

Trabajo fin de grado

Calibración de Combinaciones de Redes Neuronales Profundas
Convolucionales



Jorge Hernán Urgel

Escuela Politécnica Superior
Universidad Autónoma de Madrid
C\Francisco Tomás y Valiente nº 11

UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR



Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

**Calibración de Combinaciones de Redes
Neuronales Profundas
Convolucionales**

Autor: Jorge Hernán Urgel

Tutor: Juan Maroñas Molano

Ponente: Daniel Ramos Castro

junio 2021

Todos los derechos reservados.

Queda prohibida, salvo excepción prevista en la Ley, cualquier forma de reproducción, distribución comunicación pública y transformación de esta obra sin contar con la autorización de los titulares de la propiedad intelectual.

La infracción de los derechos mencionados puede ser constitutiva de delito contra la propiedad intelectual (*arts. 270 y sgts. del Código Penal*).

DERECHOS RESERVADOS

© 3 de Noviembre de 2017 por UNIVERSIDAD AUTÓNOMA DE MADRID
Francisco Tomás y Valiente, nº 1
Madrid, 28049
Spain

Jorge Hernán Urgel
Calibración de Combinaciones de Redes Neuronales Profundas
Convolucionales

Jorge Hernán Urgel
C\ Francisco Tomás y Valiente N° 11

A mi hermano

*El asedio con valor tuvimos que soportar...
y en la historia se grabó*

AGRADECIMIENTOS

En un momento tan importante como es presentar un Trabajo de Fin de Grado, es inevitable acordarse de las personas que han estado ahí a lo largo de estos años (sobre todo estos cuatro últimos). No podría empezar esto sin acordarme del grupo de compañeros que me han acompañado a lo largo de este camino: Adrián, Marcos, Jaime, Jorge... Poco puedo decir, solo dar las gracias por estar ahí, ayudarme cuando lo he necesitado y por haber sacado lo mejor de mí para poder llegar a la meta. Las amistades que me llevo con vosotros son más valiosas que cualquier otro logro que haya obtenido o que esté por llegar. Este equipo es para siempre. Gracias.

Las siguientes personas de las que me acuerdo, que no menos importantes, mi familia. Por darme la oportunidad de tener unos estudios, por hacer mi día a día más fácil, por no soltar mi mano en ningún momento y por haberme hecho llegar a ser lo que soy a día de hoy. Pero sobre todo, mencionar a mi hermano, por aguantar conmigo todo y más. El verdadero capitán de este barco que está a punto de llegar a puerto. Como dicen en nuestra tierra, ¡rasmia!

No quiero cerrar esta ronda de agradecimientos sin mencionar a mi tutor, Juan Maroñas, y a mi ponente Daniel Ramos. Gracias por darme la oportunidad de realizar este trabajo, cuando ya pensaba que no encontraría ninguno, me ofrecisteis la posibilidad de realizar uno que además me entusiasmaba. No solo debo agradecerlos eso, sino las facilidades que me habeis dado, la ayuda para entender conceptos que para mí eran absolutamente desconocidos, y la disponibilidad de las máquinas con que he podido llevar a cabo este trabajo.

RESUMEN

El objetivo de este Trabajo de Fin de Grado es púramente descriptivo. Se centrará en estudiar el comportamiento de una topología particular de redes neuronales profundas: las redes neuronales convolucionales. Para ello se hará uso de modelos de redes ya implementados (ResNet18, ResNet50, DenseNet121, EfficientNetB0 y UpaNet), y se llevará a cabo el entrenamiento de los mismos sobre distintos conjuntos de datos para calcular ciertas medidas que se estudiarán, tras lo cual se aplicará alguna técnica de optimización y se comprobará su efecto.

Lo primero que se realizará es un profundo estudio del estado del arte, con el objetivo de poner en contexto la situación planteada en el proyecto. Se detallarán una serie de conceptos acerca de redes neuronales profundas, y en concreto de redes neuronales convolucionales, que son necesarios conocer para poder comprender los distintos experimentos que se llevarán a cabo. Una vez definidos estos conceptos y explicado el funcionamiento de este tipo de redes, se introducirá el objetivo principal del trabajo: la calibración de redes neuronales.

Tras presentar el principal objetivo se introducirá el entorno software en que se ha llevado a cabo el proyecto: el lenguaje de programación *Python* y más concretamente la librería *Pytorch*. Una vez explicada esta librería, se detallarán los aspectos más relevantes de cada modelo de red neuronal. En el siguiente capítulo se explicarán los experimentos, los conjunto de datos empleados en estos y las métricas con las que se evaluarán estos, así como las optimizaciones empleadas para mejorar el rendimiento de las redes.

Finalmente se expondrán los resultados, explicando el porqué de algunos de ellos y dejando la puerta abierta a un futuro trabajo para entender y mejorar algunos comportamientos detectados.

PALABRAS CLAVE

Redes neuronales profundas, redes neuronales convolucionales, calibración, *Pytorch*

ABSTRACT

The objective of this Final Degree Thesis is purely descriptive. It is focused on the study of the behavior of a particular type of deep neural networks: the convolutional ones. For that propose, models that are already implemented (ResNet18, ResNet50, DenseNet121, EfficientNetB0 and UpaNet) are used and trained. After that, some optimization techniques are applied and checked their results.

Firstly, a deep study of the state of art is presented, being the aim of this part to provides an overview of the outlined situation. In this part, several concepts concerning deep neural networks are highlighted, specifically the ones related with convolutional neural networks, which are necessary to understand the different tests carried through in this document. Once these concepts and the functioning of this type of network are described, the main goal of the Thesis is introduced: the calibration of neural networks.

After presenting the key points of this goal, the software environment used in this Thesis is introduced: the programming language Python and specifically the library Pytorch. After the explanation of this library, the most relevant features of each model of neural networks are detailed.

In the next chapter are explained he experiments, datasets that have been used and the metrics with which they will be evaluated, as well as the optimizations used to improve the network performance.

Finally, the obtained results are evaluated and discussed, leaving an open door to future work to understand and improve some behaviors detected.

KEYWORDS

Deep Neural Networks, Convolutional Neural Networks, Calibration, Pytorch,

ÍNDICE

1 Introducción	1
1.1 Objetivos	1
1.2 Motivaciones	2
1.3 Introducción del tema	2
2 Estado del arte	5
2.1 Deep Learning	5
2.1.1 Conceptos básicos	6
2.2 Redes Neuronales Convolucionales	10
2.2.1 Funcionamiento	10
2.3 Calibración de redes neuronales	12
3 Desarrollo	15
3.1 Entorno de trabajo	15
3.1.1 Entorno software: Python y Pytorch	15
3.2 Modelos de redes Neuronales	17
3.2.1 ResNet	17
3.2.2 DenseNet121	19
3.2.3 EfficientNetB0	20
3.2.4 UpaNet	22
4 Experimentos	23
4.1 Introducción	23
4.2 Datasets	23
4.2.1 CIFAR10	24
4.2.2 CIFAR100	24
4.3 Transformaciones aplicadas	24
4.4 Entrenamiento de los modelos	25
4.5 Métricas de evaluación	25
4.6 Técnicas de optimización	26
5 Resultados	29
5.1 Resultados obtenidos	29
5.1.1 ResNet18	29
5.1.2 ResNet50	30

5.1.3 DenseNet121	32
5.1.4 EfficientNetB0	34
5.1.5 UpaNet	35
5.2 Análisis de los resultados obtenidos	37
6 Conclusiones y trabajo futuro	39
6.1 Conclusiones	39
6.2 Trabajo futuro	40
Bibliografía	42
Apéndices	43
A Apendice A: Gráficos de confianza de los modelos	45
A.1 ResNet18	45
A.2 ResNet50	46
A.3 DenseNet121	47
A.4 EfficientNetB0	48
A.5 UpaNet	49
B Apendice B: Otras métricas de calibración en redes neuronales convolucionales	51
B.1 NLL: Negative Log Likelihood	51
B.2 BRIER	52
B.3 NLL y BRIER obtenidos en los experimentos	52
B.3.1 ResNet18	52
B.3.2 ResNet50	53
B.3.3 DenseNet121	54
B.3.4 EfficientNetB0	55
B.3.5 UpaNet	56

LISTAS

Listado de ecuaciones

2.1	Entrada neta de una neurona	7
2.2	Regularización Lasso o L1	8
2.3	Regularización Ridge o L2	9
4.1	Tasa de aciertos de una red neuronal	26
4.2	Error de calibración esperado	26
4.3	Error de calibración máximo	26
4.4	Temperature Scaling	27
B.1	NLL	51
B.2	BRIER	52

Listado de figuras

2.1	Arquitectura de redes neuronales profundas	6
2.2	Convolución	10
2.3	Arquitectura de una red neuronal convolucional	10
2.4	Aplicación de la convolución con una imagen de entrada y un kernel	11
3.1	Comparativa del uso de Python para <i>Data Science</i> frente a otros lenguajes de alto nivel	16
3.2	Bloques residuales en ResNet	17
3.3	Arquitectura de la ResNet18	18
3.4	Arquitectura DenseNet	19
3.5	Distintos modelos DenseNet	20
3.6	Escalado EfficientNet	21
3.7	Comparativa de EfficientNet con otros modelos	21
3.8	Arquitectura UpaNet	22
5.1	sobre entrenamiento en UpaNet	37
A.1	Resultados calibración Resnet18	45
A.2	Resultados calibración Resnet50	46
A.3	Resultados calibración DenseNet121	47

A.4	Resultados calibración EfficientNetB0	48
A.5	Resultados calibración upanets	49

Lista de tablas

5.1	Resultados para la Resnet18 sin Calibración	30
5.2	Resultados para la Resnet18 tras aplicar calibración	30
5.3	Resultados para la Resnet50 sin Calibración	31
5.4	Resultados para la Resnet50 tras aplicar calibración	32
5.5	Resultados para la DenseNets sin Calibración	33
5.6	Resultados para la DenseNet121 tras aplicar calibración	33
5.7	Resultados para la EfficientNetB0 sin Calibración	34
5.8	Resultados para la EfficientNetB0 tras aplicar calibración	35
5.9	Resultados para la UpaNet sin Calibración	36
5.10	Resultados para la UpaNet tras aplicar calibración	36
B.1	BRIER y NLL para Resnet18 sin Calibración	52
B.2	BRIER y NLL para la Resnet18 tras aplicar calibración	53
B.3	BRIER y NLL para Resnet50 sin Calibración	53
B.4	BRIER y NLL para la Resnet50 tras aplicar calibración	54
B.5	BRIER y NLL para Densenet121 sin Calibración	54
B.6	BRIER y NLL para la Densenet121 tras aplicar calibración	55
B.7	BRIER y NLL para EfficientNetB0 sin Calibración	55
B.8	BRIER y NLL para la EfficientNetB0 tras aplicar calibración	56
B.9	BRIER y NLL para UpaNet sin Calibración	56
B.10	BRIER y NLL para la UpaNet tras aplicar calibración	57

INTRODUCCIÓN

1.1. Objetivos

El objetivo de este proyecto es estudiar un aspecto fundamental a la hora de realizar predicciones confiables con una red neuronal: su calibración. Normalmente, a la hora de evaluar si un modelo es bueno o no, nos fijamos en medidas como la función de coste o la tasa de acierto, lo cual no es malo, pero si insuficiente a la hora de cumplir el objetivo. La tasa de acierto, por si sola, no es suficiente a la hora de implementar un modelo de red neuronal fiable. Hay otro concepto, la **confianza** del modelo. Como su propio nombre indica, va a indicar cuánto de 'confiable' es el modelo.

El proyecto estudiará un tipo particular de redes neuronales: **las redes neuronales convolucionales**. Se revisarán la bibliografía más reciente con el objetivo de situarse en el punto actual de las redes neuronales. Se trabajará con distintos modelos de redes neuronales convolucionales en el estado del arte, se analizarán sus arquitecturas, funcionamiento, sus resultados y se analizarán estos, con el objetivo de concluir que problemas existen en las redes actuales.

Se propondrán una serie de técnicas para mejorar el entrenamiento, tales como la regularización, el aumento de datos, así como técnicas para mejorar la calibración de dichos modelos.

Se trabajará con modelos individuales y posteriormente con combinaciones de varios de estos (**ensembles**), con el objetivo de analizar las diferencias y el rendimiento de las redes por separado y en conjunto.

1.2. Motivaciones

La inteligencia artificial se ha convertido en una disciplina fundamental, no solo en el campo de la informática sino en multitud de ellos, como la medicina, el trabajo, la economía o incluso la política. Sin ir más lejos, los sondeos electorales hoy en día son realizados con mecanismos basados en inteligencia artificial. Es inimaginable la capacidad que está adquiriendo la inteligencia artificial. Hoy en día hasta en el mundo del fútbol se realizan importantes inversiones en I+D+I (Investigación + Desarrollo + Innovación). Sin ir más lejos, el Manchester City inglés contrató a varios científicos especializados en *Big Data*, tal como informó el Diario Marca en el mes de marzo [1]. En dicho artículo se hace referencia a la importancia que está adquiriendo el campo de la inteligencia artificial, tanto que ha llegado hasta el mundo del deporte, en concreto al fútbol. Es un caso que puede relacionarse estrechamente con este TFG. Solo hay que imaginarse la de imágenes que puede ser captadas a lo largo de un partido de fútbol. Estudiar imágenes de la anchura media de un equipo a lo largo de un partido, imágenes que estimen la distancia media entre la línea de defensa y la de medio campo... Infinidad de cosas. Otro ejemplo, en este caso centrado en medicina, es que mediante la inteligencia artificial (y en especial el *deep learning*, en el que está enfocado este proyecto) se pueden diagnosticar enfermedades con imágenes (radiografías por ejemplo). Se puede diagnosticar si un paciente sufre COVID-19 mediante una radiografía de su tórax, procesándola con redes neuronales convolucionales, por ejemplo.

Todo esto me llevó a querer elegir un TFG relacionado con este campo. Me considero un apasionado del dato, de todos los procesos relacionados con él. Casualmente, encontré a mi tutor Juan, que proponía un tema bastante interesante. Me llamaba la atención estudiar un aspecto de las redes neuronales que yo mismo desconocía, con el objetivo de ampliar mis conocimientos y disfrutar de una investigación muy novedosa para mí.

1.3. Introducción del tema

Las redes neuronales convolucionales son un tipo de red multicapa, cuya principal peculiaridad es el uso de la convolución en sus distintas capas. Están especialmente diseñadas para el procesamiento de datos de dos dimensiones, como reconocimiento de imágenes y señales de voz.

Gracias a los avances en materias como el aprendizaje profundo (*Deep Learning*) las redes neuronales modernas se han visto considerablemente mejoradas (sus tasas de acierto son considerablemente buenas en relación con la complejidad del problema que resuelven). Esto ha dado pie a que sean utilizadas para multitud de tareas dados los buenos resultados que ofrecen. Sin embargo, algunas de estas mejoras (como es por ejemplo el aumento de profundidad de la red), han hecho que las redes neuronales actuales estén mal calibradas.

La calibración de una red neuronal consiste tratar de igualar la confianza del modelo y su tasa de acierto. Es por ello que se trata de algo muy importante a la hora de cerciorarse de que una red neuronal tiene un buen rendimiento o no. Poniendo esto en contexto: si las redes neuronales tienen muy buenos resultados, ya que como se ha dicho antes los avances han permitido grandes mejoras en la tasa de acierto, pero su calibración no es del todo buena, quiere decir que la confianza es mala. Y si la confianza es mala, el objetivo de generar predicciones fiables no va a cumplirse. Por tanto, será necesario calibrar el modelo.

El TFG seguirá esta línea de investigación. Se analizarán los distintos modelos de redes neuronales y para cada uno de ellos se estudiarán las medidas que determinan la calibración. A su vez, se aplicará alguna técnica de calibración y se hará una comparativa de los resultados, analizando el sentido de estos.

ESTADO DEL ARTE

En este capítulo se va a realizar una profunda revisión del tema a tratar: las redes neuronales convolucionales, su entrenamiento y calibración. Para ello se abordará el tema del *Deep Learning*, así como una serie de conceptos fundamentales que son necesarios conocer para realizar las distintas pruebas que se plantearán en el capítulo 4. Una vez vistos estos conceptos se presentarán las redes neuronales convolucionales: qué son y cómo funcionan. Finalmente se presentará el contexto de la calibración de redes neuronales convolucionales.

2.1. Deep Learning

Se conoce como *Deep Learning* o aprendizaje profundo al conjunto de técnicas o algoritmos cuyo objetivo es imitar al cerebro humano y su capacidad de reconocimiento y aprendizaje de determinados patrones. Esto se lleva a cabo mediante capas, formadas por unidades individuales de procesamiento (*neuronas*) que en su conjunto forman arquitecturas que admiten transformaciones no lineales (redes neuronales). El objetivo principal de las técnicas de *Deep Learning* es el de entrenar dichas redes neuronales, con una serie de conjuntos de entrada etiquetados. Una vez entrenadas las computadoras, estas tienen que tener la capacidad de resolver prácticamente cualquier entrada que se les proporcione, dando lugar a salidas adecuadas. Para ello, el *Deep Learning* emplea modelos estadísticos para localizar patrones en entornos caracterizados por cierta aleatoriedad y, de esta manera, poder diferenciarlos de manera automática.

Las técnicas de *Deep Learning* tienden a proveer mecanismos para aportar capacidad de generalización a las máquinas, y que estas no aprendan específicamente el conjunto de pruebas que les pasa en su entrenamiento (conocido como sobreentrenamiento).

En la figura 2.1 se puede ver una representación del funcionamiento de las redes neuronales profundas. En este tipo de redes la información se propaga únicamente hacia delante (redes *feedforward*). En ellas, las primeras capas se encargan de la percepción de patrones (generalmente más concretos) que van propagándose por la red, dando lugar a patrones más abstractos a medida que van avanzando a lo largo de la red neuronal.

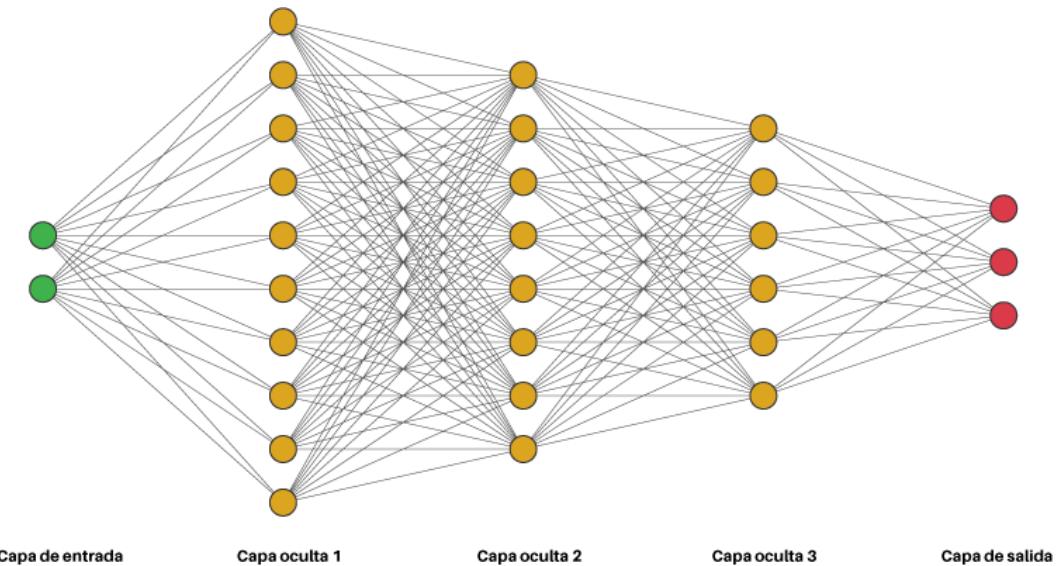


Figura 2.1: Ejemplo de una arquitectura simple de red neuronal profunda. Extraída de [2]

2.1.1. Conceptos básicos

Conjuntos de datos

El conjunto de datos hace referencia a los datos que van a procesar la red neuronal. Suelen ser conjuntos los suficientemente grandes como para que la red adquiera un alto nivel de aprendizaje de los patrones. Estos patrones son valores etiquetados que siguen una estructura $(s:t)$, donde s hace referencia al vector que recibe la capa de entrada y t al vector de salidas objetivo que debería alcanzarse con dicha entrada. Según el uso que reciba el conjunto de datos, se puede distinguir principalmente 3 tipos: **conjunto de entrenamiento**, usado como su propio nombre indica para entrenar el modelo, **conjunto de clasificación**, usado tras el entrenamiento de la red para comprobar el grado de aprendizaje de la misma y **conjunto de validación**, que suele ser un sub-conjunto de alguno de los anteriores (generalmente de entrenamiento) para comprobar el correcto funcionamiento del modelo. Sirve por ejemplo para detectar problemas de sobre-entrenamiento. En este trabajo (sección 4.6) el conjunto de validación será empleado en la técnica elegida para la calibración de modelos.

Tasa de aprendizaje

La tasa de aprendizaje es uno de los hiperparámetros a ajustar para el entrenamiento de la red neuronal. Determina la rapidez con que se alcanza la convergencia a unos valores óptimos. Una tasa de aprendizaje demasiado elevada podría impedir que se alcance la solución óptima. Un valor demasiado pequeño ralentiza mucho la ejecución del algoritmo. La tasa de aprendizaje suele tener valores pequeños, como 0.01 o 0.001. La tasa puede ser fija o adaptativa. Esta última es ideal cuando se disponen de pocos patrones de entrenamiento para una clase.

En este caso la tasa se suele aumentar cuando se procesan patrones de dicha clase. También puede darse el caso de que, para cada conexión, exista una tasa de aprendizaje, que varía más o menos en función de la actualización de pesos previa.

Función de activación

El concepto de función de activación hace referencia a la transformación que se le aplica a una entrada de una red neuronal cuando se activa, antes de propagar el valor resultante de dicha transformación. La función de activación o función de transferencia recibe como valor la entrada neta de la neurona (eq. 2.1) y devuelve el valor correspondiente a dicha función.

$$Z_{in} = \sum_{i,j}^n x_i * w_{ij} + b \quad (2.1)$$

Este valor se propagará el valor a través de las conexiones a las neuronas de otras capas. Las funciones de activación más usadas son **sigmoide**, la **ReLU**, y la **softmax** para problemas de clasificación multiclas.

La **sigmoide** es una función de activación que recibe un valor de entrada y devuelve una salida de dicho valor, mapeada en el intervalo [0,1]. Presenta como principal desventaja que, al estar acotada en un intervalo muy pequeño, si se emplea en una arquitectura muy profunda los gradientes sucesivos se hacen muy pequeños, llegando a las primeras capas de la red un valor de corrección de pesos muy pequeño, lo que desemboca en un algoritmo de entrenamiento lento.

La **ReLU** es una función de activación que dado un valor de entrada, devuelve ese mismo valor en caso de ser mayor o igual que cero. Si la entrada es negativa, devuelve cero. Es la más usada en aprendizaje profundo, ya que su gradiente no satura como en el caso de las sigmoides, al tener un intervalo más grande. Como principal desventaja presenta que no es derivable en 0, pero este problema se resuelve tomando la derivada por la izquierda.

La **softmax** es una función de activación empleada cuando se dispone de datos clasificados en más de 2 clases. Se trata de una generalización de la función logística que recibe un vector de tamaño T y devuelve otro vector con sus componentes en el intervalo [0,1]. La clase predicha será la que mayor valor tenga.

Función de coste asociada

"La función de perdida, también conocida como función de coste asociada, es la función que nos dice que tan buena es la red neuronal. Esta es la función que se optimiza en el algoritmo de retropropagación" [3]. Esta función determina lo buena que es la neurona midiendo la diferencia entre salida obtenida y la salida esperada. En función de la distribución que sigan nuestros datos, la función de coste asociada puede variar. Hay varias funciones de coste asociada, siendo las más frecuentes de uso la función del **error cuadrático medio** y la **cross entropy**.

Sobreajuste de parámetros

Se conoce como sobre ajuste de parámetros o sobre entrenamiento (en inglés *overfitting*) al fenómeno dado cuando una red aprende demasiado bien los patrones de entrenamiento, perdiendo la capacidad de generalización. Esto da como resultado un excelente rendimiento con los patrones de entrenamiento, suceso que no se repite con los patrones de clasificación que la red no ha visto. Suele detectarse al graficar el progreso de la función de coste. Mientras que la función de coste para los patrones de entrenamiento disminuye progresivamente, la función de coste para los patrones de clasificación se estanca en un determinado punto y comienza a ascender. Una de las principales causas del sobre-entrenamiento es el tamaño de las redes (que casualmente también es causa de una mala calibración). Cuanto más profunda o más ancha sea la red (más conexiones tenga), más parámetros va a tener que ajustar y por tanto más va a memorizar el conjunto de entrenamiento, perdiendo la capacidad de generalizar. Existen muchas técnicas para evitar el sobre-entrenamiento. Una de las principales es la regularización.

Regularización

La regularización es una técnica empleada para evitar el sobre entrenamiento. Consiste en modificar el algoritmo de entrenamiento, añadiendo algún valor constante, de tal forma que la función de coste para el conjunto de clasificación no se estanke en cierto punto. Para ser exactos, "*la regularización consiste en añadir una penalización a la función de coste. Esta penalización produce modelos más simples que generalizan mejor.*" [4]. Los dos métodos de regularización más conocidos son el L1 (lasso) y L2 (ridge).

La regulación L1 penaliza la suma del valor absoluto de los coeficientes de regresión.

$$J_{L1} = J + \lambda * \sum_w |w| \quad (2.2)$$

Por otro lado, la regularización L2 penaliza la suma de los cuadrados de los coeficientes de regresión.

$$J_{L2} = J + \lambda * \sum_w w^2 \quad (2.3)$$

Aumento de datos

"El término aumento de datos se refiere a métodos para construir algoritmos de optimización o muestreo iterativos mediante la introducción de datos no observados o variables latentes" [5]. En otras palabras, consiste en generar más cantidad de datos a partir de los disponibles inicialmente mediante ligeras modificaciones de los originales. Por ejemplo, para redes convolucionales los giros de imágenes o la aleatorización de estas son técnicas de aumento de datos. En la sección 4.3 se verán algunos métodos de aumento de datos utilizados en los conjuntos de datos.

Descenso por gradiente

El descenso por gradiente es una técnica de optimización que permite converger al valor mínimo de una función de forma iterativa, a través de su gradiente (derivada). Se usará para minimizar la función de coste, en función de los parámetros de la red (los pesos). De esta forma, irá buscando los pesos que minimizan el valor de dicha función.

Entrenamiento de las redes neuronales: Algoritmo backpropagation

El entrenamiento de una red neuronal es el proceso mediante el cual se ajustan los pesos de todas sus conexiones de tal forma que las salidas que la red proporcione se ajusten lo máximo posible a las salidas objetivo para cada entrada. Es la fase más costosa, pues encontrar todos los pesos óptimos de manera conjunta suele requerir mucho tiempo. Hay entrenamientos que pueden durar hasta meses. Por ello, a la hora de programar un algoritmo de entrenamiento y aplicarlo hay que tener en cuenta la red que se va a entrenar (cuanto más profunda sea, más parámetros (conexiones) tenga más largo será el entrenamiento), así como los recursos hardware de los que se dispone. Por el contrario, la fase de explotación de datos luego es muy rápida, llevando apenas unos pocos segundos. Para el entrenamiento de redes neuronales hay múltiples algoritmos. El más reconocido y que será empleado en los experimentos es el *Backpropagation*.

Se conoce como backpropagation al método de cálculo del gradiente empleado en algoritmos de aprendizaje supervisado para el entrenamiento de redes neuronales profundas [6]. El proceso ajusta repetidamente los pesos de las conexiones en la red, tratando de minimizar la medida de error entre la salida obtenida y la esperada, empleando para ello el método del descenso por gradiente.

2.2. Redes Neuronales Convolucionales

Las redes neuronales convolucionales son un tipo de redes neuronales profundas dedicadas principalmente al tratamiento de imágenes. Están inspiradas en la organización de la corteza del cerebro, con un funcionamiento muy parecido al de este.

Se caracterizan principalmente por emplear la operación de **convolución**. La convolución es una operación matemática que, dadas dos funciones de entrada f y g devuelve una tercera que representa la superposición de una con otra. En el contexto de las redes neuronales convolucionales, las funciones de entrada serán unos píxeles de una imagen y una máscara (o kernel), dando como salida otra serie de píxeles que resultan de la combinación lineal de los dos componentes anteriores (ver figura 2.2)

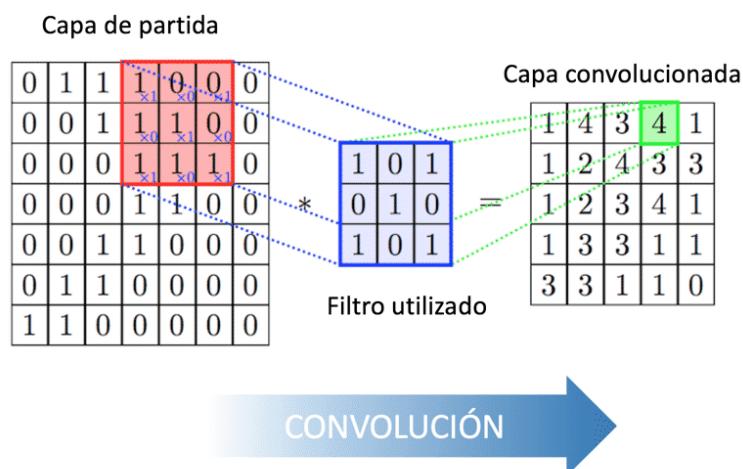


Figura 2.2: Operación de convolución en redes neuronales convolucionales. Extraída de [7]

2.2.1. Funcionamiento de las redes neuronales convolucionales

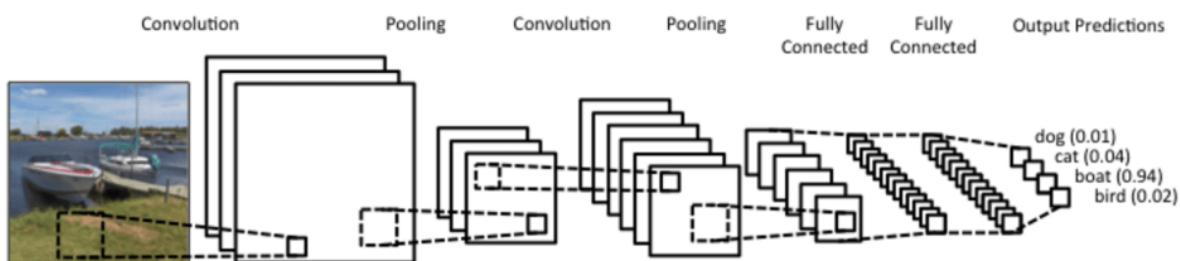


Figura 2.3: Ejemplo de arquitectura de una red neuronal convolucional. Extraída de [8]

Las imágenes que procesan las redes neuronales convolucionales son simplemente matrices de píxeles. Cada píxel tiene un valor entre 0 y 255 (escala de colores), que antes de la entrada a la red se normaliza, siendo mapeado el valor dentro del intervalo [0,1].

El tamaño y escala de colores de la imagen determina el n.^o de neuronas. Por ejemplo para imágenes de 28x28, se usarían 784 neuronas (en escala de grises). Si fuese en color (RGB), serían $784 \times 3 = 2352$ neuronas.

Capa de convolución

La capa de convolución realiza, como su propio nombre indica, una operación de convolución. Se trata de la capa que mayor coste computacional desempeña, pues la operación de convolución cuando se reciben matrices (imágenes) muy grandes es bastante costosa. Se van tomando píxeles de la imagen y se van procesando junto con el conjunto de *kernels*. En estas capas de convolución hay N *kernels*, cada uno del tamaño (W,H,D) donde W será la anchura (*Width*), H la altura (*Height*) y D el número de dimensiones (*Dimensions*). Se realiza la operación de la convolución con cada kernel, por lo que a la salida de esta capa convolucional habrá N matrices de salida. Cada filtro almacena en su matriz de salida correspondiente es una serie de características de cada imagen.

Después de que cada kernel haya terminado de barrer la matriz de entrada, se aplica la función de activación.

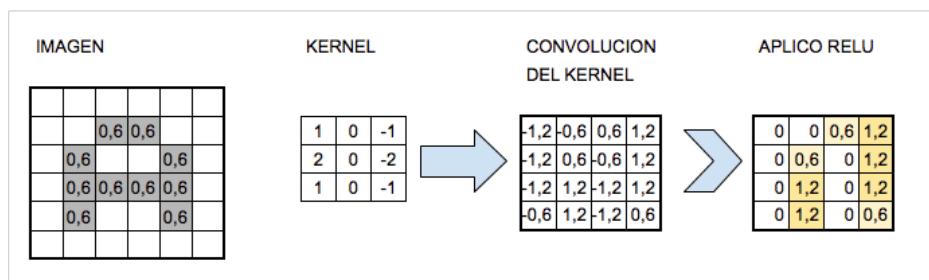


Figura 2.4: Ejemplo del proceso que es llevado a cabo por cada kernel de la capa de convolución. Este proceso es repetido por los N kernels que forman parte de la capa de convolución, dando como resultado N matrices como la última. Extraída de [9]

Capa de Pooling

A la salida de la capa convolucional se generan N matrices de dimensión (W, H, D) donde N es el número de *kernels* que tiene dicha capa. Esto termina dando lugar a un número de neuronas muy elevado, algo computacionalmente inviable. Por ello, a la salida de algunas capas de convolución, como se puede ver en la figura 2.3, se encuentra una capa de *pooling*, cuyo objetivo es generar unas sub-muestras de tamaño (PxP) de las matrices salientes de la capa convolucional previa.

A la salida de la capa de pooling, por tanto, habrá N matrices, pero el tamaño de las mismas será reducido considerablemente. Ahora serán matrices (W/P , H/P , D). El pooling puede ser llevado a cabo de dos formas: *MaxPooling*, que escoge como salida el valor más alto de los seleccionados y *AvgPooling*, que devuelve la media de esos valores.

Capa feedforward

Siguiendo la figura 2.3, al final de la red neuronal convolucional hay una capa *feedforward*. Esta no es más que un perceptrón multicapa. También se la puede llamar capa *fully-connected*. El objetivo que tiene esta capa es transformar la información de la red convolucional, de tal forma que se reduce la complejidad de la información y se adapta para ser procesada por la *softmax*. Esta red genera los valores que recibirán las neuronas de la capa de salida tras haber sido pasados por dicha función.

2.3. Calibración de redes neuronales

El concepto de calibración hace referencia a la similitud entre la precisión del modelo y su confianza. Es un error común pensar que con obtener una precisión muy buena, el modelo ya es fiable. Generalmente, las predicciones realizadas por la *softmax* suelen estar sobrevaloradas. Para poner en contexto, si se dan 100 predicciones, con una confianza de 0.9 cada una de ellas, el objetivo será que la precisión de la red sea del 90 %. El objetivo de la calibración no es otro que el de dotar a la red neuronal de confianza, permitiendo de esta forma que la red pueda llevar a cabo tareas de alto riesgo con la seguridad de que va a actuar correctamente. Es decir, el objetivo es permitir que la red realice predicciones *reliables* o confiables.

Hoy en día, los avances en *Deep Learning* han permitido que las redes neuronales alcancen tasas de aciertos más elevadas. Sin embargo, las redes neuronales actuales están peor calibradas que las de hace una década. La profundidad y anchura de la red, así como el uso de la normalización por lotes son factores que descalibran las redes neuronales [10].

Las redes neuronales modernas están formadas por muchas capas, las cuales a su vez están compuestas por multitud de neuronas. Es sabido también que las redes con muchos parámetros (muchas conexiones) pierden la capacidad de generalización. Este es el principal síntoma del sobreentrenamiento. Por lo que se puede deducir que la mala calibración y el sobreentrenamiento están estrechamente relacionados.

Las redes neuronales están en un principio mal calibradas. Se las aplicará un método de calibración que será explicado en la sección 4.6. Este método a priori debe regular las predicciones del modelo y proporcionar una mayor calibración. A su vez, para poder comparar con mayor facilidad la efectividad de dicho método, se presentarán los llamados diagramas de fiabilidad: histogramas que muestran la confianza del modelo frente a su precisión. Mediante estos histogramas se pretende facilitar la comprensión de las medidas mostradas en el capítulo 5, ya que visualmente se observará mejor la diferencia entre una red bien calibrada y una red mal calibrada.

DESARROLLO

Una vez realizado un profundo estudio del estado del arte del tema que se va a tratar, ya se está en disposición de entender el trabajo realizado a lo largo del proyecto. En este capítulo se va a explicar el proceso llevado a cabo para el desarrollo de este proyecto. En primer lugar se hará un análisis del entorno en el que se ha trabajado, explicando la librería elegida para llevar a cabo las distintas pruebas. Se hablará también de los modelos de redes elegidos para llevar a cabo las distintas pruebas.

3.1. Entorno de trabajo

En esta sección se va a analizar el entorno sobre el que se han llevado a cabo los experimentos tratados en el proyecto. Se hablará de Python como lenguaje de programación, analizando por qué es el lenguaje más utilizado en el entorno de *Data Sciences*. A su vez, se hablará de la librería proporcionada por Python con la que se ha trabajado: **Pytorch**.

3.1.1. Entorno software: Python y Pytorch

Para el desarrollo de este proyecto se ha empleado el lenguaje de programación Python. Python es sin duda (ver figura 3.1) el lenguaje de programación más popular en el ámbito del aprendizaje automático. En los últimos años su uso se ha extendido a multitud de campos. Si bien para lógica de negocio todavía está por detrás de otros como Java, en el campo de la ciencia de datos es sin lugar a duda un referente a día de hoy. La diferencia para ser el favorito en el campo del *Machine Learning* es sin duda la cantidad de librerías externas que posee y la utilidad que estas dan a la hora de transformar datos y procesarlos. En este aspecto, se pueden encontrar multitud de librerías para el trabajo de datos, como numPy o sciPy, librerías que son altamente compatibles con las específicas para el trabajo con redes neuronales. Aquí caben destacar tres librerías: TensorFlow, Keras y Pytorch. Dado que el proyecto está realizado con Pytorch, se profundizará sobre esta librería.

Pytorch es una biblioteca de Python que emplea el paquete Torch, destinada para el aprendizaje profundo sobre datos de entrada irregulares, como gráficos (imágenes, videos...) [12].

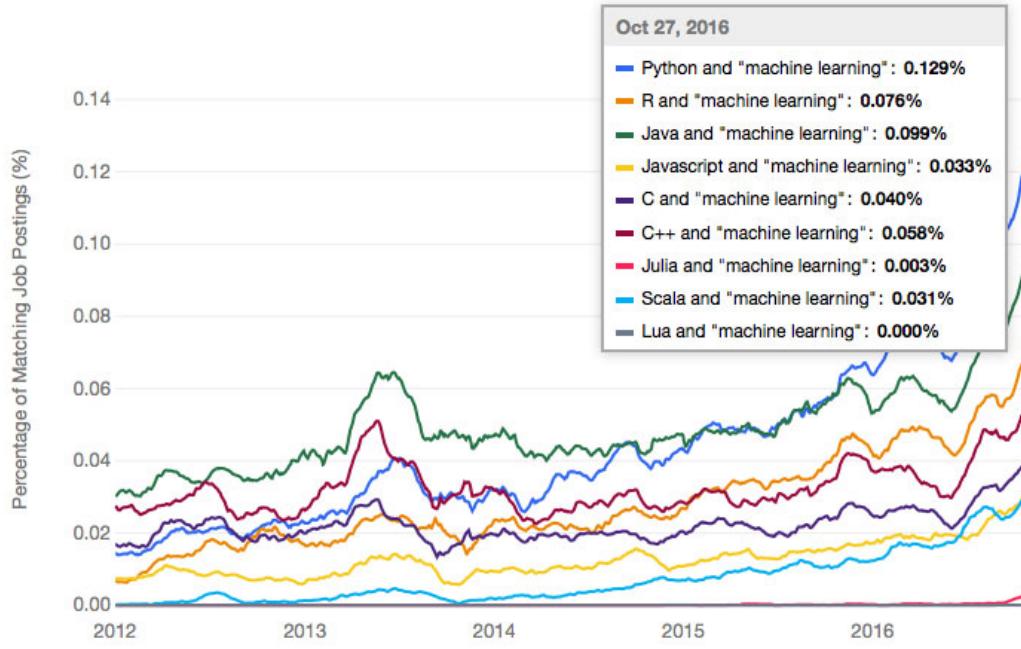


Figura 3.1: Comparativa del uso de distintos lenguajes de programación en el campo del Machine Learning. Extraída de [11]

Trabaja, al igual que Keras y TensorFlow con estructuras vectoriales y de matrices (tensores), muy similares a los arrays de numpy por ejemplo, lo que hace que combinar funcionalidad de dichas librerías sea bastante fácil y útil. Pytorch proporciona varias ventajas respecto a sus principales competidoras, como la generación dinámica de grafos en vez de estáticos. Esto permite que las operaciones se realicen según se va ejecutando el programa. Otra ventaja muy importante es que Pytorch proporciona clases para crear tus redes a partir de las creadas en la propia librería (por ejemplo *Nn.Module*). También proporciona procesamiento paralelo y uso de GPUs a través de la interfaz CUDA, que permite conectar la CPU con la GPU, lo cual supone una gran ventaja al disponer de varias unidades de procesamiento gráfico, que permitirán acelerar el entrenamiento de las redes neuronales.

Procedimiento: Entrenamiento de las redes con Pytorch

Con Pytorch se ha implementado el entrenamiento de las redes empleando *backpropagation*. Lo primero, tras importar las distintas librerías, será definir las transformaciones que serán aplicadas a los datos con los que se va a trabajar. Estas transformaciones se verán más específicamente en la sección 4.3. Una vez definidas las transformaciones, el siguiente paso será cargar los conjuntos de datos con los que se va a trabajar. Estos conjuntos de datos (sección 4.2) se cargan de mediante el módulo *torchvision.datasets*. Una vez cargados y guardados, se crea la red neuronal que se va a entrenar, y se aloja en el dispositivo con el que se vaya a trabajar (en este caso, se aloja en la GPU y se paralleliza con *DataParallel*). Una vez definido el modelo, hay que definir la función de coste (**CrossEntropyLoss**) y el optimizador (**SGD**). Con esto, ya puede empezar el entrenamiento de la red neuronal.

Se procesará en cada época el conjunto de entrenamiento y el de test, con el objetivo de identificar síntomas de sobre entrenamiento. Al finalizar el entrenamiento, se guardará el modelo en un fichero de extensión `.pt`.

3.2. Modelos de redes Neuronales

A continuación se hablará de cada modelo de red neuronal empleado. Se pueden diferenciar cuatro tipos de redes neuronales: ResNet (ResNet18 y ResNet50), DenseNet, EfficientNet y UpaNet.

3.2.1. ResNet

Las ResNet (*Residual Neural Network*) son una topología de redes neuronales convolucionales. Deben su nombre al concepto de residuo. El término viene dado por los llamados 'bloques residuales' de la red. Hace referencia a aquellas conexiones que se saltan ciertos bloques (tratados como residuos). Esto se suele hacer típicamente para evitar desvanecimientos de los gradientes. Se lleva información importante saltando la capa (ver figura 3.2) hasta que esta aprende sus pesos. Emplear esta técnica hace que sea más sencillo optimizar estas redes y además se obtiene una mayor tasa de aciertos aumentando la profundidad. [13]. Aumentar la profundidad de la red en principio debería mejorar la precisión de esta. El problema es que una red muy profunda va a encontrar problemas a la hora de converger en un ajuste de parámetros que la haga óptima. Como se ha visto en la sección 2.1.1, en los apartados de funciones de coste y descenso por gradiente, cuando la red es muy profunda los gradientes se saturan (se hacen muy pequeños) y los pesos apenas varían de una época a otra.

El funcionamiento de las ResNet que van a utilizarse en este proyecto son idénticos. Únicamente va a cambiar el n.^º de capas que tiene cada red (al margen de rendimiento). Concretamente se trabajará con Resnet18 y Resnet50

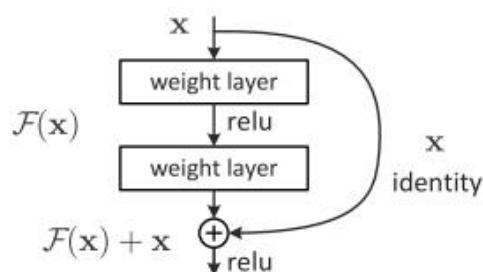


Figura 3.2: Imagen que muestra el funcionamiento de la técnica de bloques residuales empleada en las arquitecturas ResNet. Extraída de [13]

ResNet18

La Resnet18 es una red neuronal convolucional perteneciente a la familia de las ResNet. Consta de 18 capas de profundidad. Como se puede observar en la figura 3.3 realmente la red está organizada en cuatro grandes capas. Dentro de cada una de esas cuatro capas hay otras cuatro capas de convolución (lo que hace un total de 16 capas). A estas hay que sumarles una capa de convolución inicial y la capa *feedforward*, que dan el 18 esperado. Cada una de las 4 capas principales tiene un funcionamiento similar. Recibe una entrada, la procesa en una capa convolucional y luego aplica *batch normalization*. Esta salida pasa a la segunda capa convolucional, se procesa y se aplica otra vez *batch normalization*. Tras ello, se realiza la suma con la entrada de la capa y se propaga con la función de activación (ReLU). Puede verse en la figura 3.3 que a la salida de la primera capa convolucional, antes de la entrada al primero de los 4 bloques se realiza un *max pooling*. A la salida del último de estos 4 bloques se realiza otro pooling, en este caso un *average pooling*, con el objetivo de reducir el tamaño de los datos que va a recibir la capa *feedforward*.

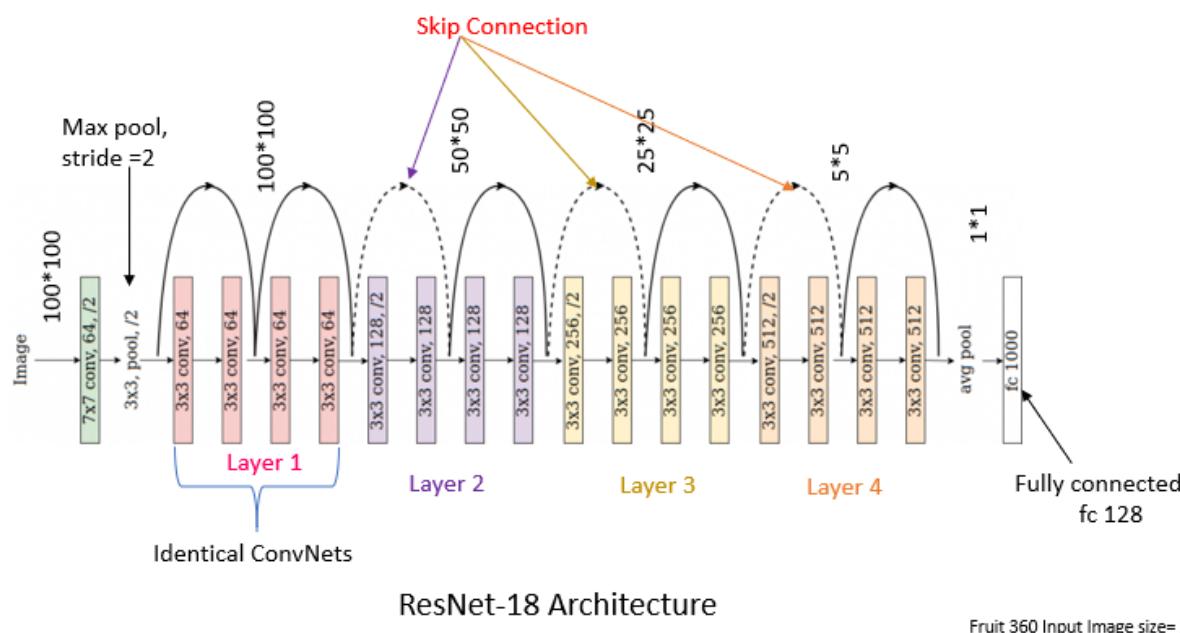


Figura 3.3: Imagen que muestra la organización de una ResNet18, su n.^o de bloques y las distintas operaciones por las que la información pasa. Extraída de [14]

ResNet50

La Resnet50, al igual que la ResNet18, pertenece a la familia de las ResNet. Su funcionamiento es idéntico al de la ResNet18, simplemente pasa de tener 18 capas de profundidad a tener 50.

3.2.2. DenseNet121

La DenseNet121 es una red neuronal perteneciente a la familia de las DenseNet. A diferencia de las ResNet, que tienen N conexiones conectando las N capas, estando cada capa conectada únicamente con la capa predecesora y con la antecesora, las DenseNet tienen $\frac{N(N+1)}{2}$ conexiones, en forma de retroalimentación, como se puede ver en la figura 3.4. Trabajos recientes han demostrado que las redes convolucionales pueden ser sustancialmente más profundas, más precisas y eficientes de entrenar si contienen conexiones más cortas entre capas cercanas a la entrada y aquellas cercanas a la salida [15].

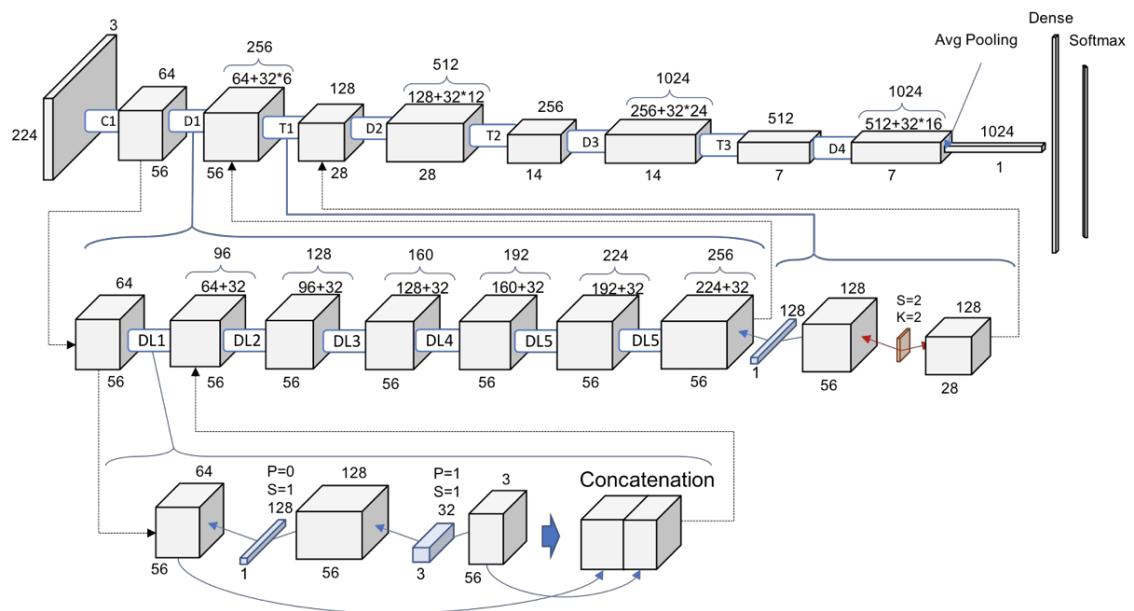


Figura 3.4: Ejemplo de arquitectura de una DenseNet. Extraída de [16]

La DenseNet, en comparación con las ResNet, mejora muchos aspectos. Termina con el problema de desvanecimiento del gradiente sin prescindir de bloques (como hacen las ResNet con los bloques residuales). A su vez, mejora en términos de eficiencia computacional, ya que requiere menos uso de memoria.

En la figura 3.5 se pueden observar los principales modelos de DenseNet. En el proyecto se trabajará únicamente con la DenseNet121. Común a todos los modelos son las dos primeras capas. La primera realiza una convolución. Dada una entrada de 112x112, se realiza una convolución y seguidamente un *maxpooling* que reduce el tamaño a la mitad (56x56). Esta es la entrada al primer bloque Dense.

Los bloques Dense son similares a las 4 capas que se han visto en la Resnet18. En el caso de la DenseNet121 (y del resto de modelos), hay 6 pares, cada uno con 2 capas de convolución, con su respectivo batch normalization posterior a cada una de ellas. Lo siguiente que se puede observar es una *transition layer*. Sirve para reajustar el tamaño de la imagen que viaja a través de las capas.

Como a la salida del primer bloque Dense el tamaño es 56x56 y el siguiente espera 28x28, la red cuenta con dicha capa para reajustar las dimensiones. Esto es similar en el resto del tránsito de la imagen por toda la red. Finalmente se llega al *averagePooling*, que reduce la última salida a una dimensión para ser procesada por la capa feedforward y devolver la salida correspondiente.

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112		7 × 7 conv, stride 2		
Pooling	56 × 56		3 × 3 max pool, stride 2		
Dense Block (1)	56 × 56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56 × 56		1 × 1 conv		
	28 × 28		2 × 2 average pool, stride 2		
Dense Block (2)	28 × 28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28 × 28		1 × 1 conv		
	14 × 14		2 × 2 average pool, stride 2		
Dense Block (3)	14 × 14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	14 × 14		1 × 1 conv		
	7 × 7		2 × 2 average pool, stride 2		
Dense Block (4)	7 × 7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	1 × 1		7 × 7 global average pool		1000D fully-connected, softmax

Figura 3.5: Distintos modelos de DenseNet. Se pueden observar los distintos bloques que componen cada red, sus capas y el tamaño de las entradas y salidas. Extraída de [16]

3.2.3. EfficientNetB0

La EfficientNetB0 es la más simple de las redes pertenecientes a la familia EfficientNet. Las redes EfficientNet son peculiares respecto a las demás. Se han visto hasta ahora tres modelos de redes neuronales (ResNet18, ResNet50 y DenseNet121). En ellos se han visto detalles como el tamaño de sus salidas (resolución) o el número de capas que tienen (profundidad de la red). Ahora se verá otro concepto nuevo: la anchura. La anchura mide el número de neuronas que tiene la red en sus capas. Y este concepto, junto a la profundidad y la resolución, es vital a la hora de hablar de la peculiaridad de las EfficientNet.

EfficientNet es una arquitectura de red neuronal convolucional con un método de escalado que escala uniformemente profundidad, ancho y resolución utilizando un coeficiente compuesto. A diferencia de la práctica convencional que escala arbitrariamente estos factores, EfficientNet escala uniformemente estas medidas a través de un conjunto de coeficientes de escala fijos. [17]

Esto supone una gran ventaja, ya que de esta forma la red se adapta a la entrada que recibe.

Si la imagen es muy grande, su escalado aumentará la profundidad, la anchura y la resolución. Esto mejora notablemente el rendimiento general de la red.

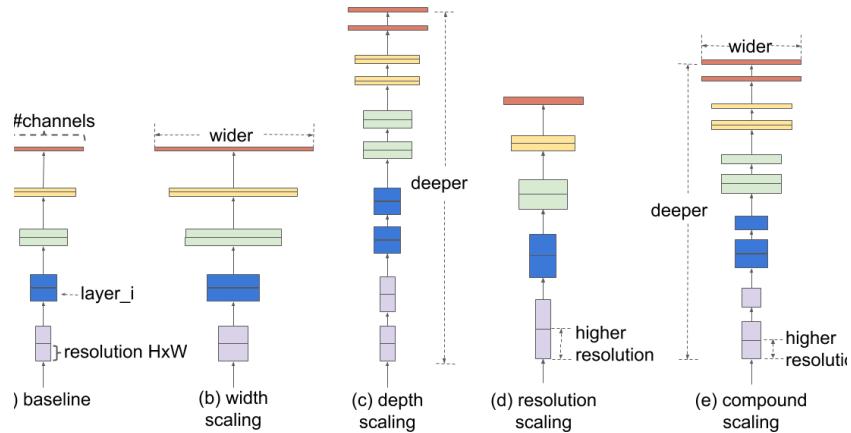


Figura 3.6: Ejemplo de escalado: se pueden observar los distintos escalados de anchura, profundidad y resolución de imagen. Extraída de [17]

En la figura 3.6 se pueden ver los distintos escalados de las redes EfficientNet. Como se ha dicho, la EfficientNetB0 es el modelo más sencillo de esta topología de redes convolucionales. Es un modelo simple, fácil de escalar, pero que garantiza unos resultados bastante buenos y empleando un n.^o de parámetros bastante menor que otras redes neuronales convolucionales, como se puede ver en la figura 3.7. En 3.7 se puede ver una comparativa de esta topología con otras redes, en cuanto a n.^o de parámetros y tasa de acierto. La EfficientNet correspondiente en cuanto a tamaño a la ResNet50 sería la EfficientNetB5, que proporciona una tasa de aciertos un 10 % mejor.

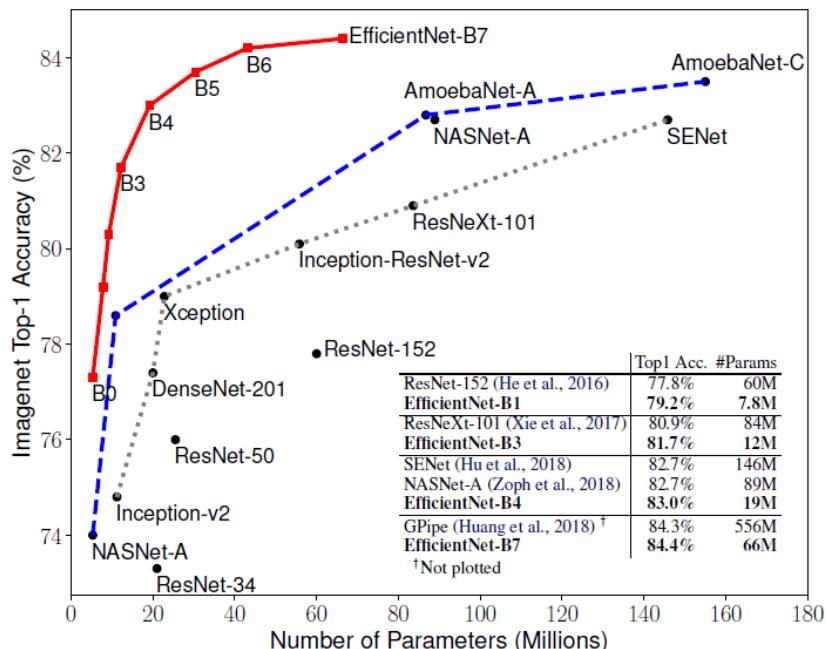


Figura 3.7: Comparativa entre las estructuras efficientnet y otros modelos de redes convolucionales para el conjunto de datos ImageNet. Extraída de [17]

3.2.4. UpaNet

UpaNet es, sin duda, la red neuronal más actual de todas las vistas en este proyecto (data de marzo de 2021). Se trata de una red creada por varios reconocidos científicos de distintos países, quienes tras realizar un estudio profundo acerca de las redes neuronales más exitosas, fueron capaces de diseñar esta red neuronal, combinando las distintas ventajas de los modelos más reconocidos del estado del arte de las redes neuronales convolucionales. De esta forma surge UpaNet. Se propone como una arquitectura que equipa la atención por canal, con una estructura híbrida de redes residuales (ResNet) de redes densamente conectadas (DenseNet), *'con el objetivo de poder procesar información global y local'* [18]. Esto, junto con una estructura de conexión extrema, convierte a UpaNet en un modelo muy efectivo, eficiente en su entrenamiento y muy robusto, habiendo superado el éxito de otras redes neuronales convolucionales.

Se propone el estudio de esta red neuronal para evaluar las ventajas de combinar dos de las técnicas vistas hasta ahora (ResNet y DenseNet), con reconocido éxito en el campo de la visión por computación.

En la figura 3.8 se puede ver la organización de las distintas capas de la UpaNet. El bloque 0 está formado a su vez por otros dos bloques que utilizan la técnica de residuos de las ResNet. [18]. Se utiliza la idea de arquitectura densamente conectada con el objetivo de reutilizar mapas de características y reducir el número de parámetros de la red, al estilo de la DenseNet. Sin embargo, el modelo UpaNet modifica esta idea, con una estructura distinta, como se puede ver en la 3.8.

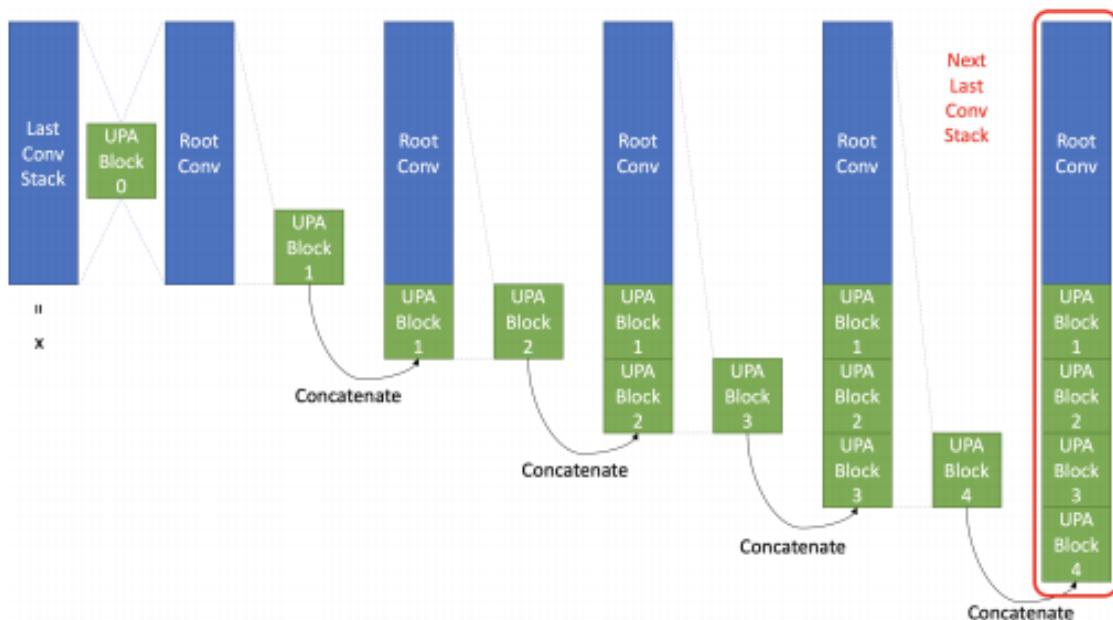


Figura 3.8: Ejemplo de la organización de las capas del modelo UpaNet. Extraída de [18]

EXPERIMENTOS

4.1. Introducción

En esta sección se procederá a explicar como se han llevado a cabo las distintas pruebas para responder al objetivo del proyecto: estudiar el comportamiento de los distintos modelos de redes neuronales y de la combinación de varios de ellos (*ensembles*). Se realizará el entrenamiento de redes neuronales, y la clasificación en distintos *datasets*. Se procederá a evaluar la calibración de los distintos modelos, obteniendo las métricas correspondientes y finalmente se aplicarán técnicas que optimicen dichas medidas

4.2. Datasets

Los *datasets*, como se ha visto en la sección 2.1.1, son los conjunto de datos con los que se entrena el modelo (conjunto de entrenamiento), y posteriormente se evalúa (conjunto de clasificación). Para la realización de los experimentos se ha realizado la siguiente división de conjuntos:

- Conjunto de entrenamiento
 - conjunto de entrenamiento: 100 %
- conjunto de test
 - conjunto de validación: 20 %
 - conjunto de test: 80 %

El conjunto de entrenamiento ha sido utilizado para el propio entrenamiento de las redes neuronales. Con él se ha llevado a cabo el algoritmo mediante el cual se han ajustado los parámetros de los distintos modelos de redes neuronales. El conjunto de test a su vez se utiliza, por un lado, para validar el modelo durante el entrenamiento (en este caso el 100 % del conjunto). Por otro lado, se utiliza el 80 % como conjunto de clasificación. El otro 20 % del conjunto de test será utilizado para entrenar el parámetro T empleado en la técnica de calibración *Temperature Scaling*.

En el presente proyecto se han llevado a cabo las pruebas con dos *datasets* bastante populares en el campo de las redes neuronales convolucionales, y que guardan bastantes similitudes entre ellos, pero también alguna diferencia, lo cual hacía bastante interesante mostrar una comparativa entre ambos conjuntos de datos. Estos son **CIFAR10** y **CIFAR100**

4.2.1. CIFAR10

El conjunto de datos CIFAR-10 es una colección de imágenes usada para el entrenamiento en aprendizaje automático y visión por computadora.

El conjunto de datos CIFAR-10 consta de 60000 imágenes en color de 32x32 en 10 clases, con 6000 imágenes por clase. Hay 50000 imágenes de entrenamiento y 10000 imágenes de prueba. [19]. El conjunto de datos se divide en cinco lotes de entrenamiento y un lote de prueba, cada uno con 10000 imágenes. El lote de prueba contiene 1000 imágenes seleccionadas al azar de cada clase. Los lotes del conjunto de entrenamiento contienen las imágenes restantes en orden aleatorio, pero algunos lotes de entrenamiento pueden contener más imágenes de una clase que de otra. Los lotes de entrenamiento contienen exactamente 5000 imágenes de cada clase.

4.2.2. CIFAR100

CIFAR-100 es exactamente igual a CIFAR-10, exceptuando que tiene 100 clases, cada una con 600 imágenes (500 para el conjunto de entrenamiento y 100 para el conjunto de prueba). Las 100 clases de CIFAR-100 se agrupan en 20 superclases. Dentro de cada superclass hay 5 clases, que hacen las 100 totales. En CIFAR100, cada imagen viene con dos etiquetas: una '*fine label*', que indica la clase a la que pertenece y una '*coarse label*', que indica la superclass a la que pertenece.

4.3. Transformaciones aplicadas

Antes de empezar el entrenamiento de las redes neuronales, hay que preparar el *dataset* que van a procesar las redes neuronales. Se deben seguir unos pasos previos.

El primer paso será definir las transformaciones de los datos que se van a procesar. Es importante mencionar que no se aplican las mismas transformaciones en el conjunto de entrenamiento que en el conjunto de clasificación. Para el conjunto de entrenamiento se aplican las transformaciones empleando el módulo *torch.transforms*. Las transformaciones aplicadas son las siguientes:

- 1.– *transforms.randomCrop(imgSize, padding)*, dada una imagen, devuelve esta imagen en una localización distinta, asignada de manera aleatoria.
- 2.– *transforms.RandomHorizontalFlip(p)*, dada una imagen, gira esta imagen con una probabilidad determinada.
- 3.– *transforms.toTensor()*, pasa los datos a un formato Tensor para ser manejados por la red, lo que facilita las operaciones.

4.– `transforms.Normalize(mean, variance)`, normaliza los valores de la imagen, que pasan de estar en el intervalo [0,255] a estarlo en el intervalo [0,1], usando para ello una distribución con media *mean* y varianza *variance*.

Las dos primeras responden a la aplicación del método de aumento de datos, mientras que las dos últimas son la adaptación de los datos a la entrada de la red, normalizando sus valores y guardándolos en un formato que facilitará el procesamiento de estos.

Como se ha dicho, las transformaciones no son las mismas para el conjunto de entrenamiento que para el conjunto de pruebas. Para este segundo solo se llevan a cabo las dos últimas (el aumento de datos solo se aplica en el conjunto de entrenamiento). Es importante que al normalizar los datos, la media y la varianza sea igual en ambos conjuntos.

4.4. Entrenamiento de los modelos

Una vez aplicadas las transformaciones y cargados los *datasets* en sus correspondientes *DataLoaders*, el siguiente paso de todas las pruebas es generar N modelos donde N será el tamaño del *ensemble* (en este proyecto se ha establecido N = 5), y llevar a cabo su entrenamiento. El entrenamiento llevado a cabo es común para todos los modelos de redes neuronales entrenados. El algoritmo empleado para llevar a cabo este es el *backpropagation*. Lo primero será definir una serie de componentes. El primero de ellos será una variable *criterion*, que no es más que la función de coste asociada. En este proyecto se trabajará con la **CrossEntropyLoss**. Esta nos va a permitir calcular los gradientes para la optimización de pesos. El siguiente componente a definir es el optimizador. Se empleará el **SGD** (Descenso por Gradiente Estocástico). Con este se irán actualizando los pesos de la red de tal forma que la salida de la función de coste sea cada vez menor.

Una vez definidos estos componentes el entrenamiento de la red es sencillo. Se calcula la salida de la red, se pasa por la función de coste y se calcula su gradiente. Se invoca al optimizador para que actualice los pesos (mediante la sentencia *step*) y se reinicia el valor del gradiente para que no se acumulen. Una vez terminadas las épocas que se hayan establecido (por defecto serán 250), el entrenamiento de la red neuronal se dará por concluido.

4.5. Métricas de evaluación

Una vez entrenados todos los modelos de redes neuronales, se procede a su evaluación. Mediante otro script, se cargará cada uno de los modelos generados en los entrenamientos y se calcularán las métricas con las que se va a evaluar el modelo.

La primera métrica que se calculará es la **tasa de aciertos**. Viene dada por la fórmula 4.1. Indica cuanto de preciso es el modelo, cuantos aciertos genera para el *dataset* dado.

$$Acc = \frac{nAciertos}{nMuestrasTotal} \quad (4.1)$$

Se mencionó que uno de los objetivos de este trabajo era definir modelos que generen predicciones confiables. Atender a la tasa de aciertos es correcto, pero no hay que detenerse ahí. Hay que tener en cuenta la confianza del modelo y que esta sea lo más pareja a la tasa de aciertos, es decir, que estén calibradas: un modelo preciso y fiable.

Para estudiar la calibración de un modelo hay dos métricas fundamentales: el error de calibración esperado (**ECE**) y el error de calibración máximo (**MCE**).

El **ECE** calcula, para las predicciones de una red neuronal divididas en M bloques iguales, la diferencia absoluta entre su porcentaje de acierto y su confianza. Indica cuál es la descalibración real del modelo (la diferencia exacta entre su precisión y su confianza). Viene dado por la siguiente fórmula

$$ECE = \sum_{m=1}^M \frac{B_m}{n} * |acc(B_m) - conf(B_m)| \quad (4.2)$$

siendo n el número de muestras.

Otra medida a tener en cuenta será el **MCE**. El MCE indica el máximo error posible, o lo que es lo mismo, el peor caso que puede darse. Gráficamente es la distancia entre la barra del histograma más alejada de la recta de perfecta calibración. Viene dado por la siguiente fórmula:

$$MCE = max_{m \in (1...M)} |acc(B_m) - conf(B_m)| \quad (4.3)$$

4.6. Técnicas de optimización

Una red a priori no suele tener un rendimiento perfecto. Será necesario aplicar una serie de optimizaciones, tanto en el entrenamiento como en la posterior calibración del modelo.

A la hora de entrenar las redes neuronales, pytorch ofrece un módulo llamado *optimizer*. En este caso se empleará como optimizador el *SGD* (*Stochastic gradient descent* o descenso por gradiente estocástico). Funciona de forma similar al descenso por gradiente, la principal diferencia es que reemplaza el gradiente real por una aproximación de este. El optimizador cuenta con dos hiperparámetros que habrá que establecer antes del entrenamiento: **la tasa de aprendizaje** y el **momentum**. Una vez establecidos, él mismo se encarga de optimizar la actualización de pesos en cada iteración. Otra optimización empleada en el entrenamiento de los modelos es el *batch normalization* o normalización por lotes. Esta viene incluida en el diseño de los modelos entrenados y consiste básicamente en normalizar y centrar las salidas de cada capa convolucional, haciendo más rápido el entrenamiento sin perder eficacia. Estos son los métodos que han sido aplicados en entrenamiento de las redes neuronales para optimizar el proceso y obtener una mayor eficacia en la fase de explotación de la red.

Una vez entrenados los modelos y calculada su tasa de acierto. Se procede a estimar las medidas de calibración de los modelos. Como se ha visto previamente, las redes neuronales modernas han visto incrementada su precisión, pero por contra su calibración ha empeorado. Partiendo de esa base, se obtendrán unos valores para el ECE y MCE que habrá que mejorar (minimizar). Existen múltiples métodos para ello. En este proyecto se ha escogido el método de Temperature Scaling, que se explicará y que será con el que se trabaje en los experimentos. Se ha escogido este método, puesto que es el que más eficacia presenta, siendo además el más sencillo de implementar, en comparación con otros métodos de calibración para problemas multiclase como son *Matrix* y *Vector Scaling*

El método de Temperature Scaling, en física, es utilizado para calibrar temperaturas. Esto se traslada de forma íntegra a las redes neuronales. Este método equilibrará las salidas de la red neuronal. Para ello se empleará un parámetro T . En la sección 4.2 se habló de un conjunto de validación que suponía el 20 % de las muestras del conjunto de clasificación*. Bien, pues con este conjunto se entrenará el parámetro T de tal forma que se vaya minimizando la probabilidad logarítmica negativa o NLL (vista en B.1). Este problema es convexo, tiene un mínimo global, lo cual significa que, empleándose cualquier método, se tiene que llegar al mismo valor, que será el que haga mínima esta medida.

Una vez entrenado el parámetro T , simplemente hay que multiplicar las predicciones obtenidas a la salida de la red por el mismo. La versión original del Temperature Scaling es dividiendo. En este TFG se ha optado por emplear la multiplicación dado que proporcionaba mejores resultados que la división para las pruebas realizadas. Por tanto, el método de Temperature Scaling empleado es el siguiente:

$$P(y) = \frac{m_i^{z*T}}{\sum_j m_i^{z_j*T}} \quad (4.4)$$

donde $P(y)$ es la predicción equilibrada, z es el logit inicial y T el parámetro regulador.

A su vez, se podrá observar en las tablas expuestas en el capítulo 5 que el método *Temp. Scaling* no influye en la tasa de aciertos del modelo, ya que no altera el valor máximo de la softmax, es decir, la clase predicha [10], al multiplicarse todos los valores por el mismo parámetro. Esto hará más útil este método, pues si ya se dispone de una tasa de aciertos muy buena, no habrá que preocuparse de ella a la hora de aplicarlo.

*Es muy importante tener en cuenta que el parámetro T no puede ajustarse en el entrenamiento del modelo para generar muestras ya calibradas. Esto tiene como explicación el principio de sobre-entrenamiento. Si se realizase de esta forma, las predicciones para el conjunto de entrenamiento tendrían una calibración muy buena, pero esto no se trasladaría al conjunto de clasificación. Por eso es fundamental entrenar y ajustar el parámetro T con un conjunto de datos arbitrario, que no sea ni el de entrenamiento ni el de clasificación (será por tanto, el de validación)

RESULTADOS

En esta sección se van a mostrar los distintos resultados obtenidos durante las pruebas. Por un lado se mostrarán una serie de tablas con las medidas obtenidas en las distintas pruebas. Por otro lado se adjuntan varias gráficas para llevar a cabo una comparativa y, de forma más visible, comprobar los efectos de las distintas pruebas.

5.1. Resultados obtenidos

5.1.1. ResNet18

Las tablas 5.1(a) y 5.1(b) muestran, para CIFAR-10 y CIFAR-100 respectivamente, los valores de las métricas presentadas en la sección 4.5 antes de aplicar *Temperature Scaling*, mientras que las tablas 5.2(a) y 5.2(b) los muestran una vez aplicada dicha técnica de calibración.

Como puede observarse, la tasa de aciertos de cada modelo es similar, rondando el 92% en CIFAR-10 y el 72% en CIFAR-100. Atendiendo a las métricas óptimas del repositorio del cual se ha extraído el código de la red [20], donde se indica que la métrica máxima obtenida es del 93,02% en CIFAR-10, se pueden dar por válidos los valores obtenidos. En CIFAR-100 el porcentaje obtenido baja, como era de esperar, dado que clasificar 100 clases es más complicado que clasificar 10. Aun así se obtienen unos resultados bastante decentes, teniendo en cuenta que el estado del arte de la ResNet18 rondará el 74% para este conjunto de datos, sabiendo que la ResNet-1001 llega al 77.3%.

Puede observarse que, a su vez, la tasa de aciertos del *ensemble* mejora la tasa de los modelos individuales.

En cuanto a las medidas de calibración de las ResNet18, se puede observar que para ambos conjuntos de datos el *ensemble* mejora las métricas de los modelos individuales, por la misma razón por la que mejoraba la tasa de aciertos. En cuanto a la técnica *Temperature Scaling*, se puede ver como mejora las métricas de forma notoria en ambos conjuntos (sobre todo en CIFAR-100). Sin embargo, a la hora de calibrar el *ensemble*, como era de esperar, las métricas empeoran.

En el apéndice A se pueden observar los histogramas del error de calibración esperado, tanto antes (A.5(a) y A.5(c)), como después de aplicar el método de calibración (A.5(b) y A.5(d)). Puede observarse una gran diferencia.

Por otro lado, en la figura A.1 se muestra la evolución de las gráficas del error de calibración para ambos conjuntos de datos, antes y después de aplicar la técnica de calibración ya mencionada.

Nº	Acc (%)	ECE (%)	MCE (%)
1	91.0	6.85	3.10
2	92.26	6.00	2.85
3	91.16	6.97	3.51
4	92.22	5.84	2.80
5	91.18	6.81	3.15
ens	93.38	2.27	0.57

(a) Resultados de la Resnet18 en CIFAR-10 **antes** de aplicar **Temp. Scaling**

Nº	Acc (%)	ECE (%)	MCE (%)
1	72.67	21.08	8.13
2	72.37	21.17	8.16
3	72.70	20.78	8.07
4	72.21	21.40	8.71
5	72.17	21.46	8.34
ens	76.49	6.92	1.73

(b) Resultados de la Resnet18 en CIFAR-100 **antes** de aplicar **Temp. Scaling**

Tabla 5.1: Tablas con los resultados obtenidos para la ResNet18 sobre CIFAR-10 (5.1(a)) y CIFAR-100 (5.1(b)) previos a la aplicación de la técnica de calibración *Temperature Scaling*

Nº	Acc (%)	ECE (%)	MCE (%)
1	91.00	1.90	0.51
2	92.26	2.27	0.59
3	91.16	2.42	0.66
4	92.22	1.98	0.48
5	91.18	1.36	0.33
ens	93.38	5.11	1.12

(a) Resultados de la Resnet18 en CIFAR-10 **después** de aplicar **Temp. Scaling**

Nº	Acc (%)	ECE (%)	MCE (%)
1	72.67	4.70	0.72
2	72.37	3.02	0.60
3	72.70	2.41	0.56
4	72.21	2.96	0.58
5	72.17	2.58	0.64
ens	76.49	10.22	1.76

(b) Resultados de la Resnet18 en CIFAR-100 **después** de aplicar **Temp. Scaling**

Tabla 5.2: Tablas con los resultados obtenidos para la ResNet18 sobre CIFAR-10 (5.2(a)) y CIFAR-100 (5.2(b)) después de aplicar la técnica de calibración *Temperature Scaling*

5.1.2. ResNet50

Las tablas 5.3(a) y 5.3(b) muestran, de la misma forma que se ha visto para la ResNet18, los resultados de la ResNet50 para CIFAR-10 y CIFAR-100 respectivamente antes de aplicar *Temperature Scaling*, mientras que las tablas 5.4(a) y 5.4(b) los muestran una vez aplicada dicha técnica de calibración.

Puede observarse que el rendimiento de la ResNet50 es ligeramente mejor que el de la ResNet18 en cuanto a tasa de aciertos se refiere. Mejora ligeramente ese porcentaje, lo cual cuadra con la cifra estipulada en el repositorio GitHub [20], donde se indica que la mejor tasa de aciertos obtenida es del 93.63 % para CIFAR-10. Para CIFAR-100, teniendo en cuenta que la *ResNet50 Without Transfer Learning* alcanza el 67 % de tasa de aciertos y que la ResNet-1001 llega al 77.3 %, se estima que el estado del arte de la ResNet50 rondará el 74 % de acierto para este Datasets, por lo que los resultados obtenidos pueden darse por buenos.

De nuevo se aprecia el mismo comportamiento con el *ensemble*, el cual mejora el porcentaje de los modelos por separado en ambos conjuntos.

Las medidas de calibración que se obtienen son bastante parecidas a la ResNet18. Como se ha mencionado en la sección 2.3, a medida que aumenta la profundidad el error de calibración es mayor, si bien se llega un punto (en torno a las 30 capas de profundidad) en que el error se acaba estabilizando [10]. Pese a que debería ser ligeramente mayor el error en ResNet50, se puede ver que los resultados obtenidos en CIFAR-10 son muy parejos mientras que en CIFAR-100 si se aprecia ese comportamiento. Nuevamente, el *ensemble* mejora la calibración de los modelos individuales. De la misma forma, el *Temperature Scaling* mejora la calibración de los modelos, no así del *ensemble*, que ve empeorados sus resultados nuevamente.

En la figura A.2 se muestran los histogramas del ECE de la ResNet50 para ambos conjuntos de datos, antes y después de aplicar *Temp. Scaling*. Nuevamente se puede apreciar una importante mejora, al igual que sucedía en ResNet18.

Nº	Acc (%)	ECE (%)	MCE (%)
1	92.46	5.84	4.71
2	92.50	6.13	5.07
3	92.38	5.88	4.69
4	93.01	6.48	4.96
5	92.68	6.81	3.15
ens	94.16	1.32	0.98

(a) Resultados de la Resnet50 en CIFAR-10 **antes** de aplicar **Temp. Scaling**

Nº	Acc (%)	ECE (%)	MCE (%)
1	73.75	21.59	14.50
2	72.21	22.49	14.86
3	72.30	25.12	16.84
4	75.06	20.30	13.79
5	72.98	22.62	14.52
ens	76.89	6.25	2.17

(b) Resultados de la Resnet50 en CIFAR-100 **antes** de aplicar **Temp. Scaling**

Tabla 5.3: Tablas con los resultados obtenidos para la Resnet50 sobre CIFAR-10 (5.3(a)) y CIFAR-100 (5.3(b)) previos a la aplicación de la técnica de calibración *Temperature Scaling*

Nº	Acc (%)	ECE (%)	MCE (%)
1	92.46	1.54	0.36
2	92.50	1.87	0.67
3	92.38	1.54	0.46
4	93.01	1.87	0.45
5	92.68	1.32	0.38
ens	94.16	1.32	0.98

(a) Resultados de la Resnet50 en CIFAR-10 **después** de aplicar
Temp. Scaling

Nº	Acc (%)	ECE (%)	MCE (%)
1	73.75	2.03	0.36
2	72.21	6.50	1.08
3	72.30	4.98	1.45
4	75.06	6.10	2.16
5	72.98	3.66	0.69
ens	76.89	14.98	2.35

(b) Resultados de la Resnet50 en CIFAR-100 **después** de aplicar
Temp. Scaling**Tabla 5.4:** Tablas con los resultados obtenidos para la Resnet50 sobre CIFAR-10 (5.4(a)) y CIFAR-100 (5.4(b)) después de aplicar la técnica de calibración *Temperature Scaling*

5.1.3. DenseNet121

La DenseNet121 debería, a priori, arrojar unos resultados mejores que los de los dos modelos ResNet, puesto que su profundidad y n.^o de parámetros invita a pensar que se tratará de una red con una precisión mayor.

Así puede verse en las tablas 5.5 y 5.6. Se aprecia que para ambos conjuntos de datos las tasas de acierto son mejores que en ResNet, llegando el *ensemble* de DenseNet121 hasta el 95.24 % de tasa de aciertos, (en el repositorio GitHub se estima que un modelo DenseNet121 podía llegar al 95.04 %). Para CIFAR-100 el estado del arte de la DenseNet121 es ligeramente superior a las ResNet, rondando el 77 %. Los resultados obtenidos se acercan a esa cifra, como puede verse.

El comportamiento vuelve a repetirse en cuanto a medidas de calibración se refiere. El *ensemble* mejora las métricas. Al aplicarse *Tem. Scaling*, los modelos mejoran su calibración. El *ensemble* nuevamente se descalibra.

La figura A.3 representa, de la misma forma que se hacía con las ResNet, el histograma del ECE. De la misma forma, se puede apreciar una mejora considerable tras aplicar el *Temp. Scaling*.

Nº	Acc (%)	ECE (%)	MCE (%)
1	94.13	4.60	3.75
2	94.10	4.58	3.51
3	94.11	4.68	3.97
4	94.12	4.65	3.91
5	93.86	4.92	3.97
ens	95.24	1.18	0.93

(a) Resultados de la DenseNet121 en CIFAR-10 **antes** de aplicar **Temp. Scaling**

Nº	Acc (%)	ECE (%)	MCE (%)
1	74.93	19.73	12.79
2	74.72	19.20	13.00
3	74.52	19.61	13.47
4	75.70	20.08	13.59
5	76.06	19.61	13.23
ens	78.25	5.76	2.39

(b) Resultados de la DenseNet121 en CIFAR-100 **antes** de aplicar **Temp. Scaling****Tabla 5.5:** Tablas con los resultados obtenidos para la DenseNet121 sobre CIFAR-10 (5.5(a)) y CIFAR-100 (5.5(b)) previos a la aplicación de la técnica de calibración *Temperature Scaling*

Nº	Acc (%)	ECE (%)	MCE (%)
1	94.13	1.35	0.47
2	94.10	1.73	0.57
3	94.11	0.98	0.29
4	94.12	1.11	0.41
5	93.86	1.26	0.69
ens	95.24	8.46	2.85

(a) Resultados de la DenseNet121 en CIFAR-10 **después** de aplicar **Temp. Scaling**

Nº	Acc (%)	ECE (%)	MCE (%)
1	74.93	3.10	0.58
2	74.72	6.06	1.14
3	74.52	2.50	0.55
4	75.70	4.42	0.64
5	76.06	2.42	0.45
ens	78.25	10.59	1.72

(b) Resultados de la DenseNet121 en CIFAR-100 **después** de aplicar **Temp. Scaling****Tabla 5.6:** Tablas con los resultados obtenidos para la DenseNet121 sobre CIFAR-10 (5.6(a)) y CIFAR-100 (5.6(b)) después de aplicar la técnica de calibración *Temperature Scaling*

5.1.4. EfficientNetB0

Para EfficientNetB0 se esperan, como puede verse en la gráfica comparativa 3.7 unos resultados similares a los obtenidos para la ResNet50. Se puede observar que las tasas de aciertos son similares, quizá algo mejores las de ResNet50 para ambos conjuntos de datos. En cuanto a las medidas de calibración, para CIFAR-10 la EfficientNetB0 se mantiene en la línea del resto de redes neuronales. Sin embargo, para CIFAR-100 se experimenta una notable mejora. En la sección 2.3 se ha visto como la profundidad y la anchura eran factores que empeoraban la calibración de las redes neuronales. Y es sabido que la EfficientNetB0 se caracteriza por un escalado de profundidad y anchura muy optimizado. Por ello su calibración es mejor que los anteriores modelos. En esta red se muestra también la efectividad del *Temperature Scaling* así como del *ensemble* de los modelos, que mejora las tasas de acierto y la calibración. Sin embargo vuelve a repetirse que al calibrar el *ensemble*, este empeora su calibración.

Las gráficas de la figura A.4 muestran la diferencia de la calibración de la red tras aplicar el método de calibración.

Nº	Acc (%)	ECE (%)	MCE (%)
1	92.31	5.70	4.54
2	90.81	7.11	5.88
3	91.58	6.62	5.29
4	91.65	6.60	5.18
5	92.27	6.10	4.73
ens	93.86	1.54	1.03

(a) Resultados de la EfficientNetB0 en CIFAR-10 **antes** de aplicar **Temp. Scaling**

Nº	Acc (%)	ECE (%)	MCE (%)
1	69.95	9.11	2.66
2	70.56	8.72	2.22
3	70.30	9.20	2.30
4	70.97	9.86	2.64
5	70.83	8.89	2.33
ens	76.22	4.72	1.39

(b) Resultados de la EfficientNetB0 en CIFAR-100 **antes** de aplicar **Temp. Scaling**

Tabla 5.7: Tablas con los resultados obtenidos para la EfficientNetB0 sobre CIFAR-10 (5.7(a)) y CIFAR-100 (5.7(b)) previos a la aplicación de la técnica de calibración *Temperature Scaling*

Nº	Acc (%)	ECE (%)	MCE (%)
1	92.31	2.00	0.69
2	90.81	1.84	0.93
3	91.58	1.04	0.69
4	91.65	0.70	0.34
5	92.27	1.02	0.46
ens	93.86	5.01	1.51

(a) Resultados de la EfficientNetB0 en CIFAR-10 **después** de aplicar **Temp. Scaling**

Nº	Acc (%)	ECE (%)	MCE (%)
1	69.95	1.21	0.31
2	70.56	2.24	0.42
3	70.30	0.65	0.18
4	70.97	1.10	0.30
5	70.83	1.23	0.32
ens	76.22	18.22	2.61

(b) Resultados de la EfficientNetB0 en CIFAR-100 **después** de aplicar **Temp. Scaling****Tabla 5.8:** Tablas con los resultados obtenidos para la EfficientNetB0 sobre CIFAR-10 (5.8(a)) y CIFAR-100 (5.8(b)) después de aplicar la técnica de calibración *Temperature Scaling*

5.1.5. UpaNet

La UpaNet es, como se ha visto (sección 3.2.4), la red neuronal más eficaz de todas las presentadas en el proyecto. Combina de forma efectiva las principales ventajas de los modelos ResNet y DenseNet. Esto se traduce en un mejor rendimiento sin ver incrementado de forma excesiva el tiempo y recursos para su entrenamiento.

Se puede corroborar lo esperado inicialmente. Todos los modelos UpaNet presentan una tasa de aciertos que se mantiene de forma asidua en el 95 %. La cifra estipulada para la tasa de aciertos en CIFAR-10 es 96.47 % [21] mientras que para CIFAR-100 se sitúa en torno al 80.29 % [22]

En las tablas 5.9 y 5.10 se pueden ver los resultados obtenidos. Se trata, como puede verse, de una red que mejora las tasas de aciertos de todas las redes vistas hasta ahora.

Y no solo eso, se trata de una red con una calibración inicial mejor que la de resto de modelos con diferencia. Es por ello que el método *Temp. Scaling* tiene menor efecto en esta red, pese a mejorar las métricas. Como puede verse en dichas tablas y en las gráficas de la figura A.5, la diferencia entre aplicar el método y no aplicarlo es menor que en los demás modelos, ya que tiene menos margen de mejora.

Lo que si es común a todos los modelos es el comportamiento del *ensemble*: mejora la tasa de aciertos, tiene mejor calibración que los modelos individuales, pero a la hora de aplicar *Temp Scaling*, se descalibra.

Nº	Acc (%)	ECE (%)	MCE (%)
1	95.13	2.95	2.08
2	95.12	2.92	1.85
3	95.03	3.18	2.29
4	95.18	2.89	1.94
5	95.33	2.44	1.52
ens	96.19	2.11	1.25

(a) Resultados de la UpaNet en CIFAR-10 **antes** de aplicar
Temp. Scaling

Nº	Acc (%)	ECE (%)	MCE (%)
1	76.95	8.41	3.60
2	76.04	8.60	3.22
3	76.21	8.02	2.86
4	76.43	8.53	3.32
5	76.24	8.54	3.40
ens	81.50	3.29	0.51

(b) Resultados de la UpaNet en CIFAR-100 **antes** de aplicar
Temp. Scaling

Tabla 5.9: Tablas con los resultados obtenidos para la UpaNet sobre CIFAR-10 (5.9(a)) y CIFAR-100 (5.9(b)) previos a la aplicación de la técnica de calibración *Temperature Scaling*

Nº	Acc (%)	ECE (%)	MCE (%)
1	95.13	1.58	1.16
2	95.12	1.28	0.76
3	95.03	1.60	0.75
4	95.18	1.46	0.94
5	95.33	1.35	0.98
ens	96.19	16.05	10.38

(a) Resultados de la UpaNet en CIFAR-10 **después** de aplicar
Temp. Scaling

Nº	Acc (%)	ECE (%)	MCE (%)
1	76.95	1.18	0.29
2	76.04	1.21	0.35
3	76.21	0.97	0.31
4	76.43	1.15	0.26
5	76.24	1.21	0.24
ens	81.50	9.56	1.43

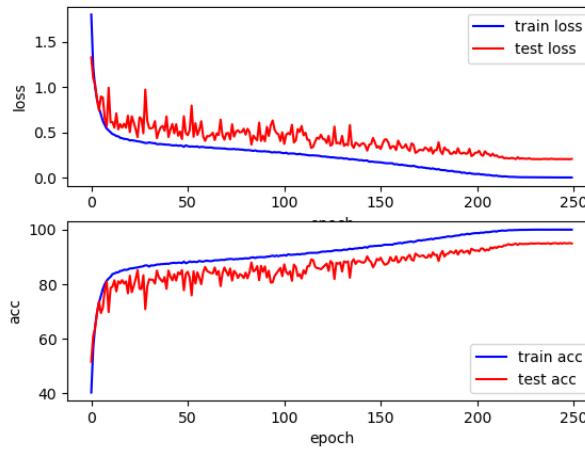
(b) Resultados de la UpaNet en CIFAR-100 **después** de aplicar
Temp. Scaling

Tabla 5.10: Tablas con los resultados obtenidos para la upanets sobre CIFAR-10 (5.10(a)) y CIFAR-100 (5.10(b)) después de aplicar la técnica de calibración *Temperature Scaling*

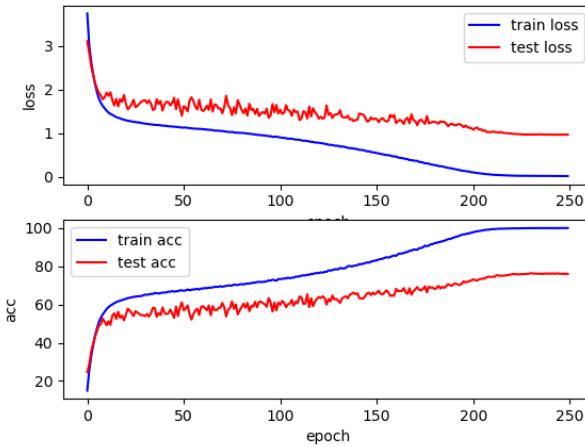
5.2. Análisis de los resultados obtenidos

Tras haber presentado los resultados de las redes neuronales, se puede sacar en claro el comportamiento que llevan a cabo los modelos de redes neuronales.

En la sección 2.3 se ha mencionado que el sobreentrenamiento de las redes neuronales estaba estrechamente relacionado con la descalibración de estas. Esto se puede observar gracias a las tablas de resultados y a las gráficas de la figura 5.1. Se muestra aquí el caso de la UpaNet, pero el comportamiento es el mismo para todas las redes.



(a) Evolución del entrenamiento en CIFAR-10



(b) Evolución del entrenamiento en CIFAR-100

Figura 5.1: Gráficas correspondientes a la progresión del error y de la tasa de aciertos en UpaNet para el conjunto de entrenamiento y el conjunto de clasificación

Se puede observar que la red presenta sobre entrenamiento. En CIFAR-10 (figura 5.1(a)) el grado de sobre entrenamiento es menor que en CIFAR-100 (figura 5.1(b)). Esto se puede apreciar viendo que la progresión del error y de la tasa de aciertos para el conjunto de clasificación (color rojo) y el conjunto de entrenamiento (color azul) es más pareja en las gráficas de la izquierda que en las de la derecha.

Observando ahora las tablas 5.9(a) y 5.9(b), se puede ver como la descalibración es mayor en CIFAR-100 que en CIFAR-10, siguiendo el grado de sobre entrenamiento que se detecta en las gráficas.

También es común a todos los modelos de redes neuronales el hecho de que el *ensemble* mejora la tasa de acierto de los modelos individuales y presenta un mejor nivel de calibración. Esto tiene un denominador común. Ciertos modelos de aprendizaje automático son propensos a sobre ajustarse. Los modelos aprenden muy bien los datos de entrenamiento, pero no funcionan igual de bien con datos nuevos (conjunto de prueba). Se dice aquí que la varianza de dichos modelos es alta, es decir, los resultados y la precisión de los modelos varía mucho. [23]. El *ensemble* mejora sus números reduciendo su varianza, de tal forma que incrementa su precisión y mejora su calibración.

Esto se debe a que las predicciones del *ensemble* provienen de las N predicciones (en este caso N=5) de los distintos modelos, mientras que las predicciones de cada modelo solo cuentan con su propia predicción. Esto ayuda al *ensemble* a eliminar el ruido de cada una de las muestras de los modelos.

Se observa que las redes neuronales no están bien calibradas de por sí. Aplicando la técnica de calibración *Temp. Scaling*, los resultados mejoran de forma clara, tal y como se puede ver en las tablas de la sección 5.1 y en los histogramas ubicados en el apéndice A.

Sin embargo, a la hora de aplicar la calibración a los distintos *ensembles*, este se descalibra, a diferencia de los modelos individuales, cuyos ECEy MCE se veían reducidos. En este contexto, la pregunta que se plantea es: ¿por qué el *ensemble*, que mejora el comportamiento de los modelos individuales, combinado con el *Temp. Scaling*, que mejora la calibración de los modelos, da lugar a un modelo peor calibrado?

CONCLUSIONES Y TRABAJO FUTURO

6.1. Conclusiones

A lo largo de este trabajo se han llevado una serie de tareas, como se ha mencionado en el capítulo 1.1. No era objeto de este trabajo el diseño de redes neuronales convolucionales, pero si el estudio de ciertos aspectos relacionados con ellas.

En un primer lugar se ha estudiado el entrenamiento de las redes neuronales convolucionales. Para ello fue fundamental el estudio del estado del arte, en el cual se lleva a cabo un repaso de los conceptos más importantes acerca de estas arquitecturas, necesarios para poder abordar ciertas técnicas. Una vez estudiados estos conceptos, se procedió a llevar a cabo la idea en torno a la cual gira este proyecto: el estudio de redes convolucionales en el estado del arte y la aplicación de técnicas de optimización.

Inicialmente presentan varios modelos de redes convolucionales y se lleva a cabo el entrenamiento de los mismos para obtener sus tasas de aciertos. Tras generar varios modelos de cada red, se propone el concepto de *ensemble* y se lleva a cabo la generación de un *ensemble* de 5 modelos para cada topología de red. Se ve que trabajar con *ensembles* presenta mejores resultados que trabajar con modelos de forma individual.

El proyecto no se detiene aquí. Ahora se pretende lograr que los modelos de redes neuronales, además de ser precisos, sean fiables. Se introduce el concepto de calibración, para entender que ventajas aporta un modelo bien calibrado y entender como optimizar las redes convolucionales. Tras realizar un estudio del arte de la calibración de redes neuronales actuales, se evalúan las métricas de calibración y se propone un método que mejore los modelos: el *Temp. Scaling*. Se aplica tanto a los modelos como a los *ensembles*. Para sorpresa, pese a poder observar que se mejoran las métricas en los modelos individuales de forma clara, esta realidad no se refleja a la hora de aplicarse sobre los *ensembles*.

6.2. Trabajo futuro

Como trabajo futuro se propone continuar esta línea de investigación abierta. Entender el origen del problema de descalibración de los *ensembles* al aplicar el *Temp. Scaling* y por qué las técnicas de calibración se muestran tan efectivas en los modelos individuales y al ser aplicadas a *ensembles*, este se descalibra.

Se propondrá ampliar la investigación iniciada en este trabajo de fin de grado, planteando una serie de objetivos, enfocados en entender el problema de calibración de *ensembles*. A su vez, se tratará de apuntalar la investigación proponiendo métodos alternativos de calibración de redes neuronales que sea igual de efectiva para los modelos individuales y sea capaz de mejorar los *ensembles* de modelos.

La idea sería publicar un artículo sobre esto, ya que se trata de un tema completamente novedoso que está en plena investigación. Por tanto el trabajo futuro sería transformar la el enfoque descriptivo que tiene este Trabajo de Fin de Grado en un enfoque mucho más práctico y novedoso.

BIBLIOGRAFÍA

- [1] R. Marca, “El manchester city se refuerza con ¡astrofísicos!,” *Marca*, mar 2021.
- [2] “¿qué es una red neuronal?.” <https://www.codificandobits.com/blog/que-es-una-red-neuronal/>. (18-06-2021).
- [3] “Conceptos básicos sobre redes neuronales.” <https://vincentblog.xyz/posts/conceptos-basicos-sobre-redes-neuronales>. (12-06-2021).
- [4] “Regularización lasso l1, ridge l2 y elasticnet.” <https://www.iartificial.net/regularizacion-lasso-l1-ridge-l2-y-elasticnet/>. (12-06-2021).
- [5] D. A. van Dyk and X. L. Meng, “The art of data augmentation,” *Journal of Computational and Graphical Statistics*, vol. 10, jan 2012.
- [6] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors.” oct 1986.
- [7] “Red neuronal convolucional cnn.” <https://www.diegocalvo.es/red-neuronal-convolucional/>. (18-06-2021).
- [8] “Redes neuronales convolucionales en profundidad.” <https://datasmarts.net/es/redes-neuronales-convolucionales-en-profundidad/>. (15-06-2021).
- [9] “Redes neuronales convolucionales.” <https://www.juanbarrios.com/redes-neurales-convolucionales/>. (16-06-2021).
- [10] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, “On calibration of modern neural networks,” aug 2017.
- [11] “Los lenguajes de programación más populares en aprendizaje automático (machine learning).” <https://www.microsiervos.com/archivo/ordenadores/lenguajes-programacion-aprendizaje-automatico-machine-learning.html>. (18-06-2021).
- [12] “Documentación pytorch.” <https://pytorch.org/docs/stable/index.html>. (14-06-2021).
- [13] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition.” 2015.
- [14] “Transfer learning with resnet in pytorch.” <https://www.pluralsight.com/guides/introduction-to-resnet>. (18-06-2021).
- [15] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” jan 2018.
- [16] P. Ruiz, “Densenets,” *Harvard University*, 2018.
- [17] M. Tan and Q. V. Le, “Efficientnet: Rethinking model scaling for convolutional neural networks,” *ICML*, 2019.
- [18] C.-H. Tseng, S.-J. Lee, J.-N. Feng, S. Mao, Y.-P. Wu, J.-Y. Shang, M.-C. Tseng, and X.-J. Zeng, “Upanets: Learning from the universal pixel attention networks,” mar 2021.
- [19] “Tensorflow-cifar10.” <https://www.tensorflow.org/datasets/catalog/cifar10>. (15-06-2021).

- [20] “kuangliu- pytorch-cifar.” <https://github.com/kuangliu/pytorch-cifar/>. (17-06-2021).
- [21] “Image classification on cifar-10.” <https://paperswithcode.com/sota/image-classification-on-cifar-10>. (17-06-2021).
- [22] “Image classification on cifar-100.” <https://paperswithcode.com/sota/image-classification-on-cifar-100>. (17-06-2021).
- [23] S. Kumar, “Does ensemble models always improve accuracy?.” <https://medium.com/analytics-vidhya/does-ensemble-models-always-improve-accuracy-c114cdbdae77>. (18-06-2021).

APÉNDICES

APENDICE A: GRÁFICOS DE CONFIANZA DE LOS MODELOS

En este apéndice se incluyen los diagramas de fiabilidad de cada modelo para los conjuntos de datos utilizados, con la idea de complementar la información mostrada en las distintas tablas del capítulo 5. Este apéndice tiene como objetivo facilitar la comprensión de los valores mostrados en dichas tablas, con la idea de mostrar de una forma visual el efecto del método *Temp. Scaling*.

A.1. ResNet18

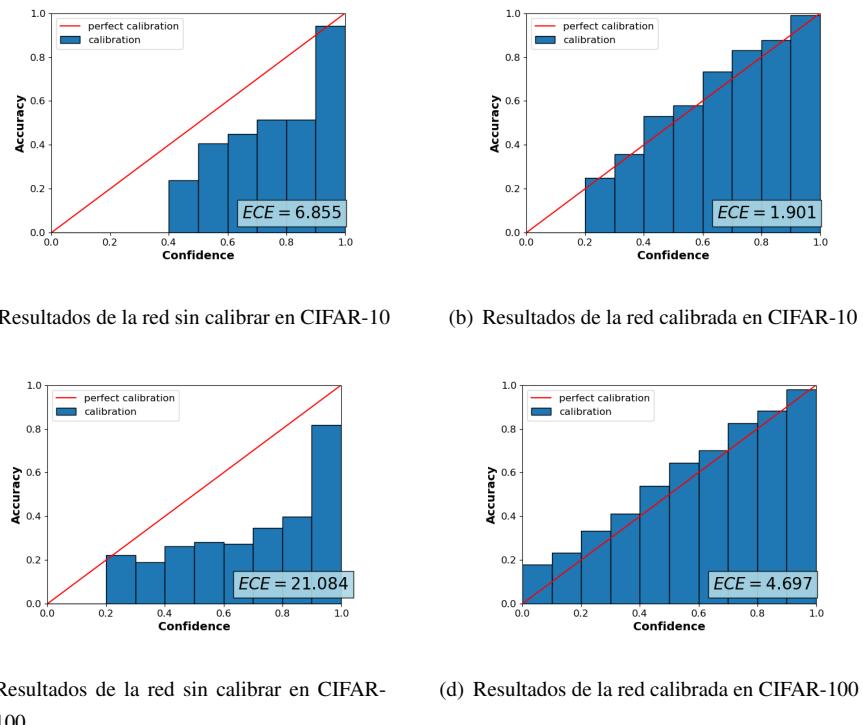
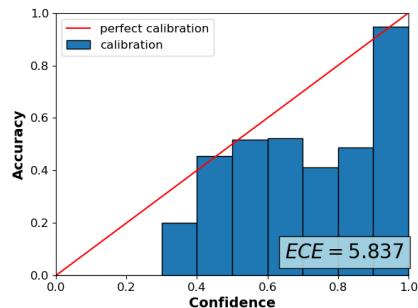
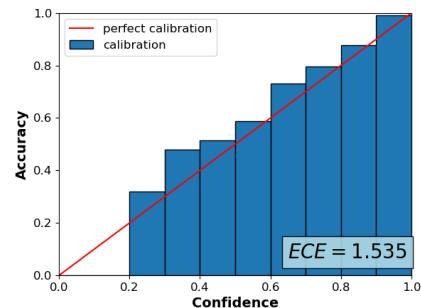


Figura A.1: Gráficas que muestran los resultados de la calibración de la **ResNet18**. Correspondientes al modelo 1.

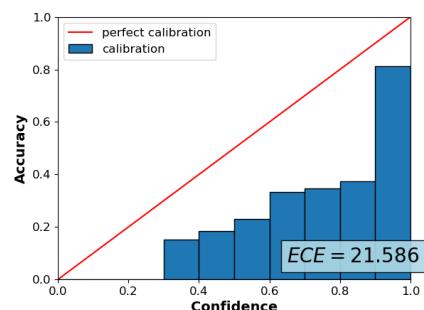
A.2. ResNet50



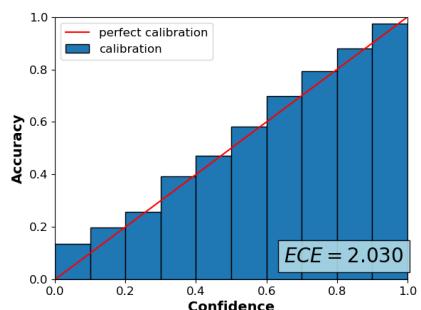
(a) Resultados de la red sin calibrar en CIFAR-10



(b) Resultados de la red calibrada en CIFAR-10



(c) Resultados de la red sin calibrar en CIFAR-100



(d) Resultados de la red calibrada en CIFAR-100

Figura A.2: Gráficas que muestran los resultados de la calibración de la **ResNet50**. Correspondientes al modelo 1.

A.3. DenseNet121

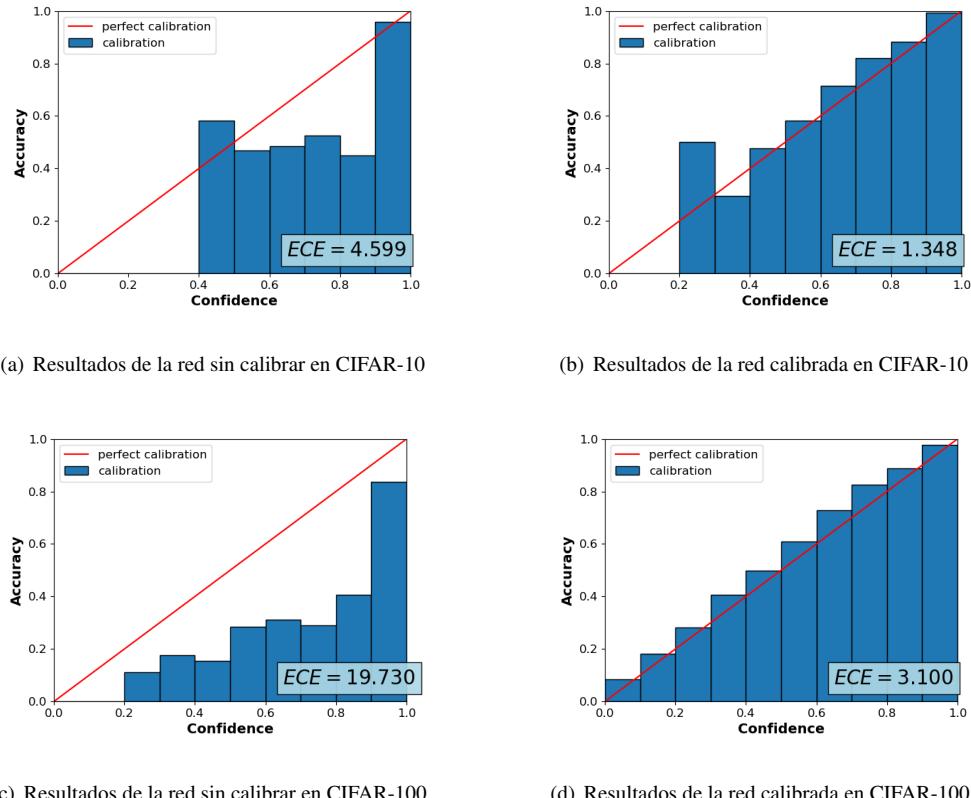
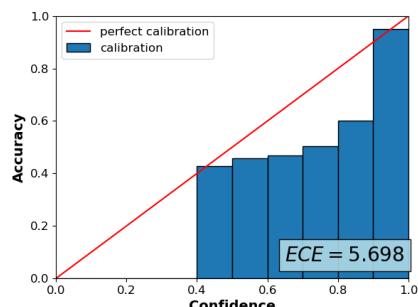
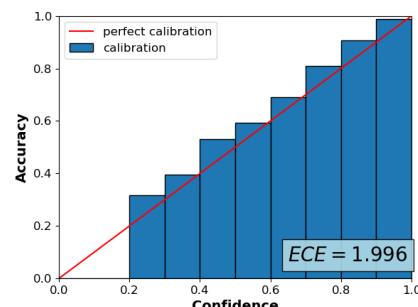


Figura A.3: Gráficas que muestran los resultados de la calibración de la **DenseNet121**. Correspondientes al modelo 1.

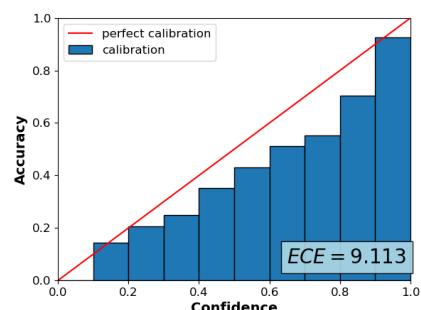
A.4. EfficientNetB0



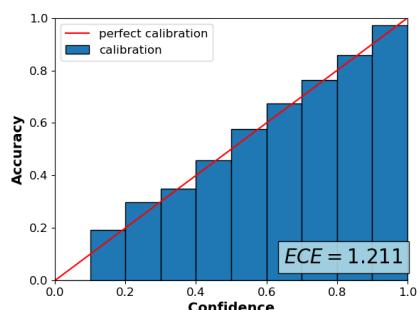
(a) Resultados de la red sin calibrar en CIFAR-10



(b) Resultados de la red calibrada en CIFAR-10



(c) Resultados de la red sin calibrar en CIFAR-100



(d) Resultados de la red calibrada en CIFAR-100

Figura A.4: Gráficas que muestran los resultados de la calibración de la **EfficientNetB0**. Correspondientes al modelo 1.

A.5. UpaNet

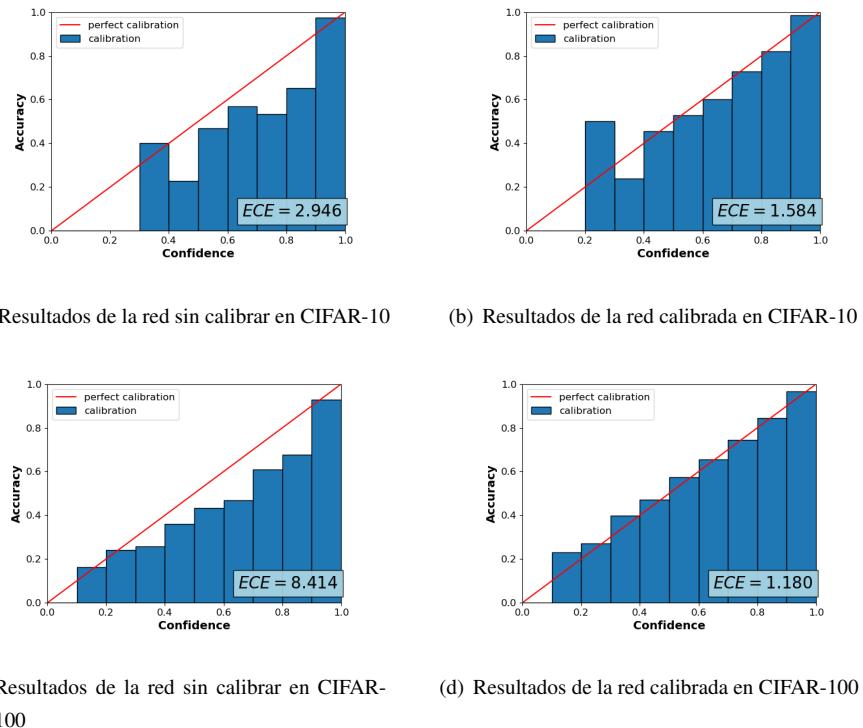


Figura A.5: Gráficas que muestran los resultados de la calibración de la **upanets**. Correspondientes al modelo 1.

APENDICE B: OTRAS MÉTRICAS DE CALIBRACIÓN EN REDES NEURONALES CONVOLUCIONALES

En la sección 4.5 se han presentado como métricas para evaluar los modelos la tasa de aciertos y respecto a la calibración, el error de calibración esperado (ECE) y el error de calibración máximo (MCE), que son las medidas más importantes a la hora de estudiar la calibración de las redes. En este apéndice se quiere completar la evaluación de la calibración de modelos estudiando otras dos medidas de calibración adicionales, que se explicarán a continuación.

B.1. NLL: Negative Log Likelihood

La NLL o función de verosimilitud logarítmica negativa es una medida empleada en la calibración de redes neuronales. Funciona de la misma forma que la función de coste asociada a la hora de obtener el mayor porcentaje de aciertos. El objetivo será minimizarla de la misma forma que se minimizaba la función de coste. El método *Temp. Scaling* lo que hace es entrenar un parámetro sobre un conjunto de datos minimizando la función de coste. Al aplicar dicho parámetro sobre las predicciones de un modelo, se está minimizando la NLL, lo que automáticamente se traduce en una mejora de la calibración de la red, reflejada en mejores ECE y MCE.

La métrica NLL viene dado por la siguiente fórmula:

$$\iota = - \sum_{i=1}^n \log(\pi(y_i|x_i)) \quad (\text{B.1})$$

B.2. BRIER

Se conoce el BRIER como una regla de puntuación adecuada para medir la precisión de las probabilidades predichas. Se calcula como el error al cuadrado de un vector de probabilidad predicho y otro vector objetivo. Viene dado por la siguiente fórmula:

$$BRIER = \frac{1}{N} \sum_{i=1}^N (f_i - o_i)^2 \quad (\text{B.2})$$

Se puede observar que su cálculo se asemeja bastante al del error cuadrático medio. De hecho, el BRIER es el cálculo del error cuadrático medio para vectores de más de una dimensión.

B.3. NLL y BRIER obtenidos en los experimentos

En la capítulo 5 se han mostrado las principales medidas (de precisión y calibración) de los modelos vistos a lo largo del proyecto. A continuación se ampliará el estudio de la calibración, introduciendo las dos vistas en este apéndice (BRIER y NLL), complementando la evaluación llevada a cabo.

B.3.1. ResNet18

Nº	BRIER	NLL
1	0.0154	0.5679
2	0.0133	0.4940
3	0.0152	0.6051
4	0.0133	0.4731
5	0.0151	0.5697
ens	0.0099	0.2563

(a) Resultados de la Resnet18 en CIFAR-10 **antes** de aplicar **Temp. Scaling**

Nº	BRIER	NLL
1	0.0047	2.3120
2	0.0047	2.3312
3	0.0046	2.3146
4	0.0048	2.3459
5	0.0048	2.3630
ens	0.0034	1.2920

(b) Resultados de la Resnet18 en CIFAR-100 **antes** de aplicar **Temp. Scaling**

Tabla B.1: Tablas que muestran los NLL y BRIER obtenidos para la ResNet18 sobre CIFAR-10 (B.1(a)) y CIFAR-100 (B.1(b)) previos a la aplicación de la técnica de calibración *Temperature Scaling*

Como puede observarse, tanto el BRIER como NLL son medidas bastante más pequeñas que el ECE y MCE, por lo que su mejora tras aplicar el método de calibración será menor. Puede verse que el comportamiento visto en el capítulo 5 se mantiene. El *ensemble* mejora los valores de los modelos individuales. El *Temp. Scaling* también mejora ambas métricas. Sin embargo nuevamente aplicar el *Temp. Scaling* al *ensemble* empeora las medidas de calibración, siguiendo el comportamiento

Nº	BRIER	NLL
1	0.0133	0.2787
2	0.0115	0.2422
3	0.0130	0.2741
4	0.0116	0.2435
5	0.0129	0.2666
ens	0.0104	0.3246

(a) Resultados de la Resnet18 en CIFAR-10 después de aplicar **Temp. Scaling**

Nº	BRIER	NLL
1	0.0038	1.0238
2	0.0038	1.0247
3	0.0037	1.0197
4	0.0038	1.0337
5	0.0038	1.0369
ens	0.0040	1.4051

(b) Resultados de la Resnet18 en CIFAR-100 después de aplicar **Temp. Scaling****Tabla B.2:** Tablas con los valores del BRIER y NLL obtenidos para la ResNet18 sobre CIFAR-10 (B.2(a)) y CIFAR-100 (B.2(b)) después de aplicar la técnica de calibración *Temperature Scaling*

observado para ECE y MCE.

B.3.2. ResNet50

Nº	BRIER	NLL
1	0.0191	0.6435
2	0.0163	0.46100
3	0.0158	0.5321
4	0.0167	0.5527
5	0.0158	0.5260
ens	0.0111	0.2659

(a) Resultados de la Resnet50 en CIFAR-10 **antes** de aplicar **Temp. Scaling**

Nº	BRIER	NLL
1	0.0048	2.4010
2	0.0051	2.4492
3	0.0056	2.8751
4	0.0046	2.2604
5	0.0051	2.4833
ens	0.0035	1.3193

(b) Resultados de la Resnet50 en CIFAR-100 **antes** de aplicar **Temp. Scaling****Tabla B.3:** Tablas que muestran los NLL y BRIER obtenidos para la Resnet50 sobre CIFAR-10 (B.3(a)) y CIFAR-100 (B.3(b)) previos a la aplicación de la técnica de calibración *Temperature Scaling*

Como puede observarse, el comportamiento que muestran estas métricas acerca de la ResNet50 es muy similar al que se ha visto con ResNet18, al igual que con ECE y MCE. El *ensemble* mejora los valores, así como lo hace el *Temp. Scaling*. Sin embargo al aplicarlo sobre el *ensemble* los resultados empeoran.

Nº	BRIER	NLL	Nº	BRIER	NLL
1	0.0184	0.4090	1	0.0038	1.0270
2	0.0221	0.5249	2	0.0041	1.1163
3	0.0140	0.2917	3	0.0045	1.2764
4	0.0148	0.3105	4	0.0041	1.1352
5	0.0143	0.3064	5	0.0041	1.1139
ens	0.0157	0.3654	ens	0.0047	1.2002

(a) Resultados de la Resnet50 en CIFAR-10 después de aplicar **Temp. Scaling**(b) Resultados de la Resnet50 en CIFAR-100 después de aplicar **Temp. Scaling****Tabla B.4:** Tablas con los valores del BRIER y NLL obtenidos para la Resnet50 sobre CIFAR-10 (B.4(a)) y CIFAR-100 (B.4(b)) después de aplicar la técnica de calibración *Temperature Scaling*

B.3.3. DenseNet121

Nº	BRIER	NLL	Nº	BRIER	NLL
1	0.0103	0.4163	1	0.0044	2.1166
2	0.0101	0.4162	2	0.0043	2.0608
3	0.0105	0.4152	3	0.0044	2.1036
4	0.0104	0.4258	4	0.0045	2.1520
5	0.0107	0.4446	5	0.0044	2.1069
ens	0.0072	0.2063	ens	0.0031	1.1393

(a) Resultados de la Densenet121 en CIFAR-10 antes de aplicar **Temp. Scaling**(b) Resultados de la Densenet121 en CIFAR-100 antes de aplicar **Temp. Scaling****Tabla B.5:** Tablas que muestran los NLL y BRIER obtenidos para la Densenet121 sobre CIFAR-10 (B.5(a)) y CIFAR-100 (B.5(b)) previos a la aplicación de la técnica de calibración *Temperature Scaling*

En DenseNet121 el comportamiento que muestran estas métricas sigue el patrón de los valores mostrados en el capítulo 5. Nuevamente se ve la efectividad del *Temp. Scaling* y del *ensemble*, pero no de ambos combinados, ya que en este caso los resultados se vuelven peores.

Nº	BRIER	NLL
1	0.0087	0.1812
2	0.0088	0.1855
3	0.0088	0.1776
4	0.0087	0.1801
5	0.0095	0.1930
ens	0.0083	0.2298

(a) Resultados de la Densenet121 en CIFAR-10 **después** de aplicar **Temp. Scaling**

Nº	BRIER	NLL
1	0.0035	0.9567
2	0.0035	0.9476
3	0.0035	0.9330
4	0.0036	0.0947
5	0.0036	0.9455
ens	0.0038	1.2351

(b) Resultados de la Densenet121 en CIFAR-100 **después** de aplicar **Temp. Scaling**

Tabla B.6: Tablas con los valores BRIER y NLL obtenidos para la Densenet121 sobre CIFAR-10 (B.6(a)) y CIFAR-100 (B.6(b)) después de aplicar la técnica de calibración *Temperature Scaling*

B.3.4. EfficientNetB0

Nº	BRIER	NLL
1	0.0132	0.4894
2	0.0162	0.5775
3	0.0149	0.5289
4	0.0145	0.5374
5	0.0146	0.5233
ens	0.0101	0.2206

(a) Resultados de la EfficientNetB0 en CIFAR-10 **antes** de aplicar **Temp. Scaling**

Nº	BRIER	NLL
1	0.0042	1.1566
2	0.0041	1.1306
3	0.0042	1.1469
4	0.0042	1.1593
5	0.0041	1.1349
ens	0.0033	0.8411

(b) Resultados de la EfficientNetB0 en CIFAR-100 **antes** de aplicar **Temp. Scaling**

Tabla B.7: Tablas que muestran los NLL y BRIER obtenidos para la EfficientNetB0 sobre CIFAR-10 (B.7(a)) y CIFAR-100 (B.7(b)) previos a la aplicación de la técnica de calibración *Temperature Scaling*

Los valores BRIER y NLL obtenidos para la EfficientNetB0 no hacen más que reafirmar que el eficiente escalado otorga una mejor calibración base que al resto de modelos. Esto puede verse claramente en la tabla B.8(b), donde el NLL sin calibrar ronda el valor 1.1 mientras que para el resto de redes vistas siempre supera 2.

El comportamiento de esta red por lo demás es similar al de resto: mejora la calibración aplicando el *Temp. Scaling*, así como formando el *ensemble*, pero no combinándolos.

Nº	BRIER	NLL	Nº	BRIER	NLL
1	0.0116	0.2511	1	0.0041	1.0794
2	0.0137	0.2879	2	0.0040	1.0610
3	0.0125	0.2634	3	0.0040	1.0684
4	0.0123	0.2601	4	0.0040	1.0737
5	0.0130	0.2654	5	0.0040	1.0573
ens	0.0131	0.2806	ens	0.0041	1.0889

(a) Resultados de la EfficientNetB0 en CIFAR-10 **después** de aplicar **Temp. Scaling**

(b) Resultados de la EfficientNetB0 en CIFAR-100 **después** de aplicar **Temp. Scaling**

Tabla B.8: Tablas que muestran los BRIER y NLL obtenidos para la EfficientNetB0 sobre CIFAR-10 (B.8(a)) y CIFAR-100 (B.8(b)) después de aplicar la técnica de calibración *Temperature Scaling*

B.3.5. UpaNet

Nº	BRIER	NLL	Nº	BRIER	NLL
1	0.0080	0.2099	1	0.0034	0.9485
2	0.0078	0.1980	2	0.0034	0.9548
3	0.0082	0.2053	3	0.0034	0.9595
4	0.0077	0.1908	4	0.0034	0.9486
5	0.0074	0.1756	5	0.0035	0.9733
ens	0.0060	0.1321	ens	0.0026	0.6834

(a) Resultados de la UpaNet en CIFAR-10 **antes** de aplicar **Temp. Scaling**

(b) Resultados de la UpaNet en CIFAR-100 **antes** de aplicar **Temp. Scaling**

Tabla B.9: Tablas que muestran los NLL y BRIER obtenidos para la UpaNet sobre CIFAR-10 (B.9(a)) y CIFAR-100 (B.9(b)) previos a la aplicación de la técnica de calibración *Temperature Scaling*

Nº	BRIER	NLL	Nº	BRIER	NLL
1	0.0076	0.1753	1	0.0033	0.8809
2	0.0073	0.1660	2	0.0033	0.8940
3	0.0076	0.1701	3	0.0033	0.9486
4	0.0072	0.1627	4	0.0033	0.8869
5	0.0070	0.1559	5	0.0033	0.9055
ens	0.0087	0.2742	ens	0.0028	0.7266

(a) Resultados de la UpaNet en CIFAR-10 después de aplicar **Temp. Scaling**(b) Resultados de la UpaNet en CIFAR-100 después de aplicar **Temp. Scaling****Tabla B.10:** Tablas que muestran los NLL y BRIER obtenidos para la UpaNet sobre CIFAR-10 (B.9(a)) y CIFAR-100 (B.9(b)) después de aplicar la técnica de calibración *Temperature Scaling*

Si bien antes se ha dicho que EfficientNetB0 presentaba los mejores valores gracias a su óptimo escalado, UpaNet mejora estos registros aún más, no haciendo más que confirmar que se trata de la red más precisa y fiable. Pese a ser la red más actual y más completa, los problemas de aplicar al ensemble el *Temp. Scaling* no desaparecen.

