Trabajo Final de Máster:

Paralelización del entrenamiento y compresión de redes neuronales convolucionales para la detección de enfermedades de tórax

Andrea Navarro Ruiz

Tutores: Manuel Francisco Dolz Zaragozá

María Isabel Castillo Catalán

Máster en Sistemas Inteligentes

Especialidad en Interacción Avanzada y Gestión del Conocimiento

Abstract

Optimization methods applied on convolutional neural networks can report multiple benefits in their training and inference stages. Specifically, using data-parallelism schemes on multi-GPU platforms allows decreasing the training time. Similarly, the use of compression techniques, such as pruning or quantization, permits minimizing the total number of parameters or the use of reduced precisions, which imply a reduction in the training stage at the expense of minimal performance losses.

In this work, data parallelism, pruning, and quantization techniques are leveraged, tuned, and evaluated on a set of pre-trained convolutional neural networks able to diagnose common diseases on chest X-rays. The use of these techniques on these models has demonstrated that data-parallel schemes using platforms with multiple GPUs can effectively reduce the training times provided that the batch size is correctly selected.

Similarly, pruning non-significant connections among neurons at training time can lead to a considerable reduction in the number of operations performed and model trainable parameters with negligible accuracy loss. On the other hand, quantification-based techniques, such as quantification-aware training, permit an even lower memory usage and training times compared to pruning-based approaches; however, their use may carry negative effects on the classification results.

Índice

Capítulo 1

Introducción

En este primer capítulo se contextualiza el problema de la interpretación de las radiografías y las motivaciones para utilizar un sistema automatizado. También se exponen los objetivos de este trabajo haciendo mención a la literatura de otros estudios previos relacionados.

1.1. Contexto y motivación

Las radiografías tienen un papel clave en la detección de distintas patologías y en los servicios de urgencias, los cuales son la principal vía de acceso al Servicio Nacional de Salud de España [1]. Las pruebas radiológicas son el primer estudio que se les solicita a los pacientes: están indicadas para distintas urgencias (código ictus, politraumatismo, sepsis, etc.), permiten tomar mejores decisiones e incluso es una buena práctica que permite reducir la prescripción innecesaria de antibióticos a más de un tercio de los pacientes [2].

En la prueba radiológica se realiza la emisión de radiación en forma de ondas electromagnéticas. Debido a que los distintos tejidos absorben una cantidad diferente de radiación, se obtiene una imagen donde se muestra el interior del cuerpo en escala de grises [3]. Sin embargo, en ocasiones estas radiografías no son fáciles de interpretar. Por ejemplo, en el caso de las neumonías, pueden desarrollarse en comorbilidad o mimetizarse con otras anomalías benignas, lo que complica su detección. Debido a esto es frecuente encontrar disparidades en los informes realizados por distintos radiólogos [4].

La alta presión sobre los hospitales, y en especial sobre los servicios de urgencias, generan un gran volumen de radiografías que requieren ser interpretadas por personal técnico especializado [1], además, muchas de ellas requieren una rápida respuesta y en cualquier momento horario. A menudo, el personal del servicio de radiodiagnóstico no tiene capacidad suficiente para analizar todo el volumen de pruebas que recibe, bien debido a limitaciones de la plantilla o a su propia organización [5]. Por otra parte, las radiografías compiten con otras pruebas más sofisticadas que requieren más tiempo de análisis por parte del personal especializado (p.ej. ecografías, tomografías computarizadas o resonancias magnéticas). Todo esto lleva a que parte de las radiografías no sean informadas a tiempo,

ocasionando retrasos, o que los informes puedan incluir errores (se estima que entre el 3-5% de los informes diarios los contienen, debido a causas humanas o por saturación del personal) [6].

Todos estos comentarios justifican la necesidad de tener un sistema automático de radiodiagnóstico fiable y eficaz que permita descongestionar los hospitales y, en especial, sus servicios de urgencias. Una vía de abordaje es el uso de técnicas de computación de Inteligencia Artificial: esta cuestión bien puede tratarse como un problema de clasificación de imágenes, en el que un modelo o clasificador sea capaz de identificar si una radiografía es patológica o no, e incluso identificar el tipo de patología si la contiene.

Así pues, la solución a este problema se puede obtener mediante el diseño y desarrollo de un sistema de análisis, clasificación y detección de radiografías para procesos asistenciales urgentes basado en técnicas computacionales. Este sistema debe ser lo suficientemente avanzado, como para detectar no solo una patología, sino todas las que se observen en la radiografía en caso de comorbilidad, lo que supone un reto complejo.

Para abordar esta solución, en este trabajo se propone el uso de modelos de redes neuronales profundas (en adelante DNNs del acrónimo inglés de *Deep Neural Networks*). Aunque existan distintas interpretaciones, comúnmente, cuando se habla de DNNs se hace referencia a aquellas redes neuronales que emplean, al menos, dos capas ocultas de procesamiento no lineal para extraer características y transformar variables. Si bien las redes neuronales como tal existen desde la década de los 60 [7] no ha sido hasta la actualidad que han resurgido, debido principalmente a:

- La gran cantidad de datos disponibles (era del *big data*) que son necesarios para entrenar apropiadamente las DNNs y que ofrezcan resultados robustos.
- El aumento de la capacidad de cómputo de los sistemas actuales, que permite reducir los tiempos de entrenamiento de las DNNs a intervalos factibles.
- Los avances en programación y algoritmia, así como las aportaciones en código abierto que han facilitado la investigación.

Todo ello ha llevado a que las DNNs se hayan establecido hoy en día como una tecnología clave para la resolución de problemas de ciencia e ingeniería en múltiples áreas de conocimiento.

Desafortunadamente las arquitecturas de computadores basadas en procesadores multinúcleo no son lo suficientemente potentes para satisfacer las altas demandas computacionales de las DNNs, por lo que es necesario la utilización de otras arquitecturas. Particularmente en la etapa de entrenamiento, se requiere manejar grandes volúmenes de datos que pueden ser procesados mediante algoritmos paralelos eficientes para plataformas de memoria distribuida con aceleradores. En esta etapa también suele ser habitual aplicar técnicas de optimización como la poda de pesos o cuantización de los modelos, que permiten reducir la cantidad de cálculos a efectuar durante el proceso de entrenamiento.

Por esos motivos, en este trabajo se plantea la clasificación de patologías por imagen mediante modelos de redes neuronales empleando paralelismo y técnicas de optimización que permitan simplificar su complejidad.

1.2. Objetivos

El objetivo de la investigación es el desarrollo de una herramienta de apoyo al radiodiagnóstico mediante el uso de redes neuronales, entrenadas mediante la combinación de herramientas de computación de altas prestaciones y de procesamiento de datos masivos (*big data*). Más específicamente, este objetivo global se puede desglosar como sigue:

- Análisis y puesta en marcha de modelos de redes neuronales convolucionales para el análisis de radiografías de tórax.
- Evaluación y ajuste de hiperparámetros de modelos de red neuronal convolucionales alternativos.
- Aplicación y evaluación del entrenamiento paralelo de datos sobre plataformas de altas prestaciones equipadas con aceleradores (GPUs).
- Aplicación y evaluación de técnicas de optimización basadas en la compresión de los modelos de red: poda y cuantización.

Así, la motivación para este trabajo es doble: por un lado, contribuye al campo de estudio de la imagen médica mediante la aplicación de técnicas de optimización sobre un modelo incluido en el estado del arte para la detección de patologías a partir de radiografías, que no ha sido optimizado previamente. Por otra parte, es la vía para asentar y probar los conocimientos adquiridos a lo largo del máster de Sistemas Inteligentes sobre aprendizaje automático profundo, reconocimiento de patrones y la programación sobre arquitecturas paralelas.

1.3. Estado del arte

En los últimos años, aparecen en la literatura referencias a algoritmos computacionales dentro del marco del aprendizaje automático para su aplicación en el ámbito de la medicina. Así, existen diversos estudios que, al igual que este trabajo, se apoyan en redes neuronales para la clasificación, evaluación o detección de patologías. Algunos ejemplos de ello son:

- Modelo matemático predictivo de la mortalidad en la neumonía adquirida en la comunidad, realizado en el Hospital Militar Dr. Carlos J. Finlay en La Habana, Cuba [8].
- Clasificación del grado de astrocitoma cerebral infantil a partir de la morfología matemática en las imágenes médicas (tomografía computarizada, resonancia magnética, tomografía por emisión de positrones o magneto encefalografía) [9].
- Modelos estadísticos aplicados al diagnóstico precoz del Alzheimer y a otras enfermedades neurológicas mediante la detección de patrones de perfusión cerebral en tomografías [10].

Además, existen otros trabajos que conviene considerar, ya que, al igual que este estudio, trabajan con redes neuronales convolucionales (en adelante CNN, acrónimo del inglés: *Convolutional Neural Network*) sobre radiografías torácicas, como:

- Radiodiagnóstico de neumonías mediante CNN en el Centro Médico para Mujeres y Niños de Guangzhou: este clasificador divide los casos en tres clases distintas que permite diferenciar una neumonía vírica, de una de origen bacteriano o la ausencia de neumonía [11].
- El modelo CheXnet donde diferentes investigadores de la Universidad de Stanford, pertenecientes a los departamentos de Medicina, Ciencias Computacionales y Radiografía proponen el algoritmo para la detección de 14 patologías a partir de radiografías de tórax frontales del National Institute of Health, también usando CNN [12].

Pese a que las redes neuronales han sido cuestionadas a menudo por la cantidad de recursos computacionales que requieren [11], también muestran una alta capacidad de aprendizaje frente a problemas complejos. Afortunadamente, el uso de redes neuronales convolucionales supone una ventaja importante en la reducción de memoria frente a las completamente conexas [13], siendo, además, las que se emplean en este estudio. Otra ventaja de este tipo de redes es que permiten omitir el paso de la extracción de características de las imágenes al ser la propia red convolucional la que realiza ese rol.

Por otro lado, ninguno de los trabajos aquí expuestos menciona que se hayan empleado técnicas de compresión durante el entrenamiento, ni tampoco el uso del paradigma de paralelismo de datos para el entrenamiento de las redes en plataformas multi-GPU.

En el caso de [12], se propone CheXnet, un modelo CNN propuesto inicialmente para la detección de neumonías y que obtuvo un éxito superior al de radiólogos experimentados, que posteriormente ha sido ampliado para incluir otras 13 patologías. Dado sus buenos resultados de predicción y la inexistencia de tratamientos de optimización previos, se ha seleccionado este algoritmo como punto de partida en este trabajo.

Capítulo 2

Conceptos previos

En este capítulo se revisan los conceptos principales sobre los que desarrollará el resto del trabajo: cómo son las redes neuronales de tipo convolucional, en qué consiste su proceso de aprendizaje, sus aplicaciones en el campo de la imagen médica y algunas de las variantes de las redes neuronales modernas: ResNet y DenseNet.

2.1. Redes Neuronales Convolucionales

Las redes neuronales son modelos computacionales pertenecientes al campo del Aprendizaje Automático Supervisado, que es, a su vez, una rama de la Inteligencia Artificial. Su diseño se inspira en el funcionamiento de las conexiones cerebrales y su entrenamiento se realiza a partir de datos etiquetados, de los cuales son capaces de extraer una serie de patrones y aprender a identificarlos, comparando sus predicciones frente a las etiquetas reales, así como ser capaz de retroalimentarse a partir de los resultados.

Cuando se aplican sobre imágenes médicas, es posible entrenar las redes neuronales obviando su geometría y extrayendo la intensidad de color de los píxeles como una lista de características. Sin embargo, con este método se altera la estructura espacial y se omite la información derivada de la cercanía entre píxeles.

Surge entonces una alternativa más eficiente para la gestión de imágenes: las redes neuronales convolucionales. Cuando una imagen se introduce en una CNN, se almacena de forma tensorial: bidimensional en el caso de escala de grises, tridimensional si es en color (RGB). Sobre este conjunto de datos inicial es posible aplicar una serie de capas. Si bien el término “capa” puede tener distintas interpretaciones, en este ámbito se define como el conjunto de funciones de transformación que aplican diferentes filtros y activaciones sobre los elementos de entrada, y devuelven una salida en forma de mapas de características.

Estos filtros recorren los tensores de entrada y consideran tanto el valor de la celda, o píxel, como el de una determinada área de proximidad. Esto permite, por ejemplo, detectar bordes y contornos, o entrenar considerando la invarianza espacial.

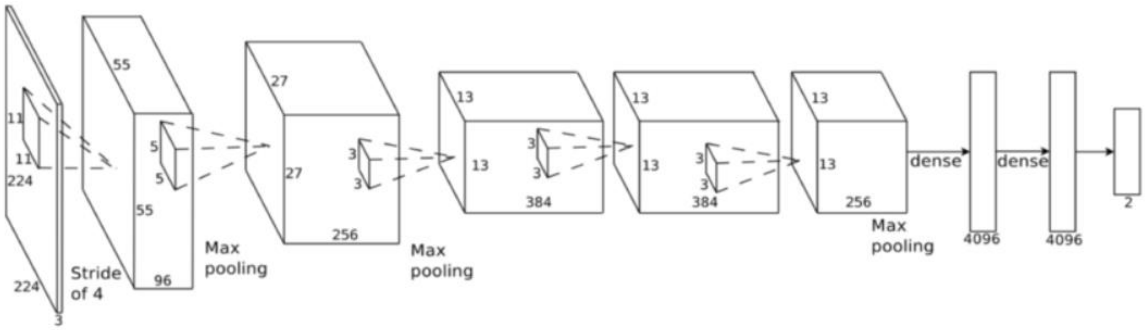

Figura 1. Arquitectura AlexNet

En la Figura 1 se muestra la arquitectura de AlexNet [14] formada por 8 capas: 5 convolucionales, 2 densas y una de salida; empleando 60 millones de parámetros. AlexNet es una de las CNNs pioneras, no porque fuera de las primeras redes en diseñarse, ya que existían como tal desde la década de los 90, pero sí fue la que mostró resultados sorprendentemente buenos, al ganar el reto de reconocimiento visual a gran escala de ImageNet en 2012, y popularizó el uso de las CNNs en visión por computador.

El procesamiento de las capas de la red neuronal puede realizarse de forma eficiente en CPUs multinúcleo o en aceleradores GPU, ya que la mayor parte de su procesamiento requiere el cómputo de operaciones de álgebra lineal. Entre estas operaciones cabe destacar, la multiplicación de matrices, que aparece paralelizada y altamente optimizada en bibliotecas de BLAS (*Basic Linear Algebra Subprograms*), p.ej. MKL para CPUs multinúcleo de Intel o cuBLAS/cuDNN para aceleradores GPU de NVIDIA [15]. Además, las CNNs, como se ha mencionado anteriormente, suponen una reducción de memoria frente a las completamente conexas, ya que poseen una estructura que permite reducir el número de parámetros a definir, disminuyendo así el tamaño del modelo y el tamaño de los gradientes que deben acumularse durante un escenario de entrenamiento distribuido. [13]. Por todo ello, las CNNs se utilizan en distintas áreas como la clasificación de series de tiempo o señales de audio, y más especialmente en la clasificación de imágenes. De hecho, actualmente, son el método más empleado en el campo de la visión por ordenador.

La gran capacidad de aprendizaje de las CNNs, así como su capacidad de resolución de problemas complejos, las expone al riesgo de sobreajuste o *overfitting*. Cuando esto sucede, la red deja de aprender de un modo generalizado y pasa a concentrarse en los pequeños detalles que forman parte del ruido de la muestra de entrenamiento, haciendo que incremente su capacidad de predicción exageradamente en ese conjunto puntual, pero le resta capacidad de predecir para otras muestras que no haya visto previamente.

Existen distintas vías para evitar este fenómeno, algunas de las cuales se aplican en este trabajo, tales como criterios de parada, regularización de los datos, simplificación del modelo, MaxPooling y técnicas de *Data Augmentation.*

Las técnicas de *Data Augmentation* son de las opciones más frecuentes contra el sobreajuste, ya que permiten mejorar, a la vez, la precisión del modelo. Estas técnicas consisten en entrenar, aplicando pequeñas modificaciones sobre las imágenes originales (deslizamientos, rotaciones, ruido, etc.), de modo que se amplía la variedad del conjunto de entrenamiento, y se le facilita generalización al algoritmo.

Otro de los inconvenientes de las redes neuronales es la gran cantidad de tiempo y recursos necesarios para su entrenamiento. Esto hace que cada vez sea más común la utilización de la transferencia de aprendizaje o *transfer learning*, técnica en la que se reciclan modelos previamente entrenados (o gran parte de ellos) para una aplicación similar, utilizando los parámetros que constituyen el peso entre conexiones como punto de partida para adaptar el modelo al nuevo problema. El modelo resultante puede requerir ajustes adicionales en las capas finales y su reentrenamiento, pero el tiempo de convergencia será notablemente inferior, ahorrando así recursos computacionales y temporales.

2.2. Entrenamiento de redes neuronales

El método en el que aprenden los modelos durante el entrenamiento se conoce como *backpropagation* o retropropagación. Seguidamente se describe en detalle este proceso, en el que se asume que el conjunto de todas las muestras disponible se define como un ciclo, o época, que se divide en grupos de muestras de menor tamaño, a las que se denomina batch.

El entrenamiento se inicializa con unos pesos aleatorios o preentrenados $W = (\omega_{1,}\, \omega_2, \omega_3 \ldots)$ asignados en las diferentes capas de la red neuronal. Seguidamente, se le proporciona a la red un batch de muestras etiquetadas que la recorren a través de todas sus capas, dando lugar a un conjunto de salidas; este proceso se conoce como *forward pass* o propagación hacia adelante (Figura 2a). Con posterioridad, se utiliza la función pérdida elegida para calcular el error de salida (δ) asociado a ese conjunto de muestras frente a las etiquetas reales, y ese error se retropropaga en la red, recorriéndola en sentido inverso (Figura 2b), y calculando en cada capa de la red neuronal tanto los gradientes de los pesos como los sesgos asociados.

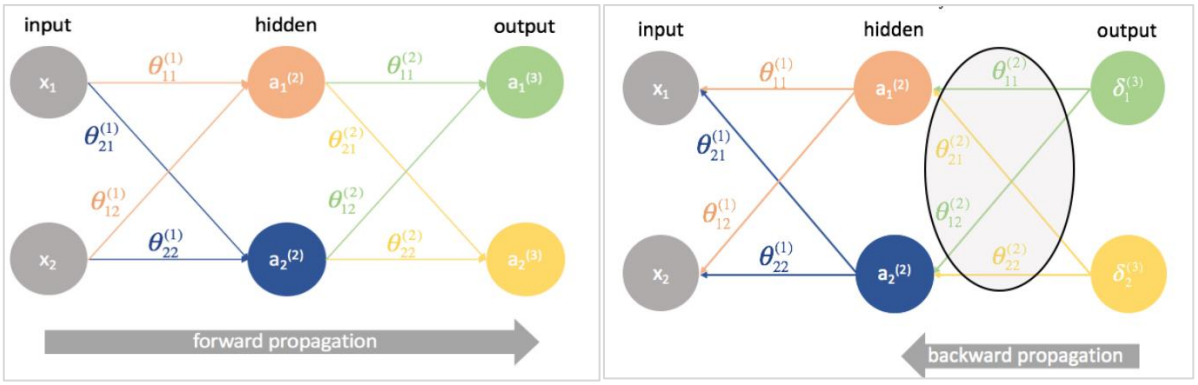

Figura 2. (a) En la parte superior se muestra un ejemplo de propagación hacia adelante, siendo x las muestras, a los resultados tras la activación y θ los pesos. (b) En la parte inferior se muestra como el error δ se retropropaga a través de la red.

Existen distintos métodos para realizar la actualización de los pesos; uno de los más comunes, cuando la pérdida es una función diferenciable, es el descenso estocástico del gradiente (SGD). Para ello, se deriva la función pérdida (L), que permite calcular el gradiente respecto a cada uno de los pesos, obteniendo la dirección hacía la que deben moverse los pesos para converger a un mínimo local de pérdida. Así, el incremento de los parámetros en cada actualización se define como:

$$\Delta\omega = -\gamma \frac{\partial L(\omega)}{\partial \omega}$$

siendo ω un parámetro de peso, componente de la función de pérdida L.

Por su parte, el hiperparámetro γ se conoce como el factor de aprendizaje o *learning rate* y marca la velocidad de aprendizaje. Algunos algoritmos pueden realizar cambios en este hiperparámetro a medida que avanza el proceso de entrenamiento para mejorar la actualización de los pesos.

Una vez se han actualizado los pesos, se repite el proceso con el siguiente batch de muestras. La razón por la que los pesos se actualizan a cada batch y no individualmente no es arbitraria: el conjunto de muestras mitiga la varianza en la actualización permitiendo una convergencia más estable. Es por ello que, el tamaño del batch también es un hiperparámetro relevante a la hora de confeccionar un modelo.

2.3. Redes aplicadas a la imagen médica: CheXnet

En el área de la automatización de diagnóstico por imagen basado en radiografías, existen distintos trabajos como Wang, 2017 [16], Yao, 2017 [17] y CheXnet [34], también de 2017. Este último destaca por sus altos resultados predictivos frente a los anteriores en todas las patologías a estudio, que lo convirtió en una referencia en esta área de investigación.

El modelo CheXnet, es un modelo de clasificación de 14 patologías:

- Atelectasia
- Cardiomegalia
- Derrame
- Infiltración
- Hernia
- Masa
- Nódulo
- Neumonía
- Neumotórax
- Fibrosis
- Consolidación
- Edema
- Enfisema
- Engrosamiento plural

Está construido sobre una arquitectura Densenet121, una variedad moderna de CNN que contiene conexiones densas hacia sus capas posteriores, por las cuales recibe el nombre.

Para su aprendizaje se empleó el conjunto de datos del *National Institute of Health* (NIH) que contiene 112.120 radiografías frontales de dimensiones 1024x1024, todas ellas apropiadamente anonimizadas para preservar la intimidad de los pacientes [18]. Dado que los informes médicos tampoco son públicos, estos se trataron mediante un algoritmo de procesamiento de lenguaje natural que permitieron extraer las etiquetas de las 14 patologías mencionadas, obteniendo una precisión superior al 90%. Observando el conjunto de etiquetas para cada clase, se aprecia un desbalance significativo, ya que algunas patologías, como la hernia, tienen una frecuencia muy baja (227 casos) frente a otras mucho más recurrentes, como la atelectasia (11.561 muestras) o la infiltración (19.898).

Clasificación binaria

CheXnet fue diseñado inicialmente para la detección únicamente de neumonías, es decir se desarrolló como un problema de clasificación binaria. Para acelerar el proceso de aprendizaje se partió de unos pesos preentrenados para el conjunto ImageNet. Sin embargo, esto requirió ajustar la arquitectura de la red al problema binario, reemplazando la última capa de salida (que previamente era *softmax)* por una densamente conexa de salida única con función sigmoide.

El motivo del cambio en el tipo de capa se justifica por el tipo de salida requerida. La sigmoide es una función $f(x)\colon \mathbb{R}^n \rightarrow \mathbb{R}^1$ escalar con forma de función escalón suavizada, lo que permite mantener la continuidad durante la transición a partir de la cual la función se satura. Por su parte, la función *softmax* es una función vectorial $g(x)\colon \mathbb{R}^n \rightarrow \mathbb{R}^m \ : m > 1$ que transforma un vector dado en un vector distribución de probabilidad cuyo máximo valor es realizado mientras que el resto de valores se comprime y homogeniza. Dado que el reto de ImageNet era de multiclasificación con salida vectorial, la alternativa correcta era utilizar *softmax*, pero en este caso se trataba de un problema de clasificación binario, por lo que la salida de la red debía ser un escalar binario, así pues, se requirió el uso de una función de activación de tipo sigmoide.

Una vez dispuesta la arquitectura de la red, fue reentrenada con el conjunto de radiografías del NIH. El total de radiografías se dividió en tres subconjuntos: entrenamiento, validación y test. Los dos primeros participaron en el entrenamiento, mientras que el tercero permaneció oculto y solo intervino en la fase final de evaluación del modelo.

De los dos primeros conjuntos, como en un proceso de aprendizaje supervisado habitual, el de entrenamiento propiamente se usó para alimentar el algoritmo, como elemento de entrada, predicción de salida, comparación y obtención del gradiente de los pesos. Además, también se empleó el conjunto de validación para evaluar los progresos del algoritmo en cada época.

La totalidad de las imágenes o radiografías pasaron por un preprocesamiento de datos que consistieron en:

- **Reetiquetado**: Se obviaron las etiquetas correspondientes a otras patologías, y solo se consideraron patológicas si contenían "neumonía", considerando que era "sana" en caso contrario.
- **Partición**: Se aplicó la subdivisión de entrenamiento, validación y test, forzando a que las radiografías de un mismo paciente aparecieran siempre en un único conjunto.
- **Reescalado**: Se redimensionaron las imágenes de 1024x1024 a 224x224 píxeles.
- **Estandarización**: Proceso de tipificación estándar, empleando la media y desviación del conjunto ImageNet.
- ***Flipping***: Se aplicaron reflexiones horizontales sobre las imágenes de entrenamiento para ampliar la muestra.

Una vez terminado este procesamiento, se entrenó el modelo. De este modo, para cada imagen procesada X, el modelo emitió una etiqueta de salida dicotómica $y \in \{0,1\}$ indicando la presencia o ausencia de neumonía.

La función pérdida de tipo entropía cruzada binaria (*binary cross-entropy*) se definió del siguiente modo:

$$L(X, y) = -w_+ \cdot y \log p(Y = 1|X) - w_- \cdot (1 - y) \log p(Y = 0|X)$$

siendo $p(Y = i|X)$ la probabilidad de que dada la imagen X, la red asigne la etiqueta i, y sean $w_+ = |\mathrm{N}|/(|\mathrm{P}| + |\mathrm{N}|)$ y $w_- = |\mathrm{P}|/(|\mathrm{P}| + |\mathrm{N}|)$ las correcciones de pesos, tales que |P| y |N| definen, respectivamente, el total de casos positivos y negativos de neumonía en el conjunto de entrenamiento.

De modo que, iterando en la optimización de la función perdida para encontrar sus mínimos, se convergía a una selección de parámetros (pesos) que proporcionaban una clasificación con mayor volumen de aciertos, manteniendo, además, un equilibrio entre ambas clases.

Multiclasificador

Este modelo se pudo extender a las 14 patologías anteriormente mencionadas con una serie de cambios menores:

- Reemplazando la capa de salida por otra densamente conexa que contenga un vector binario de 14 componentes y función de activación *softmax*.
- Reetiquetando las imágenes para que considerasen todas las patologías.

- Reformulando la función de entropía cruzada para incluir una ponderación que equilibrase la importancia de todas las clases, definida como sigue:

  Sean c=1,2… 14 las distintas clases de patologías, se define la función pérdida tipo entropía cruzada binaria tal que:

$$L(X, y) = \sum_{c=1}^{14} [-y_c \cdot \log p(Y_c = 1|X) - y_c \cdot (1 - y_c) \log p(Y_c = 0|X)]$$

  donde y_c es la etiqueta correspondiente a la clase c para la imagen X, y $p(Y_c = i|X)$ es la probabilidad de que el modelo devuelva la etiqueta i para la clase c, cuando el elemento de entrada es la imagen X.

Pathology	Wang et al. (2017)	Yao et al. (2017)	CheXNet (ours)
Atelectasis	0.716	0.772	**0.8094**
Cardiomegaly	0.807	0.904	**0.9248**
Effusion	0.784	0.859	**0.8638**
Infiltration	0.609	0.695	**0.7345**
Mass	0.706	0.792	**0.8676**
Nodule	0.671	0.717	**0.7802**
Pneumonia	0.633	0.713	**0.7680**
Pneumothorax	0.806	0.841	**0.8887**
Consolidation	0.708	0.788	**0.7901**
Edema	0.835	0.882	**0.8878**
Emphysema	0.815	0.829	**0.9371**
Fibrosis	0.769	0.767	**0.8047**
Pleural Thickening	0.708	0.765	**0.8062**
Hernia	0.767	0.914	**0.9164**

Tabla 1. Resultados de AUROC del algoritmo CheXnet para 14 patologías frente a otros algoritmos.

Los resultados del modelo CheXnet para las 14 patologías, que como se ha mencionado con anterioridad se han convertido en valores de referencia, se muestran en la Tabla 1. En ella, se puede apreciar como los mejores índices de detección se obtuvieron para algunas patologías como la cardiomegalia, el enfisema o la hernia. Cabe destacar el caso de la hernia pues estos resultados se obtuvieron a pesar de su escasa representatividad en la muestra.

2.4. Modelos utilizados

Como se ha visto en los conceptos previos, las CNNs son modelos computacionales que consisten en la aplicación de diferentes capas (transformaciones) sobre los datos de entrada. En las versiones más modernas de las CNNs, la propuesta es elegir una serie de capas formando bloques que se aplican repetidamente. Entre ellas existe múltiples familias según su arquitectura y elección de hiperparámetros, lo que varía su rendimiento, tiempos de entrenamiento y adecuación a tipos de problemas. En este trabajo se emplearán dos modelos de CNNs modernos: ResNet y DenseNet

ResNet

Si partimos de la idea de que una CNN estándar es una progresión de capas apiladas en bloques desde el elemento de entrada hasta la etiqueta de salida secuencialmente conectadas, la particularidad de la ResNet es la inclusión de conexiones residuales.

Esta idea surge de una teoría matemática que sostiene que utilizar arquitecturas con funciones anidadas (cuyas imágenes matemáticas -rango de valores de f(x)- estén contenidas en la anterior) garantiza la convergencia monótona no decreciente a un óptimo.

Sin entrar en todo el fundamento matemático subyacente al funcionamiento de las redes neuronales, que es amplio y poco relevante para los objetivos de este trabajo, se puede afirmar que emplear funciones cuya imagen matemática sea cada vez más amplia, es decir, que el conjunto de valores de salida tenga un mayor número de términos, permite incrementar la expresividad de la red.

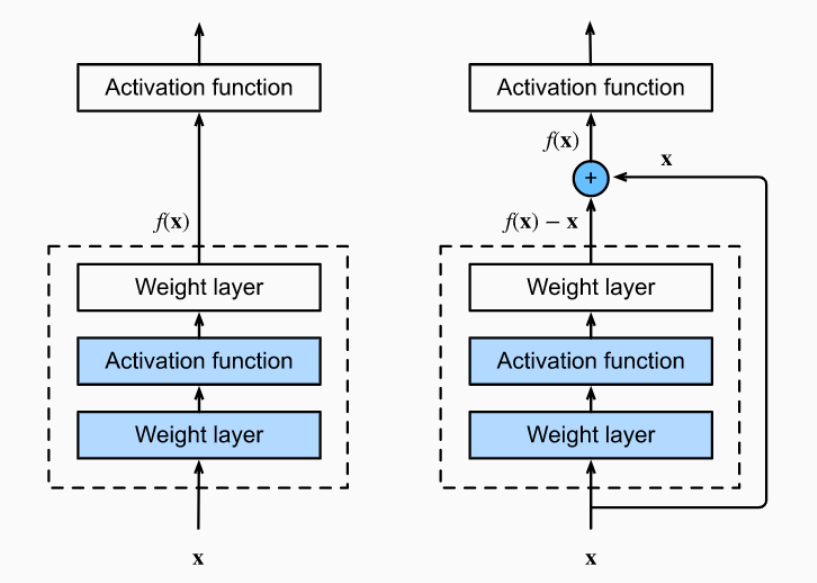

Figura 3. A la izquierda se muestra un bloque estándar, en comparación con la derecha donde se muestra un bloque residual (con conexión de acceso directo) [19].

Es por ello que se propone la aproximación a la función identidad, cuya imagen matemática es todo $\mathbb{R}$, agregando el elemento de entrada al resultado de cada bloque para hacer tender el comportamiento de $f(x) \simeq x$. Como se muestra en la Figura 3, esta conexión llamada residual o de acceso directo, enlaza el elemento de entrada del bloque con su salida. De este modo se le proporciona a la red una mejor adaptación al conjunto de entrenamiento, y que las entradas puedan propagarse hacia adelante más rápidamente.

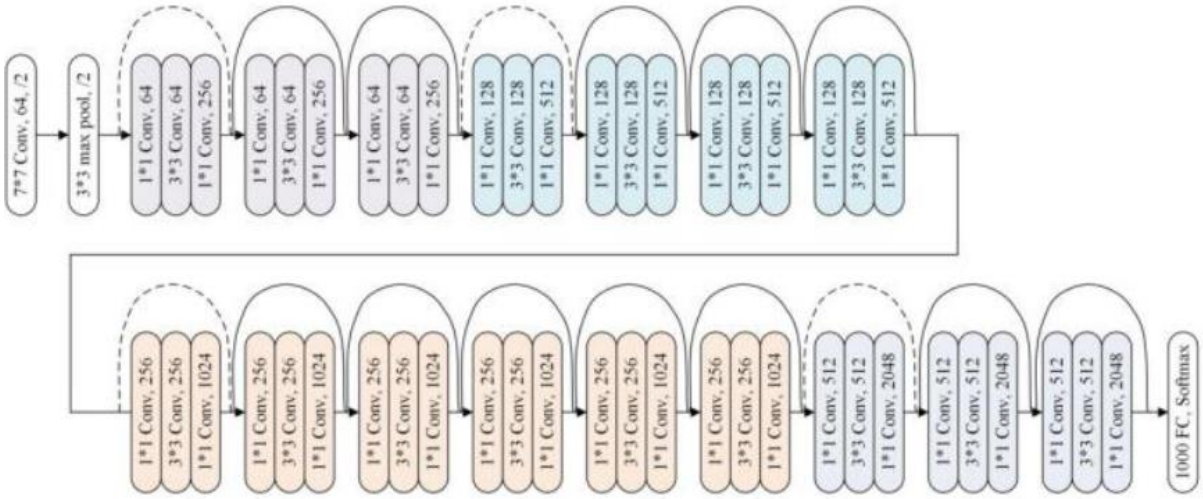

Figura 4. Arquitectura ResNet50 [21].

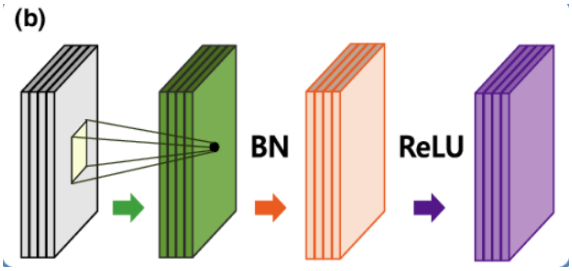

Figura 5. Arquitectura de una capa convolucional en la ResNet50.

En la Figura 4 se muestra un ejemplo de arquitectura ResNet: la ResNet50. Esta red se inicia con una capa convolucional, seguida de una capa *MaxPooling*, y luego se aplica de forma repetitiva bloques de 3 convoluciones hasta el *AvgPooling* final y una capa completamente conexa de salida.

Las capas convolucionales de la ResNet50 se vuelven más pequeñas y profundas a lo largo de la red, esto es, los mapas de características tienen más canales de salida, pero las dimensiones de filas y columnas se reducen. Como se observa en la Figura 5, cada capa está formada por una operación de convolución seguida por una de estandarización del batch (o *batch normalization*) que calcula la media y desviación típica de cada canal para todos los elementos de un batch, y con esos valores normaliza el propio canal, justo antes de aplicar una función de activación de tipo ReLU. La ReLU, o función de unidad linear rectificada, se define como $f(x) = max(0, x)$ y es una de las funciones de activación protagonistas ya que por su forma no anula el gradiente cuando está activada, a diferencia de la sigmoide que en el caso de que la entrada correspondiese a una región saturada produciría una reducción del gradiente con dificultades para el aprendizaje [22]

Por su parte, la técnica de *pooling* o agrupamiento, toma un valor representativo a partir de un conjunto de celdas, bien sea el promedio (*AvgPooling*) o de valor máximo (*MaxPooling*) reduciendo las dimensiones del canal. Esta operación fomenta la generalización del modelo, lo que también minimiza el riesgo de sobreajuste, y reduce el número de parámetros, ahorrando a su vez tiempo de entrenamiento.

DenseNet

Para definir la arquitectura DenseNet, igual que en el caso anterior, se parte de una CNN estándar. En este caso, las conexiones en lugar de ser secuenciales (cada capa está unida con la inmediatamente posterior) son densas, esto es, cada capa está unida con todas sus capas posteriores como se muestra en la Figura 6. Esto implica también que, para cada capa, todas las capas anteriores son elementos de entrada, dando lugar a funciones concatenadas:

$$x \rightarrow [x, f1(x), f2([x, f1(x)]), f3([x, f1(x), f2([x, f1(x)])]), \dots]$$

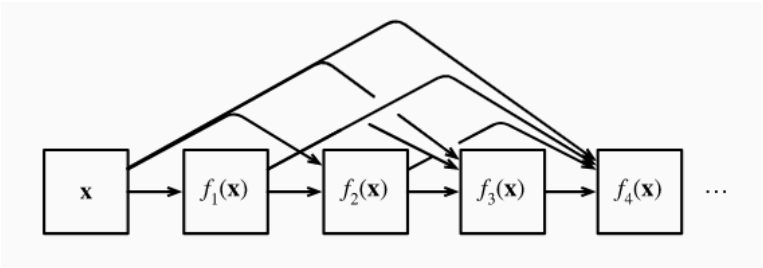

Figura 6. Conexiones densas en DenseNet [19].

La principal ventaja de esta estructura es que, enlazar todas las capas con la capa de salida, permite reentrenar las conexiones, mejorando el flujo de información en la retropropagación y facilitando así la optimización en redes profundas.

En la Figura 7 se muestra un caso genérico de esta arquitectura, donde se aprecian las conexiones densas hacia adelante. La DenseNet121, perteneciente a esta familia, se inicia con una convolución y *MaxPooling*, luego continua con 5 bloques formados por capas convolucionales con la estructura anteriormente mencionada (convolución, estandarización del batch y activación ReLU) en los que se aplica un *AvgPooling* al final de los bloques 2, 3, y 4, y concluye con una capa completamente conexa de salida.

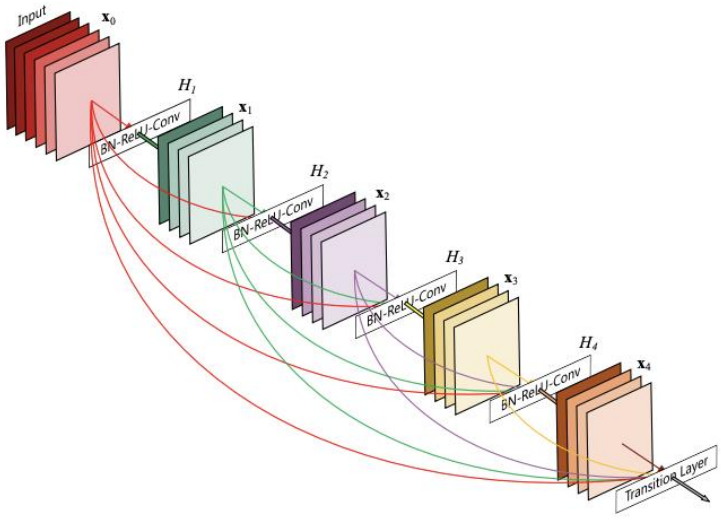

Figura 7. Arquitectura DenseNet [20] [20]

Capítulo 3

Optimización de redes neuronales convolucionales

En este capítulo se propone el uso de distintas técnicas de entrenamiento distribuido, que permiten reducir el tiempo de entrenamiento de las redes neuronales, y de compresión que permiten reducir el tamaño del modelo y, más tarde, el tiempo de inferencia.

3.1. Entrenamiento distribuido en GPUs utilizando paralelismo de datos

Un posible método para acelerar el tiempo de entrenamiento de una CNN es emplear el entrenamiento distribuido y el paralelismo de datos, distribuyendo los lotes de muestras o batchs entre más de un acelerador gráfico o GPU (*Graphic Processor Unit*). La utilización de más de una GPU para el procesamiento de estas muestras también resulta útil cuando por limitaciones del dispositivo, el batch de muestras no cabe en su memoria.

La forma de realizar el entrenamiento distribuido se puede explicar siguiendo el diagrama de la Figura 8. Inicialmente el total de batchs de muestras se divide entre un conjunto de GPUs, en las que se ha desplegado una réplica de la red, de modo que en todas ellas coinciden los pesos iniciales. Posteriormente, cada GPU efectúa los cálculos de propagación hacia adelante y hacía atrás para su conjunto de datos, hasta obtener los gradientes de los pesos y sesgos correspondientes. Finalmente, los gradientes calculados por cada una de las GPUs deben combinarse y promediarse para actualizar los pesos con el mismo criterio en cada una de ellas. Este proceso se repite para cada uno de los batchs del conjunto de datos, hasta alcanzar la convergencia del modelo.

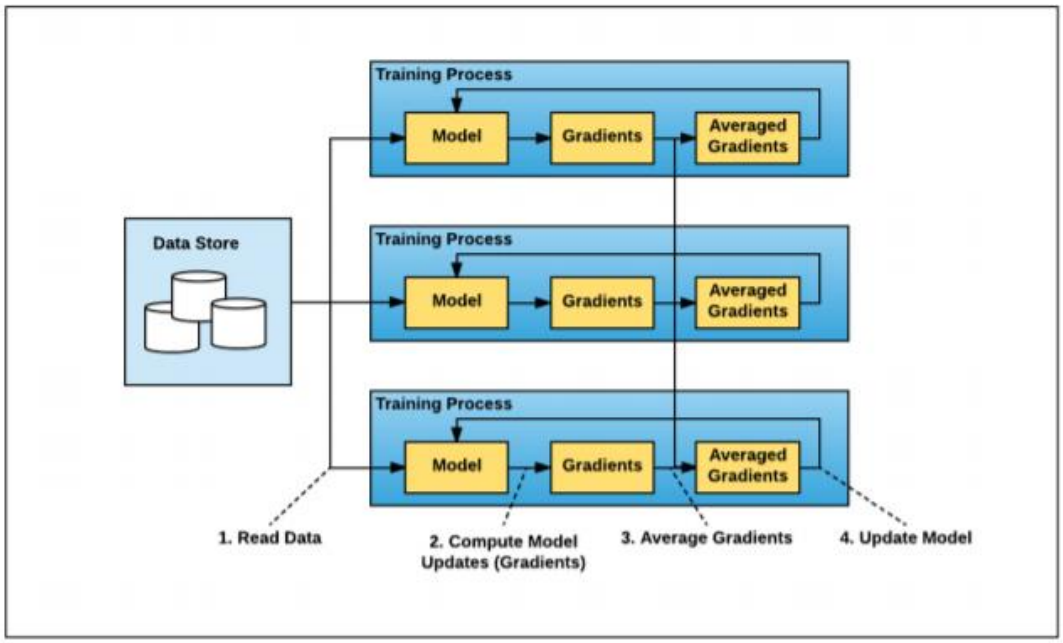

Figura 8. Diagrama de paralelismo de datos para el entrenamiento distribuido [23]

Para lleva a cabo la operación promedio de los gradientes se requiere la utilización de una operación AllReduce, que permite integrar los gradientes de las distintas GPUs y replicar el resultado en todas ellas. Esta operación puede suponer un cuello de botella debido a la transferencia de información entre las GPUs [24]. Por esta razón, se emplean bibliotecas de paso de mensajes (MPI) de propósito general y particularmente, de propósito específico de redes neuronales, tales como NVIDIA NCCL [35] o IBM DDL [36], optimizadas para la transferencia de grandes mensajes.

Este es el único paso crítico ya que el resto del proceso de entrenamiento distribuido puede llevarse a cabo en paralelo en las distintas GPUs, ya que las operaciones que se realizan sobre cada copia del modelo son independientes.

3.2. Técnicas de compresión

Los modelos de aprendizaje profundo, y en concreto los aquí propuestos, se construyen sobre arquitecturas que usan millones de parámetros, lo que supone trabajar con matrices de grandes dimensiones, dificultando la inferencia en dispositivos empotrados o móviles con memoria limitada. Es por ello, que la utilización de técnicas que reduzcan estos parámetros también reduce el tamaño de las matrices asociadas y, con ello, el coste computacional correspondiente.

Además, las técnicas de compresión permiten un uso más eficiente del consumo energético, ya que reducen los accesos a la memoria y mejoran los tiempos de inferencia En consecuencia estas técnicas permiten reducir el tamaño de los modelos y, en consecuencia, su espacio de almacenamiento.

Poda

La poda es una técnica de compresión cuyo objetivo es la simplificación del modelo, eliminando conexiones entre las neuronas de las capas. Con este fin, se suprimen los pesos del modelo que son cercanos a cero, de modo que esa conexión deja de tener influencia sobre la decisión de clasificación. La elección de las conexiones (pesos) que deben podarse no es arbitraria, sino que se seleccionan aquellos pesos cuya magnitud es menor que cierto umbral.

Los beneficios de la poda se traducen en una menor necesidad de memoria, posibles mejoras en los tiempos de inferencia y mayor eficiencia energética. A cambio, según el tipo de poda que se aplique, esta puede deteriorar la calidad de predicción del modelo, si bien con una aplicación gradual donde en cada paso se incremente la ratio de poda en una proporción inferior a la anterior, esa pérdida de rendimiento llega a ser insignificante [25] .

Por otro lado, son varios los estudios que prueban las ventajas de la utilización de un modelo sobredimensionado y posteriormente podado, frente a un modelo denso de tamaño más reducido [26]. También para el uso de modelos preentrenados (*transfer learning)* de gran dimensión se recomienda la poda como método de adaptación del modelo al nuevo reto para reducir drásticamente su tamaño y tiempo de inferencia con apenas impacto en su precisión [27].

La técnica puede aplicarse de dos modos distintos: una vez el modelo ha sido entrenado o durante el entrenamiento. El primer método es más rápido, sin embargo, reentrenar mientras se poda permite al modelo seguir aprendiendo a medida que desaparecen sus pesos, reajustando el valor de los pesos, obteniendo mejores resultados.

El segundo modo utiliza máscaras binarias sobre los pesos que se quieren anular y va incrementando el volumen de elementos nulos hasta obtener el grado de dispersión deseado. De igual modo, se recomienda que la poda sea gradual y espaciada entre pasos de entrenamiento o épocas, dejando que el modelo pueda reentrenarse, reajustando su precisión antes de aplicar la siguiente poda, como indica la Figura 9. En este sentido, una poda continuada o demasiado drástica podría hacer que el modelo perdiera la capacidad de recuperarse [25] . Este es el método que se pondrá en práctica en este trabajo.

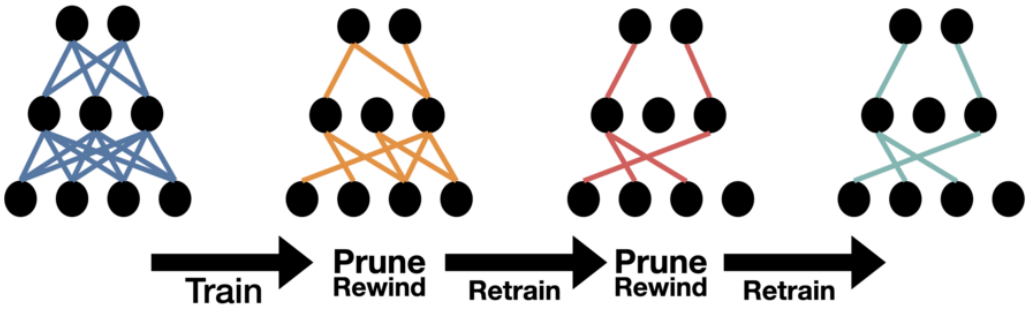

Figura 9. Diagrama de poda de una red neuronal [33].

Las podas pueden aplicarse sobre el modelo completo, pero también sobre una determinada parte o unas capas concretas. También hay que tener en cuenta que no todas las capas pueden podarse, puesto que se requiere que tengan parámetros entrenables, aunque, existen excepciones. Las capas de normalización del batch, aunque tienen parámetros entrenables no pueden podarse puesto que su objetivo es conseguir una normalización completa de los canales y una poda restaría su eficacia. Por su parte, las capas densas o completamente conexas son las que más potencial de poda tienen, ya que suelen tener más pesos (conexiones), aunque las convolucionales con un gran número de filtros también se ven beneficiadas por la poda.

Cuantización

La cuantización es una técnica que permite optimizar los modelos a través de una limitación en la precisión de los parámetros. Para ello, reduce el número de bits que se emplean para representar valores numéricos, que en modelos de aprendizaje profundo suelen ser números reales en coma flotante de simple precisión, que ocupan 32 bits (*Simple-Precision Floating-Point* o FP32).

Un modelo cuantificado podría almacenar sus parámetros de pesos en solo 16 u 8 bits (por ejemplo, utilizando FP16 o enteros de 8 bits, INT8), lo que permitiría disminuir su tamaño generando algunas ventajas adicionales como ahorro de espacio, una menor utilización de memoria RAM y una menor latencia en la inferencia, ya que en un mismo ciclo de reloj podrían realizarse varias operaciones con precisión reducida. Existen dos momentos distintos en los que es posible aplicar la cuantización: durante o tras el entrenamiento. La cuantización post-entrenamiento, que a su vez incluye distintas variantes, es más rápida y sencilla de aplicar frente a la cuantización aplicada durante el entrenamiento que, en cambio, ofrece mejores resultados.

Este segundo método, también llamado cuantización simulada o entrenamiento consciente de cuantización, permite reducir el tamaño de los modelos a una cuarta parte de su espacio inicial, con unas mejoras de latencia que van entre el 150 y 400% [29]. Por otro lado, toda cuantización lleva asociada una posible pérdida de la precisión de las predicciones según el tipo de modelo y la técnica empleada. Sin embargo, este método es el que tiene menor impacto sobre la precisión, siendo su valor poco significativo.

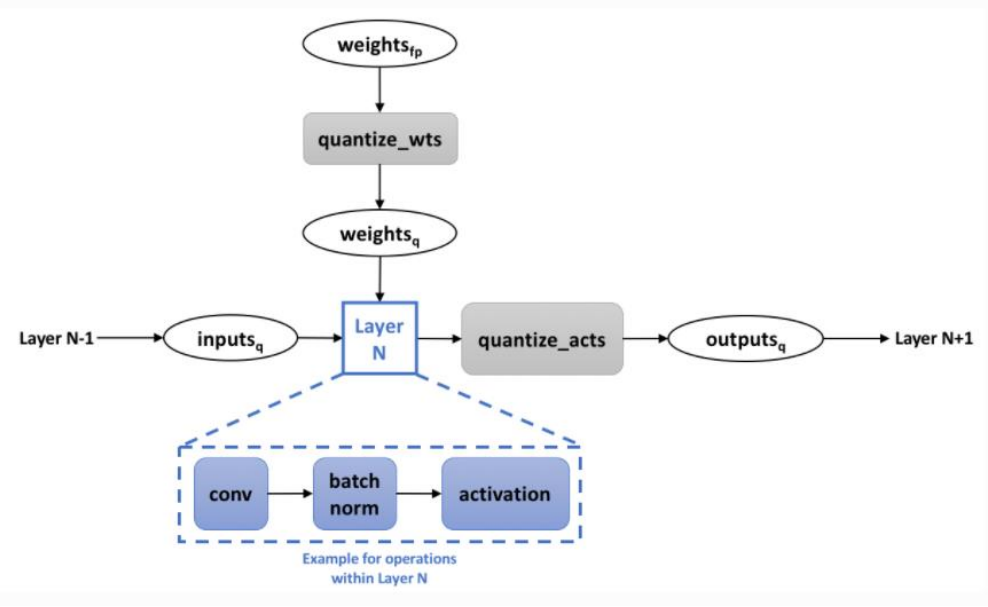

Figura 10. Diagrama del proceso de entrenamiento consciente de cuantización a nivel capa [30].

En la Figura 10 se puede ver un diagrama sobre el funcionamiento del entrenamiento consciente de la cuantización. Durante el entrenamiento, se aplica el gradiente y se calculan los pesos a máxima precisión, y estos son almacenados. Sin embargo, no se usan directamente, sino que para aplicarse sobre las siguientes capas se deben cuantificar (se redondean los valores decimales).

De igual modo, internamente cada capa realiza las operaciones con precisión decimal completa y solo una vez terminada la activación de la capa, se aplica la cuantización a su salida. De ese modo, se van recortando los valores en el paso entre capa y capa, tal y como muestra la Figura 10.

Una vez terminado el entrenamiento, se almacena el formato cuantizado y por tanto durante la inferencia solo se emplearán los parámetros con precisión reducida.

Capítulo 4

Implementación y resultados experimentales

En este capítulo se detalla todo el desarrollo experimental realizado desde la puesta en marcha de los modelos desde el marco CheXnet con unos determinados recursos computacionales, la exploración de hiperparámetros en estructuras DenseNet y ResNet con sus comparativas de desempeño y tiempos de entrenamiento, así como la aplicación de las técnicas de optimización anteriormente expuestas.

4.1. Recursos computacionales: software y hardware

Para la implementación de los modelos de CNN se utilizan los siguientes recursos de hardware y software:

	Hardware	Software
Nodo Volta	• 2x CPUs Intel(R) Xeon(R) Gold 6126 CPU @ 2.60GHz (12 cores por CPU con un total de 24 cores) sin hyperthreading activo. • Memoria RAM DDR4 de 64 GB. • Aceleradores gráficos: 1x GPU NVIDIA Tesla V100S de 32 GB GPU + 1x GPU NVIDIA Tesla V100 de 32 GB • Almacenamiento: SSD de 440 GB y SSD secundario de 3.6 TB.	• Tensorflow v2.0.0b1, Keras 2.3.1. • Para la ejecución de los kernels en GPU se emplea cuDNN v8.0.3, cuBLAS 10.1.130 junto con bibliotecas de CUDA v10.1 y NVIDIA CUDA v450.57.

Nodo Nowherman	• 2x CPU Intel(R) Xeon(R) CPU E5-2698 v4 @ 2.20GHz de 20 cores cada uno (total de 40 cores físicos) con hyperthreading activo (total de 80 cores lógicos). • Memoria RAM DDR4 de 512 GB. • Aceleradores gráficos: 4x GPU NVIDIA Tesla P100 SXM2 de 16 GiB interconectadas vía NVLink. • Almacenamiento: SSD de 2TB.	• Tensorflow v2.2.0, Keras v2.3.1 y Horovod v0.20.0 para entrenamiento distribuido utilizando la biblioteca de comunicaciones colectivas NVIDIA NCCL v2.7.8 para realizar los Allreduce de los gradientes del modelo en el esquema de paralelismo de datos. • Para la ejecución de los kernels en GPU se emplea cuDNN v8.0.3, cuBLAS v10.0.130, NCCL v2.7.3 (para Allreduce entre GPUs) y otras bibliotecas de CUDA v10.1 con driver NVIDIA CUDA v450.51.06.

4.2. Casos base

Los experimentos se han desarrollado empleando el marco de CheXnet como punto de partida, sobre el que se han añadido mediciones, se han realizado ajustes y exploración de los hiperparámetros y se han testeado arquitecturas de red: en particular, DenseNet y ResNet.

Este trabajo ha sido implementado en Python con el uso de Keras, una librería de Deep Learning, que contiene un módulo (*applications*) con diversos modelos disponibles e incluso opcionalmente preentrenados y puede ejecutarse sobre la plataforma TensorFlow.

Para el entrenamiento se emplea el conjunto de datos compuesto por 112.120 radiografías frontales etiquetadas con las 14 patologías previamente expuestas del NIH [18]. Las imágenes son redimensionadas a 224x224 acorde a la capa de entrada de los modelos de DenseNet121 y ResNet50 de Keras utilizados con los parámetros por defecto. Además, el 50% de las imágenes se someten a técnicas de *Data Augmentation* (reflexiones horizontales) que permite incluir variabilidad en las muestras consiguiendo más robustez y limitar el riesgo de sobreajuste.

Los hiperparámetros se eligen con un criterio común para DenseNet y ResNet, con el objetivo de que sean comparables. La intención de este estudio era realizar una pequeña exploración de los valores con mejor resultado para cada hiperparámetros, no obstante, los recursos y el tiempo de investigación

han sido limitados, por lo que se ha acotado esta búsqueda al factor de aprendizaje (LR del acrónimo inglés *Learning Rate*) y dimensión del batch.

Así pues, se definen de forma fija el resto de hiperparámetros como:

- Número de épocas = 100
- Criterio de paciencia para reducir el LR = 5 épocas
- Factor de reducción del LR = 0.1
- Mínimo LR = 1e-5
- Optimizador Adam
- Número de clases = 14

Adicionalmente se incluye un criterio de parada que, tras 15 épocas sin mejorar el rendimiento sobre el conjunto de validación, finaliza el entrenamiento. Esto permite acortar tiempos de entrenamiento.

La función pérdida que se emplea para evaluar el error es la entropía cruzada binaría L para 14 patologías descrita en el apartado 2.3. de este trabajo “Redes aplicadas a la imagen médica: CheXnet” subapartado “Multiclasificador”.

Durante el entrenamiento, todos los modelos se guardan el resultado de la función perdida sobre el conjunto de entrenamiento y validación en cada época, lo que permite observar su evolución. Con la intención de evitar que un exceso de épocas lleve a un sobreajuste en el conjunto de entrenamiento y que la función de pérdida se incremente en el conjunto de validación, se activa un punto de guardado. Este punto de guardado almacena el conjunto de pesos de la red que haya minimizado la función pérdida en el conjunto validación en cualquiera de las épocas, protegiendo de un posible descenso, y son los que finalmente se cargan a la red.

ResNet

Se entrena sobre la GPU NVIDIA Tesla V100S del nodo Volta los entrenamientos para modelos ResNet con la siguiente selección de hiperparámetros:

- Batch: 16, 32, 64 y 128
- LR: 0,001 y 0,01

La ejecución resulta viable para todos los experimentos excepto aquellos con tamaño de batch = 128, donde se requiere más memoria de la disponible por la GPU (32 GB) y no es posible entrenar el modelo. En este caso el uso de la segunda GPU no resulta útil porque al aplicar paralelismo de datos, la red debería replicarse en la otra GPU y procesaría otro batch del mismo tamaño ocupando la misma cantidad de memoria, por lo que el proceso seguiría excediendo la memoria disponible. Como solución a esta problemática, el modelo debería particionarse entre ambas GPUs utilizando el paradigma de paralelismo de modelo, pero esta mejora se propone como trabajo futuro.

Para la evaluación de los modelos se ha seleccionado la métrica área bajo la curva ROC (AUC del acrónimo inglés *Area Under the Curve Receiver Operating Characteristic o ROC*). El motivo de elección frente a otras métricas se debe a que el AUC no solo evalúa el desempeño global, sino también el equilibrio entre verdaderos positivos y falsos positivos, además de ser muy clarificador gráficamente. También, el uso de la ROC ha resultado ser más adecuado para análisis comparativos de aprendizaje automático frente a otras métricas como la precisión (*accuracy*) que no consideran los costes de clasificación ni la distribución por clases [32].

Tabla 2. Resultados de AUC por clase para ResNet

	LR = 0,01			LR = 0,001		
Patología	Batch = 16	Batch = 32	Batch = 64	Batch = 16	Batch = 32	Batch = 64
Atelectasia	73,5%	76,6%	74,0%	76,8%	80,4%	79,9%
Cardiomegalia	81,0%	86,0%	76,3%	82,4%	84,9%	84,8%
Derrame	84,7%	86,9%	84,8%	86,7%	88,1%	87,4%
Infiltración	67,7%	70,7%	68,6%	70,1%	67,5%	70,7%
Masa	71,1%	77,4%	74,3%	74,9%	81,1%	83,5%
Nódulo	60,3%	61,7%	60,8%	66,8%	73,2%	69,9%
Neumonía	62,5%	68,3%	62,3%	63,1%	61,7%	68,7%
Neumotórax	72,6%	84,2%	79,1%	83,0%	80,9%	86,1%
Consolidación	71,2%	71,1%	72,0%	72,7%	72,0%	72,3%
Edema	82,5%	88,2%	87,4%	86,4%	86,3%	88,4%
Enfisema	72,5%	84,8%	81,8%	88,0%	91,3%	94,4%
Fibrosis	64,3%	69,5%	62,5%	74,5%	73,9%	77,9%
Engrosamiento plural	71,0%	76,0%	74,6%	75,3%	78,8%	77,6%
Hernia	74,7%	72,5%	75,0%	75,3%	79,6%	82,9%
Promedio AUC:	**72,1%**	**76,7%**	**73,8%**	**76,9%**	**78,6%**	**80,3%**

Tabla 3. Tiempos de entrenamiento (en horas) de los modelos ResNet

	LR = 0,01			LR = 0,001		
	Batch = 16	Batch = 32	Batch = 64	Batch = 16	Batch = 32	Batch = 64
Tiempos de entrenamiento (h)	18,32	19,55	19,15	13,10	15,71	24,83
Número total de épocas	24	30	24	21	24	30
Tiempo por época (h)	0,76	0,65	0,80	0,62	0,65	0,83

En la Tabla 2 se pueden observar los resultados de AUC para los distintos modelos de ResNet abiertos por patología. Algunas patologías como la neumonía, cuyo diagnóstico se sabe que es complejo y tienden a mimetizarse con otras anormalidades, tiene las AUCs más bajas, en torno a 62-68%. Lo mismo ocurre con los nódulos, especialmente cuando el LR está en 0,01. Por el contrario, la cardiomegalia, el derrame, edema o enfisema son más detectables por el modelo, con AUC superiores al 80% en la mayoría de los casos.

Se aprecia también que la elección del LR tiene gran impacto ya que, para todos los tamaños de batch probados, un LR menor (0.001) supone un mejor rendimiento. Por su parte, la relación del tamaño del batch con el rendimiento no se ve tan clara, porque según que LR se tome, conviene seleccionar un batch u otro.

Si se analiza la duración del entrenamiento, se observa que los experimentos tienen unos tiempos por época heterogéneos y que parecen existir diferencias en los comportamientos según el LR. Para el LR = 0.001 a medida que se incrementa el tamaño del batch, también se alarga el tiempo de entrenamiento debido tanto a que se requieren más épocas para alcanzar la convergencia como a que esas épocas tienen mayor duración. Con el LR = 0.01 no parece existir una relación clara entre tiempos y tamaño del batch.

Para finalizar, se ha calculado el AUC promedio, con una media aritmética de cada clase, esto es, dándole la misma relevancia a cada patología independientemente de su peso en la muestra de datos. Teniendo en cuenta este dato, el experimento de tipo ResNet con mejor desempeño es el correspondiente al LR=0.001 y tamaño de batch = 64.

Se selecciona pues este modelo como objeto de estudio y se analiza su proceso de aprendizaje. La Figura 11 indica la evolución la pérdida en los conjuntos de entrenamiento y validación a lo largo de las épocas. Ambos conjuntos muestran un progreso inicial, relativamente suave y estable. El conjunto de entrenamiento tiene una disminución de la pérdida más marcado, mientras que el de validación muestra poca variación. Hay que considerar también que se parte de un modelo con pesos preentrenados y, en consecuencia, la pérdida inicial es baja: obtener estos resultados desde pesos aleatorios habría requerido entrenar durante muchas más épocas. Por otro lado, el conjunto de entrenamiento tiene inicialmente una pérdida superior a la de validación hasta que en la época 13 ambas líneas se intersectan. Este comportamiento es esperable considerando que se está haciendo uso de técnicas de *Data Augmentatión* al aplicar reflexiones horizontales, las cuales agregan variabilidad en el conjunto de entrenamiento a cada época, que funciona generando ruido y por tanto dificulta el aprendizaje minucioso sobre la muestra, lo que también resulta beneficioso porque contribuye a limitar el riesgo de sobreajuste

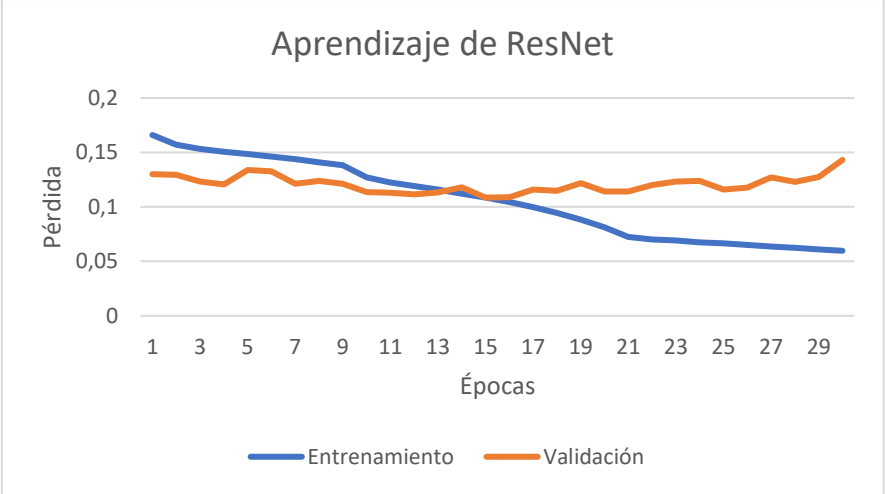

Figura 11. Evolución de la perdida durante el entrenamiento de ResNet

DenseNet

Para los modelos con estructura DenseNet, se dispone a lanzar los experimentos sobre una de las GPUs del nodo Volta con la siguiente selección de hiperparámetros:

- Batch: 16, 32, 64 y 128
- LR: 0.001 y 0.01

No obstante, igual que sucede en el modelo ResNet, las ejecuciones con un batch de tamaño = 128 no son posibles por limitaciones de memoria de la GPU.

El resto de los criterios se mantienen en consonancia con ResNet, para facilitar la comparación: misma función pérdida L, misma métrica de evaluación AUC.

Los resultados que se obtienen en este caso, expuestos en la Tabla 4, muestran nuevamente que la neumonía y los nódulos son las patologías con más dificultad de detección, en este caso por el modelo DenseNet, y edema y enfisema las más detectables.

No obstante, apreciamos algunas diferencias para LR = 0.01, donde parece haber una relación directa entre el tamaño del batch y la AUC, e inversa con la duración de entrenamiento: a mayor tamaño de batch, mejor resultado de AUC y más largo el proceso de entrenamiento. Por el contrario, los resultados parecen mostrar que para LR = 0.001 el tamaño de batch empleado es independiente o poco concluyente, ya que se obtienen AUCs muy similares en todos ellos.

Respecto a los tiempos de entrenamiento para DenseNet mostrados en la Tabla 5, se observa claramente que, pese a las distintas épocas que requiera cada experimento, su duración es muy similar en todos ellos. Algo que no sucedía en los modelos ResNet.

Tabla 4. Resultados de AUC por clase para DenseNet

	LR = 0,01			LR = 0,001		
Patología	Batch = 16	Batch = 32	Batch = 64	Batch = 16	Batch = 32	Batch = 64
Atelectasia	75,4%	75,2%	78,4%	79,5%	80,6%	82,5%
Cardiomegalia	78,9%	83,8%	86,3%	86,3%	85,9%	86,3%
Derrame	84,0%	85,4%	87,9%	88,6%	88,6%	88,4%
Infiltración	68,1%	69,6%	71,2%	70,7%	72,7%	72,7%
Masa	70,5%	74,9%	80,8%	81,8%	82,1%	80,7%
Nódulo	57,8%	60,7%	69,8%	72,3%	73,3%	71,5%
Neumonía	61,9%	67,5%	67,0%	75,0%	78,6%	77,6%
Neumotórax	76,9%	81,1%	86,2%	90,9%	87,4%	86,5%
Consolidación	71,4%	70,5%	70,8%	74,2%	72,7%	73,6%
Edema	81,8%	88,2%	89,5%	89,4%	87,3%	86,7%
Enfisema	72,3%	84,2%	87,9%	89,4%	85,9%	94,1%
Fibrosis	61,2%	67,3%	71,1%	72,7%	75,0%	73,6%
Engrosamiento plural	69,4%	75,3%	77,7%	80,2%	82,5%	81,8%
Hernia	72,6%	79,4%	82,1%	84,9%	81,5%	79,6%
Promedio AUC:	**71,6%**	**75,9%**	**79,1%**	**81,1%**	**81,0%**	**81,1%**

Tabla 5. Tiempos de entrenamiento (en horas) de los modelos DenseNet

	LR = 0,01			LR = 0,001		
	Batch = 16	Batch = 32	Batch = 64	Batch = 16	Batch = 32	Batch = 64
Tiempos de entrenamiento (h)	13,65	19,44	27,46	22,63	23,01	19,35
Total épocas	21	29	41	35	35	29
Tiempo por época (h)	0,65	0,67	0,67	0,65	0,66	0,67

Como análisis global, se puede decir que el mejor modelo tipo DenseNet se obtiene con la selección de LR = 0.001 y tamaño de batch = 64. Aunque con este mismo batch, incrementando el LR a 0.01 se pierde un 1% de AUC a cambio de reducir alrededor de 5h el tiempo de entrenamiento. Con el batch = 32 compensa menos incrementar el LR de 0.001 a 0.01 porque, aunque se reducen 5h también, el AUC baja 5%.

Paralelismo de datos con Nowherman

Para estudiar el proceso de entrenamiento usando paralelismo de datos, se replican los mismos experimentos realizados para ResNet y DenseNet en Volta sobre la máquina Nowherman que distribuye las tareas en 4 GPUs utilizando el plugin Horovod para Tensorflow [37].

Los tiempos extraídos para los modelos ResNet en Nowherman se muestran en la Tabla 6. Se aprecia que, pese a que los tiempos de entrenamientos son inferiores, para todos los experimentos excepto para uno (LR = 0.001 con Batch = 64), el entrenamiento en paralelo requiere más épocas. Esto puede suceder porque el algoritmo es estocástico y por tanto una ejecución puede tener resultados distintos. Por otra parte, en el paralelismo de datos cuando el LR no se ajusta bien, puede afectar a la velocidad de convergencia. La recomendación que se ha aplicado en este trabajo es escalar el hiperparámetro por el número de procesos p, aplicando la fórmula $LR_{paralelo} = LR * p$.

También destaca en los resultados de la Tabla 6 que la duración de una época es la misma en cualquiera de los experimentos, al contrario de lo que sucedía en la ejecución en Volta donde había mucha disparidad.

En la Tabla 7 se indican los tiempos de entrenamiento para los modelos DenseNet. Igual que ocurría con ResNet, todos los experimentos salvo el ya mencionado requieren más épocas para converger. La duración de una época en Nowherman se reduce en torno a una tercera parte en todos los experimentos y aquí de nuevo, el tiempo requerido para cada una es bastante similar.

Tabla 6. Tiempos de entrenamiento de los modelos ResNet en Nowherman

	LR = 0,01			LR = 0,001		
	Batch = 16	Batch = 32	Batch = 64	Batch = 16	Batch = 32	Batch = 64
Tiempos de entrenamiento (h)	7,76	7,90	5,94	5,47	5,37	4,47
Número total de épocas	41	41	32	29	28	24
Tiempo por época (h)	0,19	0,19	0,19	0,19	0,19	0,19

Tabla 7. Tiempos de entrenamiento de los modelos DenseNet en Nowherman

	LR = 0,01			LR = 0,001		
	Batch = 16	Batch = 32	Batch = 64	Batch = 16	Batch = 32	Batch = 64
Tiempos de entrenamiento (h)	7,87	11,54	7,37	10,57	8,21	5,32
Número total de épocas	38	59	39	51	42	28
Tiempo por época (h)	0,21	0,20	0,19	0,21	0,20	0,19

Para reparar en las diferencias entre ambos tipos de entrenamiento, se examina la comparativa en la Figura 12. Con los modelos DenseNet, obtenemos un ahorro del tiempo de entrenamiento total de entre un 41-73% usando paralelismo de datos. En ResNet, este ahorro es incluso más pronunciado, entre un 58-82% de reducción de tiempos.

Los experimentos que más se benefician del paralelismo de datos, independientemente del tipo de arquitectura ResNet o DenseNet, son aquellos con un batch de mayor tamaño: por ejemplo, con un batch de 64 muestras la reducción es del 70-80%, mientras que los batch de 16 muestras en ningún caso superan el 60% de ahorro en tiempos.

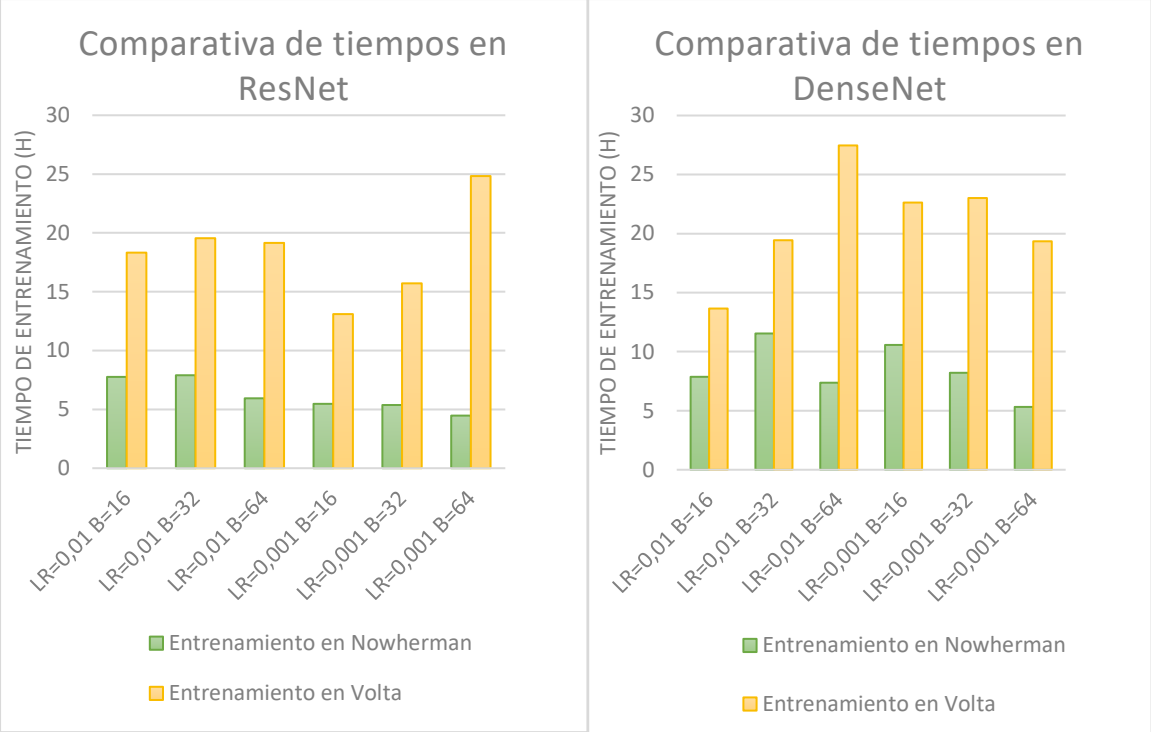

Figura 12. Comparativa de tiempos de entrenamiento: 12a) A la izquierda con modelos tipo ResNet, 12b) a la derecha, tipo DenseNet

DenseNet: Afinación

En los apartados anteriores se han lanzado experimentos usando los modelos DenseNet, misma estructura que usa CheXnet, y sin embargo el desempeño que se obtiene (Tabla 4) es inferior en todos los casos al mostrado por CheXnet (Tabla 1).

Debido a que el propio modelo de CheXnet no es replicable tanto por el componente de aleatoriedad de los modelos como por la falta de información de ciertos hiperparámetros, se trata de afinar los experimentos modificando el criterio de paciencia para reducir el LR y el mínimo LR. En los nuevos experimentos se toma el tamaño de batch = 16 por ser el que CheXnet emplea y el resto de hiperparémetros se mantienen. Así pues, se ejecutan los siguientes experimentos con el modelo DenseNet:

- Batch = 16, LR = 0.001, Min LR = 1e-7, Criterio de paciencia = 10
- Batch = 16, LR = 0.01, Min LR = 1e-7, Criterio de paciencia = 10

Tabla 8. Resultados de AUC por clase para experimentos DenseNet con criterios afinados.

	Min LR = 1e-5, Cr. Pac. = 5		Min LR = 1e-7, Cr. Pac. = 10	
Patología	LR = 0,01	LR = 0,001	LR = 0,01	LR = 0,001
Atelectasia	75,4%	79,5%	76,3%	82,3%
Cardiomegalia	78,9%	86,3%	83,8%	87,5%
Derrame	84,0%	88,6%	85,7%	88,5%
Infiltración	68,1%	70,7%	70,3%	72,3%
Masa	70,5%	81,8%	73,8%	82,4%
Nódulo	57,8%	72,3%	61,8%	73,8%
Neumonía	61,9%	75,0%	71,7%	78,9%
Neumotórax	76,9%	90,9%	81,3%	89,8%
Consolidación	71,4%	74,2%	71,2%	74,4%
Edema	81,8%	89,4%	87,7%	89,1%
Enfisema	72,3%	89,4%	82,6%	92,5%
Fibrosis	61,2%	72,7%	63,6%	76,1%
Engrosamiento plural	69,4%	80,2%	74,3%	79,4%
Hernia	72,6%	84,9%	72,1%	83,4%
Promedio AUC:	**71,6%**	**81,1%**	**75,4%**	**82,2%**

Tabla 9. Tiempos de entrenamiento (en horas) para experimentos DenseNet con hiperparámetros afinados.

	Min LR = 1e-5, Cr. Pac. = 5		Min LR = 1e-7, Cr. Pac. = 10	
	LR = 0,01	LR = 0,001	LR = 0,01	LR = 0,001
Tiempos de entrenamiento (h)	13,65	22,63	21,79	24,33
Número total de épocas	21	35	28	33
Tiempo por época (h)	0,65	0,65	0,78	0,74

Como se aprecia en la Tabla 8, la nueva selección de hiperparámetros con Min LR = 1e-7 y criterio de paciencia = 10, muestra a nivel global mejores resultados en términos de AUC: una mejora del 3.8% para el caso LR = 0.01 y del 1.1% con LR = 0.001, aunque también requieren tiempos de entrenamiento más largos, un 59.6% y 7.5% respectivamente, en parte porque la duración de las épocas también se incrementa, como indica la Tabla 9. Esta tendencia también se ve a nivel patología, donde todas las AUC correspondientes a la versión afinada con LR = 0.01 obtienen mejor resultado que los casos base. Con en LR = 0.001 la mejoría de la versión afinada a nivel clase no es tan marcada (algunas patologías como el neumotórax, la hernia o el derrame tienen resultados parecido e incluso ligeramente superiores en el escenario base) aunque sí es visible en algunas patologías como neumonía o fibrosis y, especialmente a nivel global, donde el modelo de batch = 16, LR = 0.001, Min LR = 1e-7 y criterio de paciencia = 10 obtiene los mejores resultados globales de AUC de todos los experimentos realizados tipo DenseNet.

Se selecciona este modelo entonces, para profundizar en su estudio y se analiza el comportamiento de su aprendizaje, que se muestra en la gráfica de la Figura 13. Aquí se observa por un lado que el

conjunto de entrenamiento disminuye continuamente su pérdida tras cada época, mientras que el de validación, aunque también consigue reducir la pérdida a lo largo del proceso, su comportamiento es inestable y variable. Sin embargo, el conjunto de entrenamiento tiene, de partida, una pérdida superior al de validación, es decir, lo está clasificando con más errores, que como ya se ha comentado, se debe al uso de técnicas de *Data Augmentation*, y se mantiene así hasta la época 22 donde estas líneas se cruzan y el modelo empieza a clasificar mejor el conjunto de entrenamiento que el de validación.

Al final del proceso hay, de nuevo, otro cruce de líneas y todavía poca estabilidad por parte del conjunto de validación, lo que hace preguntarse si en futuras épocas el modelo podría seguir convergiendo. Dado que el modelo finaliza en la época 32 porque tras 15 épocas (criterio de parada) no ha habido mejoría (el menor valor de pérdida se obtiene en la época 17), se propone lanzar de nuevo el mismo experimento con más épocas y estudiar si podría converger con mejores índices de clasificación.

Se ejecuta el experimento DenseNet con batch = 16 y LR = 0.001 y con un criterio de parada fijo tras 50 épocas (independiente de la función pérdida) como se refleja en la Figura 14. Sin embargo, estos resultados, aunque deben guardar cierta similitud con la ejecución anterior puesto que parten de la misma definición de hiperparámetros, no son 100% reproducibles debido a la componente estocástica del entrenamiento y a que la ejecución de operaciones internas en la GPU puede tener un orden distinto.

En este caso los resultados indican que alargar el proceso de entrenamiento no aporta beneficios. De hecho, en este experimento de 50 épocas, la mejor AUC se obtiene en la época 19.

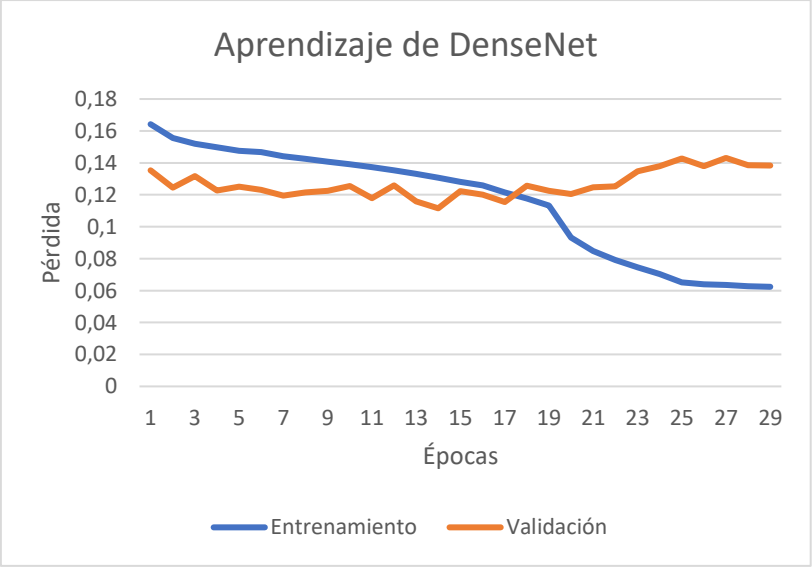

Figura 13. Evolución de la perdida durante el entrenamiento de DenseNet

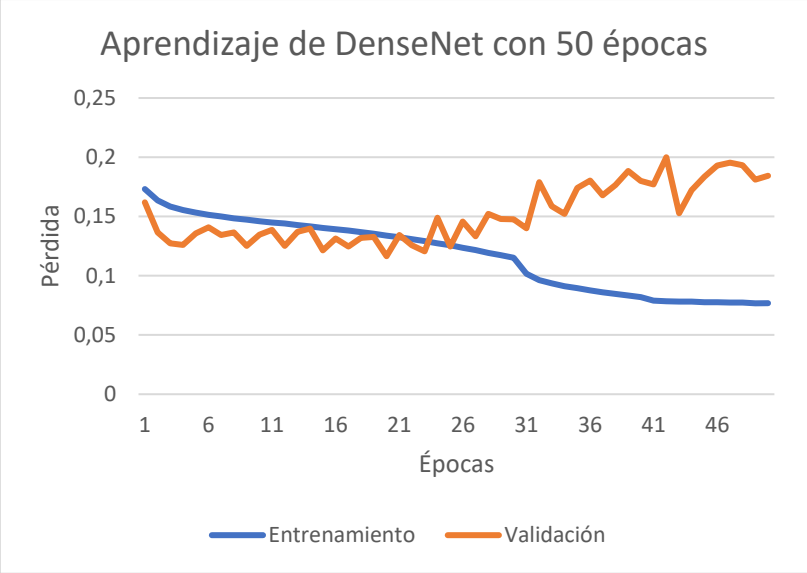

Figura 14. Evolución de la perdida durante el entrenamiento de DenseNet con 50 épocas

4.3. Poda

La técnica de poda que se ha empleado en este trabajo es la poda durante el entrenamiento. Este método permite eliminar las conexiones menos relevantes de los modelos para simplificarlos. Como se ha explicado anteriormente, las podas graduales tienen menor impacto en el deterioro de precisión del modelo, y es por eso que se elige una aplicación que vaya eliminando pesos con una progresión polinomial.

Los modelos elegidos para entrenarse con poda son el modelo ResNet con batch = 64, LR = 0.001 y el DenseNet con batch = 16, LR = 0.001 de la versión afinada (Min LR = 1e-7, Cr. Paciencia = 10) ya que son los experimentos de cada tipo de modelo que mejor resultados de AUC presentan.

Dentro de la poda durante el entrenamiento, existen distintas posibilidades como son, por ejemplo, podar un determinado número de capas por tipología, posición o profundidad. Si bien lo más común es que se aplique sobre las capas densas por tener más profundidad y por tanto más posibilidades, o convolucionales. En los modelos ResNet y DenseNet no existen capas densas, excepto la capa de salida dónde la poda no es recomendable. Es por esto que se decide aplicar la poda exclusivamente sobre los pesos de las capas convolucionales ya que los sesgos de las capas convolucionales tampoco son el objeto de la poda, dado la proporción tan reducida que suponen.

Para estos experimentos se han elegido aplicar la poda sobre los modelos tomando distintos criterios:

- ***Poda del 50% sobre las capas convolucionales:*** Progresión polinomial de grado 4, que se inicia al 10% de poda a la mitad de la primera época y cada vez aplica la poda a menor velocidad hasta alcanzar el 50% de poda al final de la séptima época.

- ***Poda del 70% sobre las capas convolucionales:*** Progresión polinomial de grado 4, que inicia al 10% de poda a la mitad de la primera época y cada vez aplica la poda a menor velocidad hasta alcanzar el 70% de poda al final de la séptima época.
- ***Poda adaptativa:*** Aplica sobre las capas convolucionales un porcentaje de poda correspondiente según su profundidad (número de pesos de la capa). Como en los casos anteriores, se aplica con una progresión polinomial de grado 4, desde el 10% hasta el porcentaje de poda determinado para cada capa, elegidos con el siguiente criterio:
    - ResNet:
        - Capas de 64 canales de salida: 25% de poda
        - Capas de 256 canales de salida: 40% de poda
        - Capas de 512 canales de salida: 55% de poda
        - Capas de 1024 canales de salida: 70% de poda
        - Capas de 2048 canales de salida: 85% de poda
    - DenseNet:
        - Capas de 32 canales de salida: 25% de poda
        - Capas de 64 canales de salida: 50% de poda

Algunos estudios han probado que el número de canales de salida o filtros de una capa influye en su capacidad de tolerar una mayor ratio de poda respetando su precisión [31]. Por este motivo se ha propuesto el enfoque de poda adaptativa. Sin embargo, hubiera sido deseable emplear un criterio uniforme entre ResNet y DenseNet para facilitar su comparativa, pero dado que ambas redes tienen arquitecturas con dimensiones salida dispares, se ha considerado apropiado ajustar el criterio adaptativo a las particularidades de cada red.

El uso de la poda según los resultados mostrados en la Tabla 10, tiene una penalización leve en el rendimiento global (AUC promediado entre todas las patologías) de los modelos DenseNet, con bajadas en torno al 1-1,5%. En el caso de la ResNet, el tipo de poda polinomial fija tiene una repercusión leve en AUC, incluso llegando a mejorar el desempeño en el caso de la poda al 70% donde se incrementa el AUC respecto al modelo sin podar un 1,5%. En el caso de la poda adaptativa para ResNet, bien por el tipo de poda o por el diseño del criterio aplicado, sí se aprecia un impacto notable con una pérdida de AUC superior al 8%.

Por otra parte, si bien no es el objetivo perseguido con la poda, su aplicación en las 7 primeras épocas también reporta beneficios en el tiempo de entrenamiento para ambos modelos: ResNet y DenseNet en todos los experimentos. Esto se refleja bien en la cantidad de épocas hasta alcanzar la convergencia que es menor, en la duración de las épocas que también se acorta o en ambos casos.

Tabla 10. Resultados tras la poda para ResNet y DenseNet

	DenseNet				ResNet			
	Sin poda	Poda 50%	Poda 70%	Poda Adaptativa	Sin poda	Poda 50%	Poda 70%	Poda Adaptativa
AUC promedio	82,2%	81,2%	80,8%	80,9%	80,3%	80,6%	81,8%	72,5%
Tiempos de entrenamiento (h)	24,33	18,65	19,18	23,33	24,83	16,24	17,14	14,09
Número total de épocas	33	25	27	34	30	24	25	21
Tiempo por época (h)	0,74	0,75	0,71	0,69	0,83	0,68	0,69	0,67

La principal ventaja obtenida con la poda es, sin embargo, la simplificación del modelo y la reducción de almacenamiento que conlleva la reducción de parámetros de peso. Sin embargo, esta ventaja no es apreciable hasta que no se aplica la compresión, porque las operaciones que se realizan en el proceso de entrenamiento siguen empleando pesos con un esquema de almacenamiento denso, de modo que aquellos pesos que son nulos o cero siguen ocupando el mismo espacio inicial. La compresión permite reducir el espacio ocupado por esos valores a cero, aunque dependiendo de su posición y agrupación puede tener mayor coste (se requiere el uso de formatos de almacenamiento disperso para localizar las coordenadas de los elementos no cero).

Otra ventaja de la poda es aprovechar la naturaleza dispersa de la matriz utilizando núcleos de álgebra lineal para matrices dispersas. Sin embargo, esta opción no siempre es conveniente pues la estructura dispersa provoca saltos en la lectura del vector o matriz densa por la que se multiplica, causando fallos de cache y provocando que se tengan que traer desde memoria principal los datos a los que se necesita acceder. Esto provoca que la velocidad de ejecución de los núcleos computacionales esté limitada por la velocidad de acceso a memoria, lo que ralentiza en consecuencia el proceso de entrenamiento. Está opción solo empieza a compensar cuando la ratio de poda (o de dispersión) es superior al 85-90% de los pesos, que no es el caso de los experimentos realizados en este estudio.

Para clarificar el proceso de compresión, se propone un ejemplo teórico empleando los experimentos ResNet y DenseNet con una ratio de poda del 50% en TensorFlow, expuesto en la Tabla 11 (nótese que para la implementación de la poda, TensorFlow puede añadir a los modelos capas específicas para tal propósito y, en consecuencia, incrementar el número de parámetros). Los modelos disponen de una cantidad de pesos, los cuales un 50% de ellos tienen un valor cero a consecuencia de la poda. Con la compresión, se eliminan los pesos cero y quedan los restantes. Asumiendo hipotéticamente que el almacenamiento solo reflejara el espacio ocupado por los pesos no nulos y éstos están en formato FP32 (cada uno de ellos ocupa 32 bits), podemos obtener el almacenamiento total antes y después de comprimir, lo que nos da en ambos casos una reducción del 50%: la misma proporción de poda aplicada a los modelos.

En la práctica, sin embargo, la compresión entraña otras dificultades porque, como se ha explicado previamente, la posición y disposición de esos valores cero influye en cómo se almacenan y en la eficiencia de los algoritmos de compresión para reducir su tamaño.

Tabla 11. Estudio teórico del impacto en almacenamiento aplicando compresión.

	ResNet		DenseNet	
	Pesos totales	Almacenamiento (MiB)	Pesos totales	Almacenamiento (MiB)
Antes de comprimir	47.071.485	188,29	13.922.488	55,96
Después de comprimir	23.616.398	94,46	7.051.854	28,20
Reducción	**50%**		**50%**	

En la Figura 15 se puede ver el espacio de almacenamiento ocupado por los modelos ResNet y DenseNet y la reducción real tras la aplicación de la compresión. En el modelo ResNet, la existencia de esos metadatos provoca que el porcentaje de reducción total no refleje la proporción de pesos eliminados, por ejemplo, en la poda fija donde se sabe claramente que la eliminación ha sido del 50 o 70% y las reducciones son del 33% y 47% respectivamente. La DenseNet en cambio, es más efectiva en este sentido llegando a alcanzar tasas de reducción de almacenamiento del 46 y 60% para los mismos casos.

Estos resultados apuntan a que la poda tiene más impacto negativo sobre el desempeño de los modelos DenseNet que ResNet, sin embargo, sobre esta última no se obtiene tanta reducción de almacenamiento.

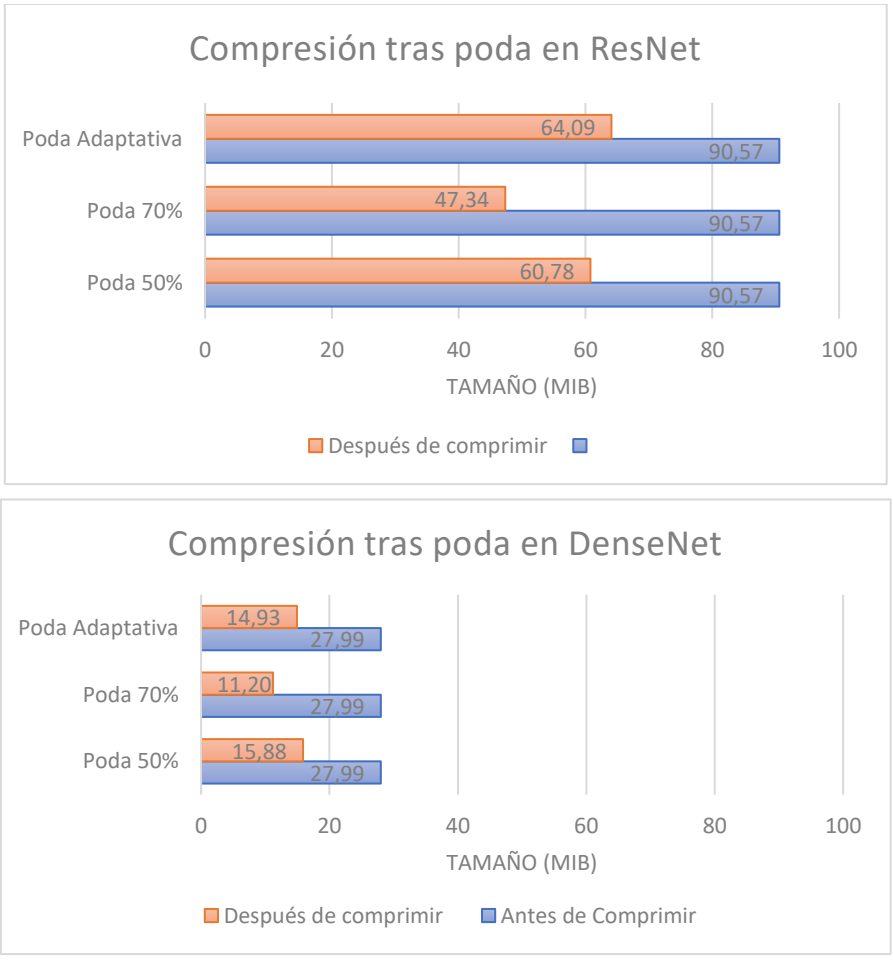

Figura 15. Espacio de almacenamiento de los modelos con y sin compresión.

4.4. Cuantización

Entre las distintas opciones de cuantización sobre CNNs, para este estudio se ha seleccionado en entrenamiento consciente de cuantización por sus ventajas es cuanto a limitación de la pérdida de precisión y se ha aplicado sobre los modelos ResNet con hiperparámetros LR = 0.001 y Batch = 64, y DenseNet con hiperparámetros LR = 0.001, Batch = 16 del experimento afinado (Min LR = 1e-7, Cr. Paciencia = 10) por ser los modelos que mejor resultado de AUC han mostrado.

Sin embargo, este entrenamiento con cuantización no ha sido posible en el caso de la red DenseNet, debido a que los requerimientos de memoria excedían los 64GB de los que dispone la GPU, por lo que el estudio solamente se centra en el modelo ResNet.

Para estos experimentos se han elegido aplicar la cuantización tomando distintos criterios:

- Cuantización sobre todas las capas convolucionales del modelo.
- Cuantización de todas las capas del modelo que soporten el proceso de cuantización, esto es: convolucionales, de activación, de *pooling*, *padding* y aditivas.

Los resultados recogidos en la Tabla 12 muestran de nuevo que en una técnica de optimización como es la compresión tiene un impacto positivo en el proceso de entrenamiento al acortar su duración. En ambas modalidades de cuantización, tanto sobre las capas convolucionales como sobre el total, se observa una reducción del tiempo de entrenamiento de más de 10h. Esto está motivado por un menor número de épocas para alcanzar la convergencia, como por la propia duración de las épocas, que se acorta en más de un 25%.

Sin embargo, la cuantización también repercute negativamente en el desempeño de los modelos, ya que se pierde la precisión en los parámetros (se recortan decimales al almacenarse solo ocupando 16 bits) como se refleja en la Tabla 12. En el experimento de cuantización sobre capas convolucionales cae el AUC promedio en torno a un 3% y también es así para la cuantización sobre todas las capas.

Se aprecia que existe una gran similitud entre los resultados de los dos tipos de cuantización, que se debe a la propia estructura de la ResNet. Como muestra a continuación en la Figura 16, la gran parte de las capas de ResNet son convolucionales, por lo que la aplicación de cuantización o no sobre otras capas, considerando su escasa presencia, apenas tiene impacto.

Tabla 12. Resultados de experimentos ResNet aplicando cuantización durante el entrenamiento.

		Cuantización	
	Sin cuantización	Capas convolucionales	Todas las capas
AUC promedio	80,3%	77,1%	77,1%
Tiempos de entrenamiento (h)	24,83	14,28	13,74
Número total de épocas	30	24	23
Tiempo por época (h)	0,83	0,60	0,60

Como se explicaba en el Capítulo 3, subapartado de Cuantización y en el diagrama de la Figura 10, aunque el algoritmo de entrenamiento emplee pesos cuantizados (FP16) sobre las capas, almacena los pesos en formato de completa precisión (FP32). Es por eso, que la reducción no es visible desde los ficheros de pesos, pero sí durante el entrenamiento e inferencia, donde los cálculos se realizan con los pesos cuantizados.

A continuación, se analiza de forma teórica la reducción esperable en el almacenamiento de los pesos, que en el caso de la red ResNet pueden pertenecer a tres tipos de capas: de normalización del batch, de salida (de tipo denso o completamente conexo) y convolucionales, siendo estas últimas las que abarcan casi la totalidad de los parámetros. En la Figura 16 se muestra el espacio ocupado por el total de parámetros por capa en cada supuesto. En el experimento sin cuantización, todos los pesos se almacenan en FP32, esto es, ocupando 32 bits. Cuando la cuantización se aplica sobre las capas convolucionales, como era el caso de nuestro primer experimento cuantizado, vemos que la reducción de los pesos es casi del 50% ya que se corresponde al grupo mayoritario de pesos. Si esta cuantización se aplica además sobre las capas de normalización del batch, aún se puede obtener una ligera reducción más de 0.2 MiB, muy poco significativa. Con estos resultados se extrae que la cuantización permite grandes reducciones en el almacenamiento, si bien empleando el criterio de cuantización solo en capas convolucionales sería suficiente.

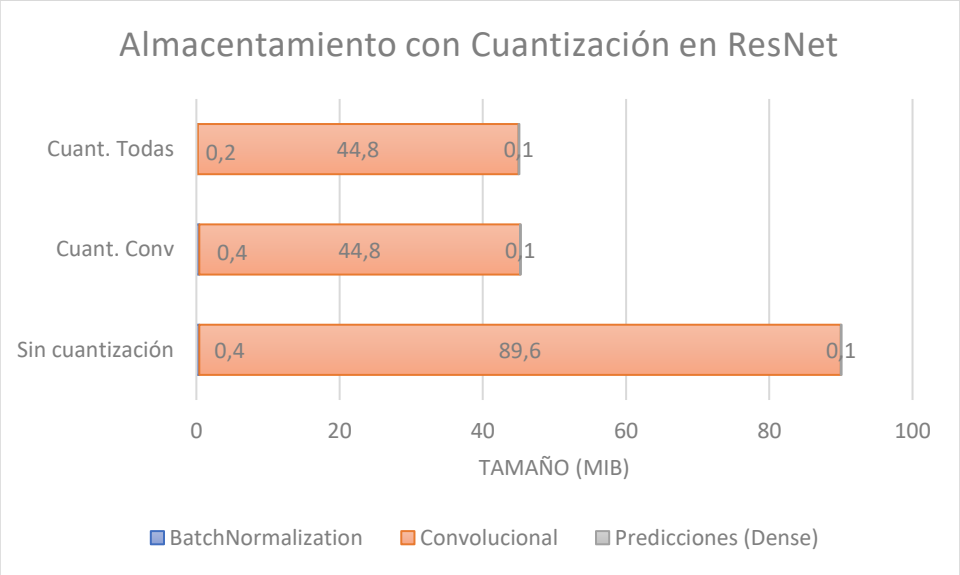

Figura 16. Almacenamiento en Mib de un modelo ResNet al aplicar Cuantización.

Capítulo 5

Conclusiones

A lo largo de este trabajo se han puesto en marcha modelos de diagnóstico de radiografías de tórax, tales como CheXnet, y se han propuesto modelos alternativos empleando arquitecturas de CNN de tipo ResNet y DenseNet.

Los distintos experimentos han mostrado que la configuración de hiperparámetros que mejores resultados ofrece en términos de AUC promedio en todas las patologías es la que actualmente se utiliza en el modelo de CheXnet. Sin embargo, este modelo también ha mostrado un amplio margen en cuanto a reducción de almacenamiento, tiempos de entrenamiento y uso de memoria con la aplicación de técnicas de optimización.

Por un lado, las técnicas de entrenamiento distribuido empleando una máquina con cuatro GPUs, en comparación con la ejecución en una única GPU, han mostrado que la reducción de tiempos de entrenamiento es notable, especialmente en la arquitectura ResNet, donde se consiguen las mayores reducciones. Este beneficio, además, está relacionado con el tamaño del batch empleado durante el entrenamiento, donde un mayor batch repercute acortando la duración del proceso de entrenamiento.

Por otra parte, se han aplicado técnicas de compresión como poda y cuantización. Estos métodos también han reducido los tiempos de entrenamiento, sin ser el objetivo de uso, y el espacio de almacenamiento de los modelos. La poda del 50% y 70% de los pesos ha supuesto reducciones de espacio proporcionales en DenseNet, a cambio de una pérdida leve de certeza en la clasificación. Por el contrario, las mismas podas en ResNet, han tenido una reducción de espacio menor, sin embargo, su aplicación ha acortado las épocas necesarias hasta alcanzar la convergencia e incluso ha mejorado el rendimiento de algunos experimentos, incrementando hasta un 1,5% su AUC. En el caso de querer disminuir más drásticamente espacio en el caso de la ResNet, se ha visto que las técnicas de cuantización resultan más efectivas con reducciones teóricas hasta del 50%, pese a tener un coste de desempeño en la clasificación de un 3% de pérdida de AUC.

Por todo ello, el método de optimización recomendable de los estudiados en el presente trabajo es relativo, según el recurso que se desee mejorar. Si se pretende acelerar el proceso de entrenamiento, sería conveniente, en el caso de que se disponga de recursos de hardware necesarios, el paralelismo de datos.

Si, por el contrario, prima el ahorro en el espacio de almacenamiento sería preferible el uso de técnicas de compresión, aunque la elección de una u otra depende del tipo de arquitectura y la precisión de clasificación que requiera el modelo según el problema a tratar.

Como trabajo futuro, se propone tanto la ampliación del análisis exploratorio de CheXnet como una continuación para observar el impacto sobre la inferencia de las técnicas probadas.

En el primer planteamiento podrían incluirse otros modelos de redes neuronales convolucionales y el testeo de hiperparámetros no explorados, así como un análisis más profundo en el *learning rate* inicial, ya que solo se evaluaron dos valores y han mostrado gran relevancia en el desempeño de los modelos.

Por otra parte, como continuación de este trabajo, se propone el análisis de impacto de las técnicas de optimización en la inferencia, así como la aplicación de poda o compresión posterior al entrenamiento y su comparativa frente a la aplicación actual durante el entrenamiento.

Bibliografía

[1] A. M. Santos, J. A. Martín. “Organización y gestión de la radiología urgente.” *Radiología*, 53 (2011):7-15.

[2] Pérez de Diego, E. et al (2019) “¿Hacemos un buen uso de las radiografías de tórax en Urgencias para el diagnóstico de neumonía?” en *XXIV Reunión de la Sociedad Española de Urgencias de Pediatría*. Disponible en <https://seup.org/pdf_public/reuniones/2019/CC/CC_166.pdf>

[3] *Portal de MedicinePlus* <https://medlineplus.gov/spanish/xrays.html>

[4] Neuman et al. Variability in the interpretation of chest radiographs for the diagnosis of pneumonia in children. *Journal of hospital medicine*, 7(4): 294–298, 2012.

[5] A. Morales. “El informe en radiología de urgencias ¿Podemos excluir la radiografía simple?” En Artigas, J.M. “Radiología de Urgencias. La oportunidad en la crisis*,” IV Congreso Nacional de Radiología de Urgencias*. Madrid. Octubre (2013).

[6] Adrian P. Brady. “Error and discrepancy in radiology: inevitable or avoidable?” *Insights into imaging* vol. 8,1 (2017): 171-182.

[7] Matich, J. D. (2001). *Redes Neuronales: Conceptos Básicos y Aplicaciones*. Universidad Tecnológica Nacional. Disponible en <https://www.frro.utn.edu.ar/repositorio/catedras/quimica/5_anio/orientadora1/monograias/matich-redesneuronales.pdf>

[8] García Álvarez, P. J. (2018). *Aplicación de redes neuronales en la predicción de mortalidad por neumonía en “*Revista Médica Electrónica*”* vol.40 no.5. Disponible en <http://scielo.sld.cu/scielo.php?script=sci_arttext&pid=S1684-18242018000501361>

[9] Martínez Robles, M. (2007). “Clasificación del grado de astrocitoma cerebral infantil. Segmentación de imágenes, morfología matemática y redes neuronales” en PortalesMedicos.com. Disponible en <https://www.portalesmedicos.com/publicaciones/articles/407/5/articles.php?ToDo=viewFavourites%5C>

[10] López Pérez, M. Nuevos modelos estadísticos para detección de patrones de hipo/perfusión-metabolismo en imágenes de tomografía funcional cerebral. Tesis doctoral. Universidad de Granada. Disponible en <http://hera.ugr.es/tesisugr/18896789.pdf>

[11] *Clasificación de radiografías de tórax con redes neuronales convolucionales.* <https://ichi.pro/es/diagnostico-de-neumonia-con-cnn-823947836421>

[12] Rajpurkar, R. et al. (2017). *CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning.*

[13] Hijazi, S. Kumar, K. y Rowen, C. (2015). "Using Convolutional Neural Networks for Image Recognition" in *Cadence*. Disponible en <https://ip.cadence.com/uploads/901/cnn_wp-pdf>

[14] Krizhevsky, A. Sutskever, I. y Hinton, G. E. *ImageNet Classification with Deep Convolutional Neural Networks*. Disponible en <https://www.nvidia.cn/content/tesla/pdf/machine-learning/imagenet-classification-with-deep-convolutional-nn.pdf>

[15] TowardsDataScience.com. *How GPU accelerte Deep Learning* <https://towardsdatascience.com/how-gpus-accelerate-deep-learning-3d9dec44667a>

[16] Wang et al. Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases*. arXiv preprint arXiv:1705.02315, 2017*

[17] Yao et al. Learning to diagnose from scratch by exploiting dependencies among labels. *arXiv preprint arXiv:1710.10501*, 2017.

[18] NIH Chest X-ray Dataset of 14 Common Thorax Disease Categories de *National Institutes of Health - Clinical Center***.**

[19] Zhang, A. et al. "Modern Convolutional Neural Network" en *Dive into Deep Learning*. Disponible en <https://d2l.ai/chapter_convolutional-modern/index.html>

[20] *Web Oficial de Pytorch* <pytorch.org>

[21] Wen, L. et al. "A negative correlation ensemble transfer learning method for fault diagnosis based on convolutional neural network" en *Mathematical Biosciences and Engineering*, 2019. Disponible en <https://www.aimspress.com/article/10.3934/mbe.2019165/figure.html>

[22] Hui Tan, H. y Hann Lim, K. "Vanishing Gradient Mitigation with Deep Learning Neural Network Optimization" in *2019 7th International Conference on Smart Computing & Communications (ICSCC).* Disponible en <https://ieeexplore.ieee.org/abstract/document/8843652>

[23] Sergeev, A. y Del Balso, M. (2018) "Horovod: fast and easy distributed deep learning in TensorFlow" en *Arxiv 1802.05799*. Disponible en <https://arxiv.org/pdf/1802.05799.pdf>

[24] Timdetmers.com *How to Parallelize Deep Learning on GPUs* <https://timdettmers.com/2014/10/09/deep-learning-data-parallelism>

[25] Pietron, M. y Wielgosz, M. (2020) “Retrain or Not Retrain? - Efficient Pruning Methods of Deep CNN Networks” *en Computational Science – ICCS 2020, pp* 452-463. Disponible en <https://link.springer.com/chapter/10.1007/978-3-030-50420-5_34>

[26] H. Zhu, M. y Gupta, S. (2018). *To prune or not to prune: Exploring the efficacy of pruning for model compression*. Disponible en <https://openreview.net/pdf?id=Sy1iIDkPM>

[27] Jacob Guildenblat. *Pruning deep neural networks to make them fast and small.* <https://jacobgil.github.io/deeplearning/pruning-deep-learning>

[28] TowardsDataScience.com. *Pruning Deep Neural Networks* <https://towardsdatascience.com/pruning-deep-neural-network-56cae1ec5505>

[29] *Tensorflow* <https://www.tensorflow.org/model_optimization/guide>

[30] *Portal de Distiller* <https://intellabs.github.io/distiller/quantization.html>

[31] Ceruso, S. 2018. *Estrategias de poda en redes neuronales convolucionales*. Trabajo Final de Master. UNED.

[32] Provost, F. Fawcett, T y Kohave, R. The case against accuracy estimation for comparing induction algorithms. Disponible en <https://eecs.wsu.edu/~holder/courses/cse6363/spr04/pubs/Provost98.pdf>

[33] “A foolproof way to shrink deep learning models*”. Portal web del MIT*. Disponible en <https://news.mit.edu/2020/foolproof-way-shrink-deep-learning-models-0430>

[34] *Rajpurkar, P et al. CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning. 2017.*

[35] NVIDIA. *Documentación de la biblioteca NVIDIA NCCL* <https://docs.nvidia.com/deeplearning/nccl/install-guide/index.html>

[36] IBM. *Documentación de la biblioteca IBM DLL*. <https://www.ibm.com/support/knowledgecenter/SS5SF7_1.6.0/navigation/pai_getstarted_ddl.html>

[37] Horovod. *Documentación de la biblioteca Horovod* <https://horovod.readthedocs.io/en/stable/>