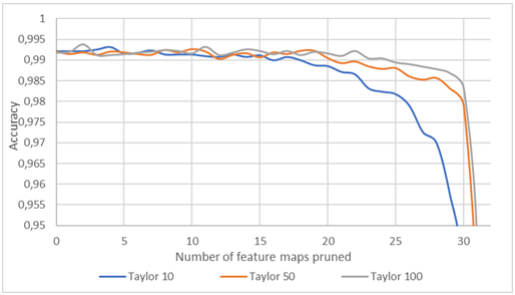
FIGURA 4.4: Poda de todas las capas de la red con diferentes métodos con regularización l_2.

En la figura 4.4 se muestra la comparativa del desempeño de los diferentes métodos aplicados a toda la red con regularización l_2. Como puede observarse, al igual que en el caso de poda por capas, el criterio de *Taylor* da el mejor desempeño.

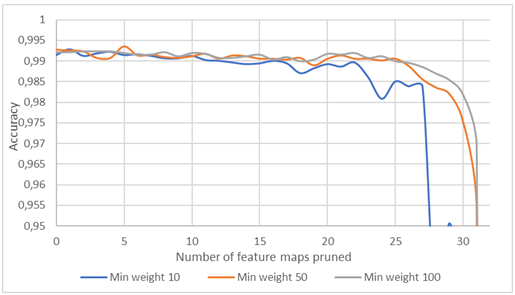
4.2. Poda con re-entrenamiento

4.2.1. Estimación de número necesario de iteraciones de re-entrenamiento por iteración de poda

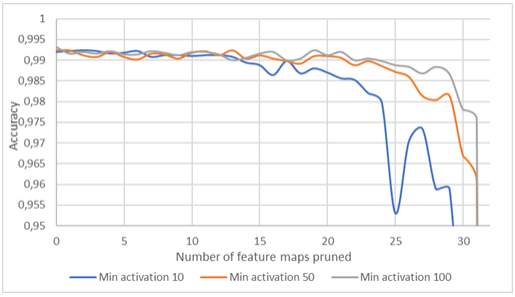
Estimación para una poda de únicamente la capa *conv1*



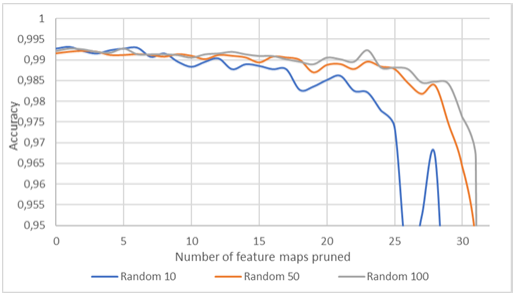
(A) Poda con criterio de *Taylor*.



(B) Poda con criterio de mínimo peso.



(C) Poda con criterio de mínima activación.

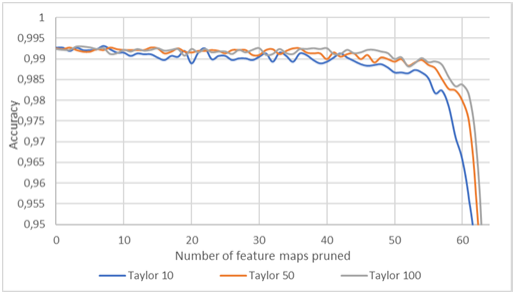


(D) Poda con criterio aleatorio.

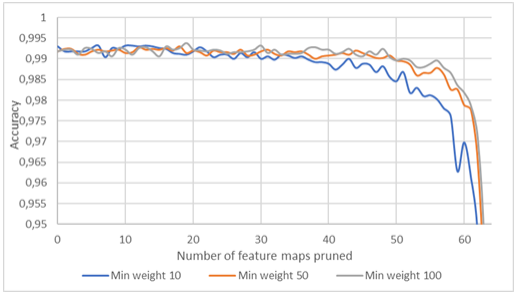
FIGURA 4.5: Poda de la capa *conv1* con diferentes métodos re-entrenando 10, 50 y 100 iteraciones.

En la figura 4.5 se muestra el resultado de aplicar los distintos criterios a la capa *conv1* re-entrenando 10, 50 y 100 iteraciones. Como puede observarse, todos los criterios mejoran su desempeño a más iteraciones de re-entreno realizadas. Esta mejora del desempeño resulta pronunciada entre la realización de 10 y 50 iteraciones, mientras que al comparar entre realizar 50 o 100 iteraciones, se puede decir que es mejor realizar 100 antes que 50, teniendo algunos casos en que el resultado es muy similar, o incluso el mismo.

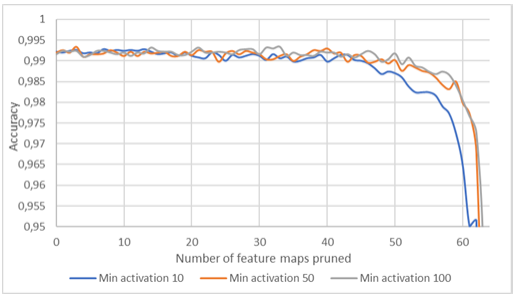
Estimación para una poda de únicamente la capa *conv2*



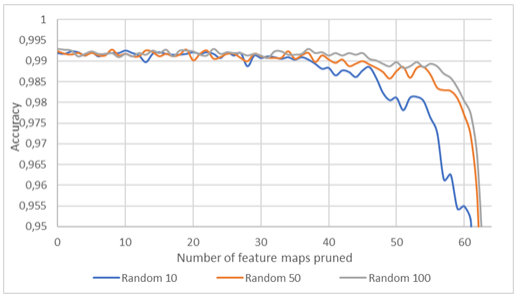
(A) Poda con criterio de *Taylor*.



(B) Poda con criterio de mínimo peso.



(C) Poda con criterio de mínima activación.

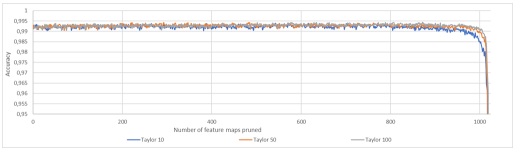


(D) Poda con criterio aleatorio.

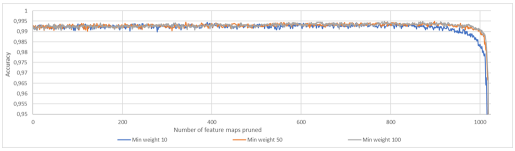
FIGURA 4.6: Poda de la capa *conv2* con diferentes métodos re-entrenando 10, 50 y 100 iteraciones.

En la figura 4.6 se muestra el resultado de aplicar los distintos criterios a la capa *conv2* re-entrenando 10, 50 y 100 iteraciones. Los resultados son similares a los obtenidos para la capa **??**, mostrando aún menos distancia entre la realización de 50 y 100 iteraciones.

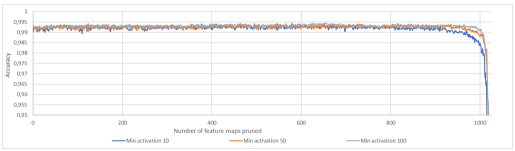
Estimación para una poda de únicamente la capa *fc1*



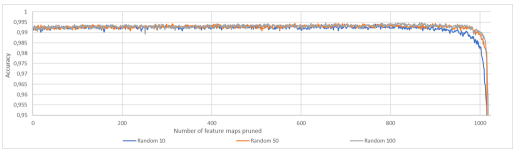
(A) Poda con criterio de *Taylor*.



(B) Poda con criterio de mínimo peso.



(C) Poda con criterio de mínima activación.

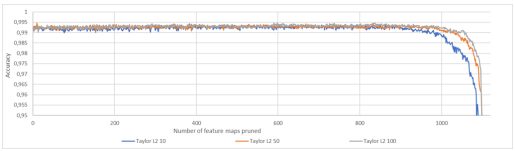


(D) Poda con criterio aleatorio.

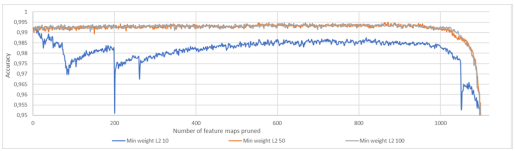
FIGURA 4.7: Poda de la capa *fc1* con diferentes métodos re-entrenando 10, 50 y 100 iteraciones.

En la figura 4.7 se muestra el resultado de aplicar los distintos criterios a la capa *fc1* re-entrenando 10, 50 y 100 iteraciones. Al igual que en el caso de la capa *conv2*, se puede decir que la distancia entre aplicar 50 y 100 iteraciones es aún menor. Por lo que, se puede deducir que a más profunda la capa que se pode, menos efecto tendrá un re-entreno con más de 100 capas.

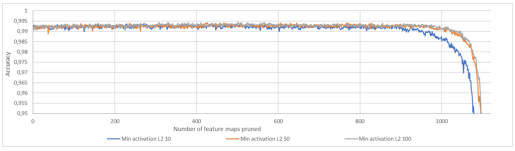
Estimación para una poda de toda la red



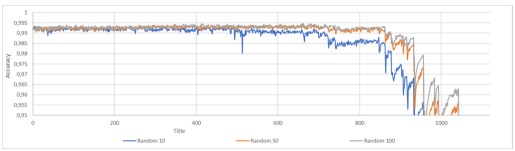
(A) Poda con criterio de *Taylor*.



(B) Poda con criterio de mínimo peso.



(C) Poda con criterio de mínima activación.



(D) Poda con criterio aleatorio.

FIGURA 4.8: Poda de toda la red con diferentes métodos re-entrenando 10, 50 y 100 iteraciones.

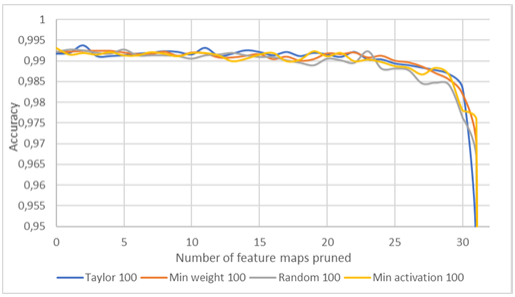
En la figura 4.8 se muestra el resultado de aplicar los distintos criterios a toda la red re-entrenando 10, 50 y 100 iteraciones. Al igual que en el caso aplicado por capas diferentes, se puede verificar que se tiene el mejor desempeño con

100 iteraciones, mostrando una pronunciada diferencia entre 10 y 50, y un resultado muy similar, rozando el límite de la diferenciación de cual es mejor, entre aplicar 50 o 100 iteraciones de re-entreno.

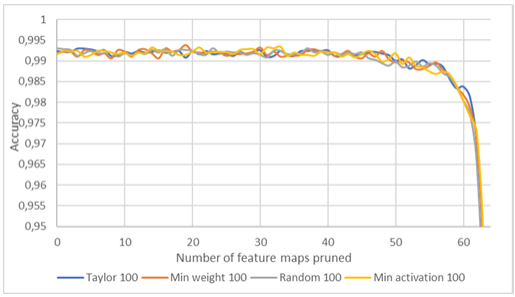
Tras la realización de estos experimentos, se concluye que el número óptimo de iteraciones de re-entreno a aplicar en este caso es 100; sin necesidad de explorar la posibilidad de aumentar este número, tanto por razones de eficiencia como de eficacia.

4.2.2. Estimación del mejor criterio para una poda con re-entreno

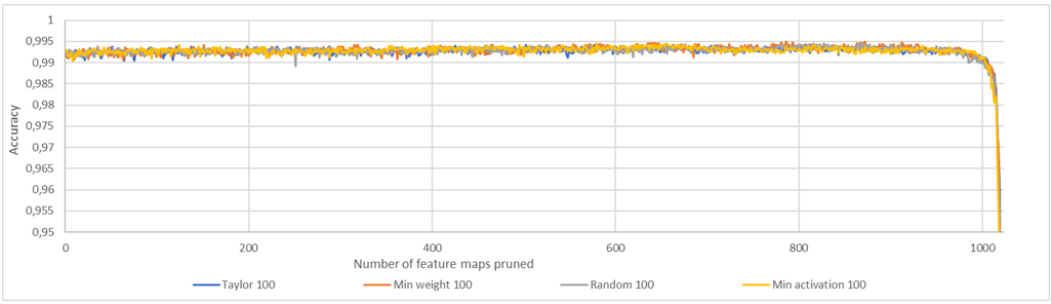
Estimación del mejor criterio para una poda con re-entreno por capas



(A) Poda de la capa *conv1*



(B) Poda de la capa *conv2*



(C) Poda de la capa *fc1*

FIGURA 4.9: Poda de las capas *conv1*, *conv2* y *fc1* con diferentes métodos re-entrenando 100 iteraciones

En la figura 4.9 se muestra la comparativa de realizar una poda por capas individuales con 100 iteraciones de re-entreno. Como se observa, todos los métodos dan resultados muy similares, incluso uno aleatorio, por lo que, no se puede establecer de forma concluyente un único método como mejor que los demás. Esto demuestra que, en este caso, el re-entreno, en el caso de realizar una poda a una capa individual, pesa más que el criterio de poda elegido.

Estimación del mejor criterio para una poda con re-entreno de toda la red

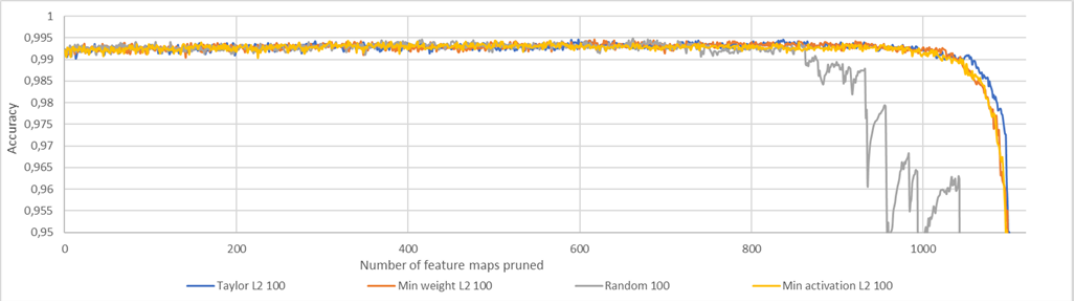
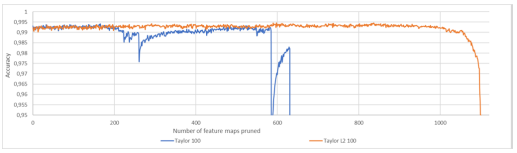


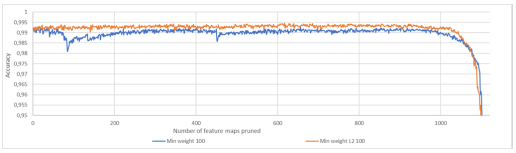
FIGURA 4.10: Poda de todas las capas de la red con diferentes métodos re-entrenando 100 iteraciones.

En la figura 4.10 se muestra la comparativa de realizar una poda a todo el modelo con 100 iteraciones de re-entreno. En este caso si que se puede definir que método ha tenido mejor desempeño, siendo este el criterio de *Taylor*, al igual que en el caso de poda sin re-entreno.

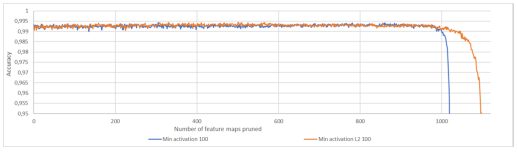
4.2.3. Análisis del efecto de la regularización en una poda con re-entrenamiento



(A) Efecto de la regularización con el método de *Taylor*.



(B) Efecto de la regularización con el método de mínimo peso.



(C) Efecto de la regularización con el método de mínima activación.

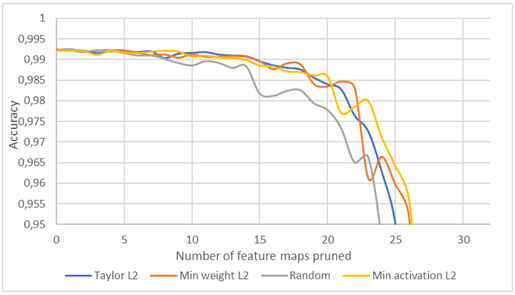
FIGURA 4.11: Comparativa del uso o no de una regularización.

En la figura 4.11 se analiza el efecto del uso o no de una regularización en una poda con re-entreno con el fin de verificar si es posible que un re-entreno elimine la necesidad de utilizar una normalización. Como se puede verificar, en la mayor parte de los casos esto no ocurre, una regularización sigue

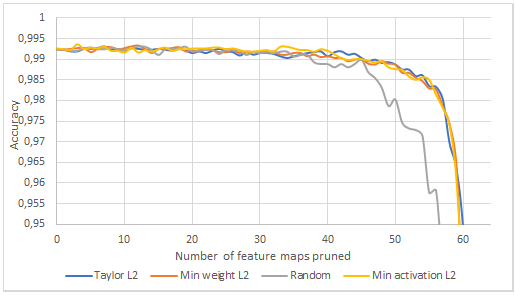
siendo necesaria para un correcto desempeño de los criterios de poda, excepto para el criterio de mínimo peso. Esto es de esperar, pues, durante un re-entrenamiento se modifican directamente los pesos del modelo, por lo que entre cada iteración es posible que se adapten para evitar una regularización en caso de utilizar un método de mínimo peso; sin embargo, el resto de métodos utilizan información diferente a parte de los pesos de la red, por lo que si requieren una normalización.

Tras estos resultados, se concluye que una normalización sigue siendo necesaria, incluso cuando se utiliza el criterio de mínimo peso.

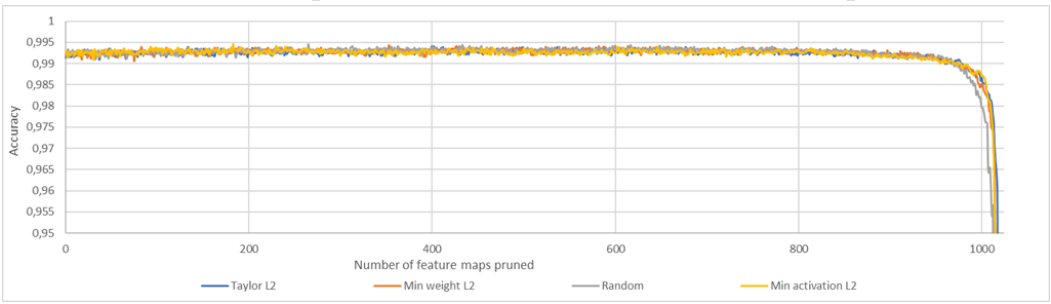
4.3. Poda con re-entreno exclusivo a cada capa podada



(A) Poda de la capa *conv1*



(B) Poda de la capa *conv2*



(C) Poda de la capa *fc1*

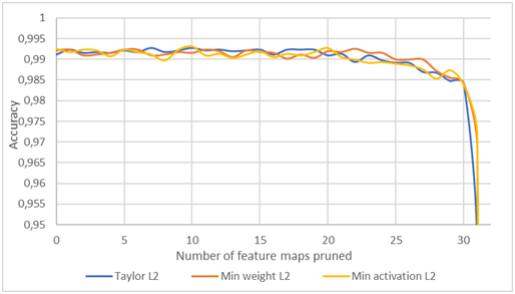
FIGURA 4.12: Poda de las capas *conv1*, *conv2* y *fc1* con diferentes métodos re-entrenando únicamente la capa podada

En la figura 4.12 se muestra el resultado de aplicar una poda a cada capa individual y re-entrenando únicamente la capa podada. Al igual que en los resultados 4.2.2, no es posible establecer un claro mejor criterio, sin embargo, realizar una poda aleatoria seguida de un re-entreno ya no garantiza un

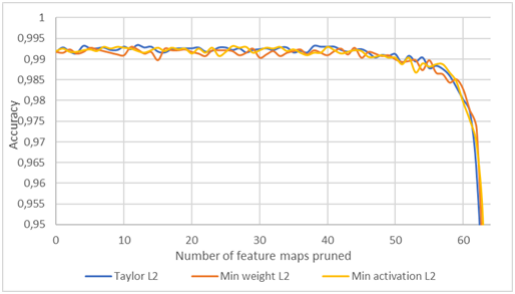
buen desempeño. Esto demuestra que en el caso de 4.2.2 se conseguían buenos resultados independientemente del criterio utilizado al compensar las conexiones mal podadas de una capa con adaptaciones de otras capas.

4.4. Poda con re-entreno sin re-cálculo del criterio

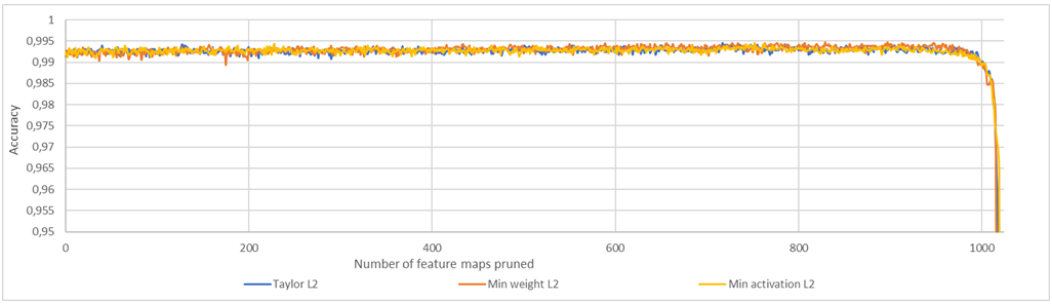
4.4.1. Poda por capas con re-entreno sin re-cálculo del criterio



(A) Poda de la capa *conv1*



(B) Poda de la capa *conv2*



(C) Poda de la capa *fc1*

FIGURA 4.13: Poda de las capas *conv1*, *conv2* y *fc1* con diferentes métodos calculados únicamente en la primera iteración

En la figura 4.13 se muestra el resultado de aplicar una poda a cada capa individual sin re-calcular el criterio en cada iteración. Como es de esperar, al igual que en el caso de 4.2.2, no puede establecerse un claro mejor criterio al pesar más el re-entreno que la selección del método en este caso.

4.4.2. Poda en toda la red con re-entreno sin re-cálculo del criterio

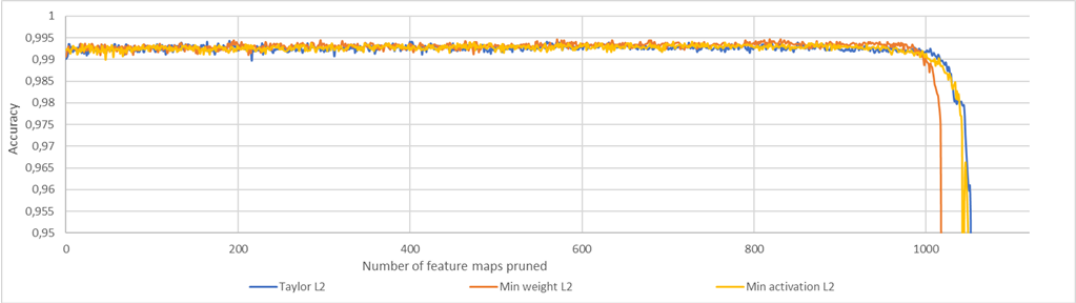
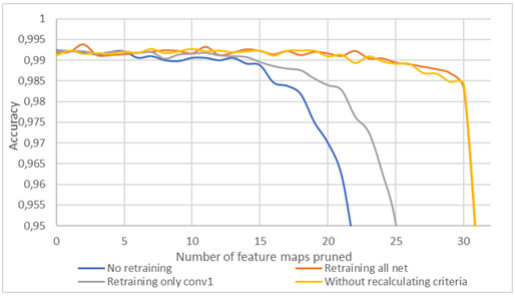


FIGURA 4.14: Poda de todas las capas de la red con diferentes métodos calculados únicamente en la primera iteración.

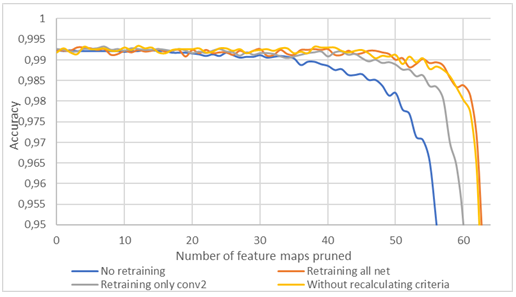
En la figura 4.14 se muestra el resultado de aplicar una poda a toda la red sin re-calcular el criterio en cada iteración. En este caso si que es posible establecer un mejor criterio, siendo este, una vez más, el crietrio de *Taylor*, seguido por poco por el de mínima activación.

4.5. Comparación de los mejores métodos de cada protocolo de poda

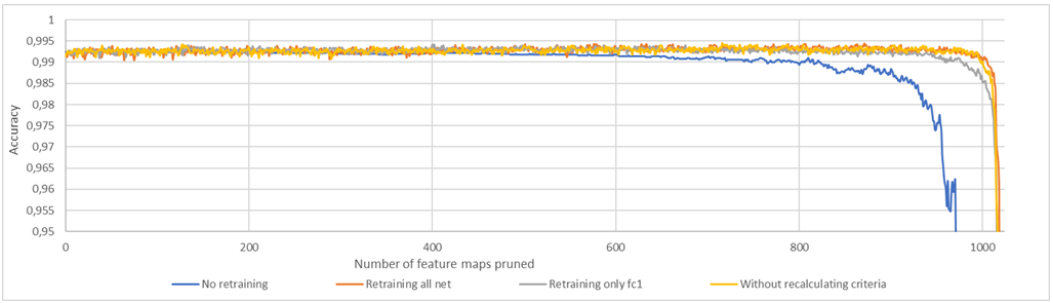
4.5.1. Comparación de los métodos aplicados por capa



(A) Poda de la capa *conv1*



(B) Poda de la capa *conv2*



(C) Poda de la capa *fc1*

FIGURA 4.15: Poda de las capas *conv1*, *conv2* y *fc1* con el criterio de *Taylor* con diferentes protocolos de poda.

En la figura 4.15 se muestra el resultado de aplicar una poda a cada capa individual con diferentes protocolos. Se puede observar que el mejor desempeño lo da el protocolo de re-entrenamiento de toda la red, seguido muy de cerca por el protocolo que no re-calcula el criterio. Con lo que, se puede concluir que en este modelo, para el caso de una poda por capa individual un re-entramiento en toda la red luego de cada iteración de poda es necesario para un buen desempeño.

4.5.2. Comparación de los métodos aplicados a toda la red

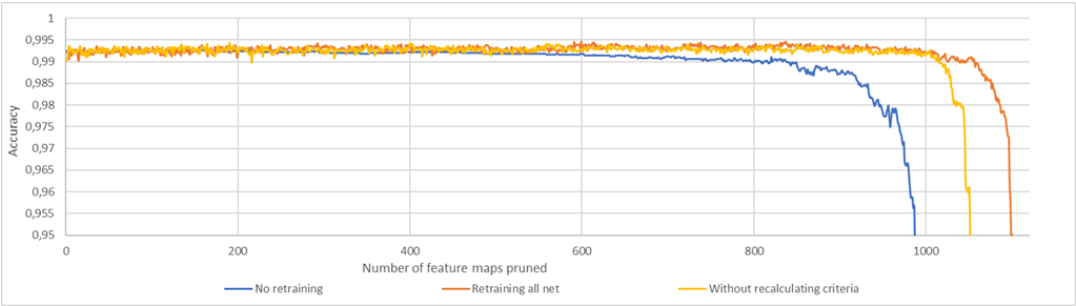


FIGURA 4.16: Poda de todas las capas de la red con el criterio de *Taylor* con diferentes protocolos de poda.

En la figura 4.16 se muestra el resultado de aplicar una poda a toda la red con diferentes protocolos. En este caso si que se ve clara la diferencia entre re-calcular o no el criterio, siendo el mejor protocolo el de re-entreno de toda la red con re-cálculo del criterio en cada iteración.

4.6. Resultados finales

Tras la realización de todos los experimentos, se puede concluir que el mejor algoritmo de poda encontrado es una poda con re-entreno de al menos 100 iteraciones por cada iteración de poda, siguiendo el criterio de *Taylor* para evaluar la importancia de cada neurona y permitiendo re-entrenar todas las conexiones del modelo.

Una vez identificado este algoritmo de poda, se analiza el resultado final de realizar una poda del modelo marcando como límite inferior una precisión del 99 %. El resultado obtenido se compara con el de entrenar un modelo de las mismas características desde cero.

	Pérdida de precisión	Compresión	Aceleración
Modelo base	0 %	x1	x1
Modelo base podado	0.239 %	x80	x7
Modelo entrenado de cero	0.839 %	x80	x7

TABLA 4.1: Comparación de un modelo podado y uno compacto entrenado desde cero con el modelo base.

En la tabla 4.1 se muestra la comparativa de aplicar un método de poda a un modelo entrenado frente a entrenar un modelo del mismo tamaño desde cero. Para obtener el modelo podado, se aplicó una poda con un protocolo con re-entrenamiento siguiendo el criterio de evaluación de *Taylor*, tal y como se verificó que es el mejor protocolo de poda para este tipo de modelo, hasta alcanzar una precisión del 99 %. Tras obtener la nueva arquitectura, se realizó un entrenamiento desde cero tal y como se realizó la primera vez para el modelo base detallado en 3.2 con la nueva arquitectura obtenida.

El nuevo modelo, utiliza tan solo un 1,25 % de los pesos del modelo original y realiza tan solo el 14 % de operaciones respecto al original. Es de notar la mayor ganancia en compresión respecto a aceleración. Esto se debe a que la mayor parte de las conexiones fueron podadas de la capa *fc1*, una capa de tipo FC, más que de las capas convolucionales, aquellas responsables del principal consumo de operaciones tal y como se verifica en la tabla 3.1.

Tras comparar la precisión del modelo entrenado desde cero con el modelo podado, se puede verificar que el primero no consigue alcanzar la precisión del segundo, verificando la teoría de la necesidad de un modelo denso para el aprendizaje.

Capítulo 5

Conclusiones y líneas futuras

5.1. Conclusiones

Se ha realizado una investigación de diversas técnicas del estado del arte para la mejora de la eficiencia tanto en consumo de memoria como realización de número de operaciones de las redes neuronales convolucionales centrándose en las técnicas basadas en poda de conexiones. Tras la realización de dicha investigación se realizó la implementación y entrenamiento de una red neuronal a utilizar como modelo base para analizar el desempeño de las diferentes técnicas de poda, para así estudiar su comportamiento a medida que el modelo se reduce tanto en tamaño como en número de operaciones y verificar qué combinación de métodos y protocolos de poda se adecua mejor al modelo.

Luego del diseño y realización de los experimentos necesarios para dicho análisis, se consiguió verificar la eficiencia de dichos métodos de poda, consiguiendo resultados positivos tanto en reducción de tamaño del modelo como en aceleración del mismo al reducir el número de operaciones necesarias para su inferencia.

Con esto se comprueba la eficacia de estos métodos, muy dependientes de su configuración y forma de aplicación, que permiten la reducción de modelos grandes e imposibles de implementar en arquitecturas de recursos limitados o de utilizar en aplicaciones que requieren un tratamiento de alto rendimiento a otros modelos mucho más eficientes que sí pueden ser implementados en dichas condiciones a costa de una pequeña pérdida de precisión.

5.2. Líneas futuras

Como línea futura se propone extender la aplicación de estos métodos a arquitecturas más complejas tanto en tamaño como en diseño de la arquitectura. Varios retos se presentarán en la aplicación de estos métodos a dichas arquitecturas, el primero será sin duda el coste computacional de la aplicación de los mismos. Si bien son algoritmos diseñados para mejorar la eficiencia del modelo, la aplicación de los mismos requiere un elevado coste computacional en muchos casos incluso para redes neuronales pequeñas; por lo que, será de esperar que el solo hecho de aplicar una poda a un modelo más profundo sea todo un reto.

Como siguiente paso natural a dar en este proceso se propone aplicar estos métodos a arquitecturas más complejas, tales como la arquitectura de ResNet [19] que el propio diseño de su arquitectura condiciona el modo en que se han de podar las conexiones, por lo que diferentes modos de podar conexiones han de diseñarse.

Bibliografía

[1] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," *ArXiv e-prints*, Mar. 2016.

[2] 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, pp. 1929–1958, 2014.

[3] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," *CoRR*, vol. abs/1510.00149, 2015.

[4] S. Zagoruyko and N. Komodakis, "Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer," *CoRR*, vol. abs/1612.03928, 2016.

[5] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017.

[6] B. Wu, A. Wan, X. Yue, P. H. Jin, S. Zhao, N. Golmant, A. Gholaminejad, J. Gonzalez, and K. Keutzer, "Shift: A zero flop, zero parameter alternative to spatial convolutions," *CoRR*, vol. abs/1711.08141, 2017.

[7] S. Anwar, K. Hwang, and W. Sung, "Structured pruning of deep convolutional neural networks," *CoRR*, vol. abs/1512.08571, 2015.

[8] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010.

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

[10] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 (canadian institute for advanced research),"

[11] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba, "Scene parsing through ade20k dataset," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[12] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014.

[13] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22Nd ACM International Conference on Multimedia*, MM '14, (New York, NY, USA), pp. 675–678, ACM, 2014.

[14] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. J. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Józefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. G. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. A. Tucker, V. Vanhoucke, V. Vasudevan, F. B. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *CoRR*, vol. abs/1603.04467, 2016.

[15] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[16] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, Nov 1998.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, (USA), pp. 1097–1105, Curran Associates Inc., 2012.

[18] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

[20] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on*

International Conference on Machine Learning, ICML'10, (USA), pp. 807–814, Omnipress, 2010.

[21] D. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *CoRR*, vol. abs/1511.07289, 2015.

[22] G. Zhao, Z. Zhang, J. Wang, and H. Guan, "Training better cnns requires to rethink relu," *CoRR*, vol. abs/1709.06247, 2017.

[23] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *CoRR*, vol. abs/1505.00853, 2015.

[24] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015.

[25] S. Ruder, "An overview of gradient descent optimization algorithms," *CoRR*, vol. abs/1609.04747, 2016.

[26] J. Van Leeuwen, "On the construction of huffman trees.," pp. 382–410, 01 1976.

[27] G. Hinton, O. Vinyals, and J. Dean, "Distilling the Knowledge in a Neural Network," *ArXiv e-prints*, Mar. 2015.

[28] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size," *CoRR*, vol. abs/1602.07360, 2016.

[29] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *CoRR*, vol. abs/1608.08710, 2016.

[30] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient transfer learning," *CoRR*, vol. abs/1611.06440, 2016.

[31] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems 2* (D. S. Touretzky, ed.), pp. 598–605, Morgan-Kaufmann, 1990.

[32] D. Merkel, "Docker: Lightweight linux containers for consistent development and deployment," *Linux J.*, vol. 2014, Mar. 2014.

[33] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.