UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

TRABAJO FIN DE MÁSTER

Sistema de ayuda al diagnóstico médico basado en Deep Learning

Máster Universitario en Ingeniería Informática

Autor: Mirón Cao, Adrián

Tutores: Sánchez Montañés, Manuel

Corbacho Abelaira, Fernando

Departamento de Ingeniería Informática

FECHA: Septiembre, 2019

I. Introducción

En este documento, se realizará un estudio de las distintas librerías y *frameworks* enfocados a Deep Learning en Python utilizados en la actualidad. Se estudiarán las ventajas e inconvenientes que tiene cada *framework*, seleccionando aquellos que resulten más interesantes.

Se partirá de una base de datos con imágenes reducidas a 64x64, que fue utilizada para la competición sobre detección de nódulos de pulmón Luna2016[1]. Se diseñará y entrenará una red convolucional profunda, y se realizará un estudio de optimización de resultados y tiempos de ejecución

Para ello se estudiará la influencia en los resultados del modelo de elementos tales como el tipo de funciones de activación, algoritmos de optimización y regularizadores. Además, se modificará su estructura añadiendo distintas capas, como Dropout o Batch Normalization con el objetivo de reducir el sobreajuste.

Tras optimizar los resultados en este modelo de prueba, se realizarán varias comparativas de rendimiento, minimizando el tiempo de ejecución para las distintas tecnologías seleccionadas, y, tratando de obtener el mejor sistema para nuestro problema: la detección automática de síntomas de cáncer de pulmón en imágenes médicas.

Una vez completada esta tarea, se utilizará una base de datos de radiografías más realista, procedente del National Institute of Health, para comprobar si los parámetros obtenidos anteriormente también ofrecen buenos resultados en nuestro nuevo modelo.

Una vez encontrados los valores para los distintos hiperparámetros que ofrezcan los mejores resultados posibles, se realizarán pruebas con un conjunto independiente de radiografías reales procedentes de hospitales españoles, con el objetivo de comprobar la capacidad de generalización del modelo obtenido anteriormente

Por último, se implementará en Dash una aplicación web basada en el apoyo al diagnóstico médico. Esta aplicación será capaz de obtener predicciones a partir de las radiografías introducidas. Además de estas predicciones, se mostrará al usuario un *heatmap* de la red neuronal, es decir, aquellas partes de la imagen que están siendo utilizadas para tomar la decisión final.

[1] https://luna16.grand-challenge.org/description/

II. Justificación y objetivos

La motivación de este TFM surge de una situación que se está dando en gran cantidad de países: la sobrecarga de trabajo en muchas profesiones relacionadas con el campo de la sanidad.

En la actualidad, se están utilizando técnicas relacionadas con el análisis de imágenes [Shen et al. 2017**]** para detectar gran cantidad de enfermedades, entre ellas, las radiografías. Estas imágenes son analizadas cuidadosamente por los radiólogos y doctores, expertos en interpretar los resultados obtenidos y realizar un diagnóstico apropiado.

Se trata de una tarea complicada, que requiere unos niveles de concentración muy elevados y puede llevar una gran cantidad de tiempo. Por tanto, es necesario que los expertos que analizan las radiografías no sufran de fatiga u otros problemas comunes que puedan perjudicar su rendimiento. Sin embargo, esto no es lo habitual. Las largas jornadas de trabajo y el aumento de personas diagnosticadas con enfermedades como cáncer de pulmón, que dependen de la correcta interpretación de radiografías para su detección temprana, suponen una carga de trabajo enorme para estos especialistas [Andrew. 2003].

A pesar de las dificultades que implica el análisis de radiografías, el colegio real de radiólogos indica en uno de sus 5 principios fundamentales que deben existir radiólogos disponibles para interpretar aquellos casos de alto riesgo 24 horas al día, todos los días de la semana.

Debido a los largos procesos de formación, la exposición a tareas intensivas durante largos periodos de tiempo y los altos niveles de estrés procedentes de la responsabilidad que implican sus interpretaciones, los radiólogos son una profesión muy propensa a sufrir bajas laborales [London: Royal college of radiologists. 2008].

Este trabajo tiene como objetivo final el desarrollo de una aplicación de apoyo al diagnóstico médico para este tipo de enfermedades, centrándonos en la detección de cáncer de pulmón. Se creará una aplicación capaz de analizar las radiografías introducidas, generando las probabilidades obtenidas por un modelo desarrollado mediante técnicas de Deep Learning, y, mostrando gráficamente las partes de la imagen en las que los médicos deberían prestar especial atención.

Para ello, realizaremos un análisis inicial de los distintos frameworks y librerías de Deep Learning disponibles para entrenar nuestro modelo utilizando el lenguaje Python, seleccionando las opciones más prometedoras.

Utilizando las tecnologías seleccionadas, se crearán benchmarks analizando los resultados y tiempos de entrenamiento tanto en CPUs como en GPUs. Para esto, haremos uso de una base de datos de nódulos de pulmón, utilizada para la competición Luna Grand Challenge.

Una vez realizados estos benchmarks, procederemos a la creación y optimización de un modelo de detección de patologías de pulmón, empleando la base de datos del National Institute of Health, ChestX-Ray14 [Wang X. et al. 2017]**.** Se realizarán pruebas validando la generación de heatmaps mediante casos reales, utilizando radiografías obtenidas a lo largo del año 2018 en un hospital español.

Por último, se implementará la aplicación de apoyo médico, creando una interfaz sencilla que permita a los médicos introducir sus imágenes, consultar los resultados obtenidos por el modelo y descargar los heatmaps resultantes para su posterior análisis. Esta aplicación será desarrollada utilizando Dash.

Contenido

1 Estado del arte

1.1 Situación actual del cáncer de pulmón

En la actualidad, el cáncer es una de las enfermedades con mayor tasa de mortalidad a nivel mundial[2] [World Health Organization. 2018]. Según GLOBOCAN (GCO, Global Cancer Observatory), la herramienta de estudio y análisis de la organización mundial de la salud, aproximadamente un 13% de todas las muertes en el mundo proceden de los distintos tipos de cáncer.

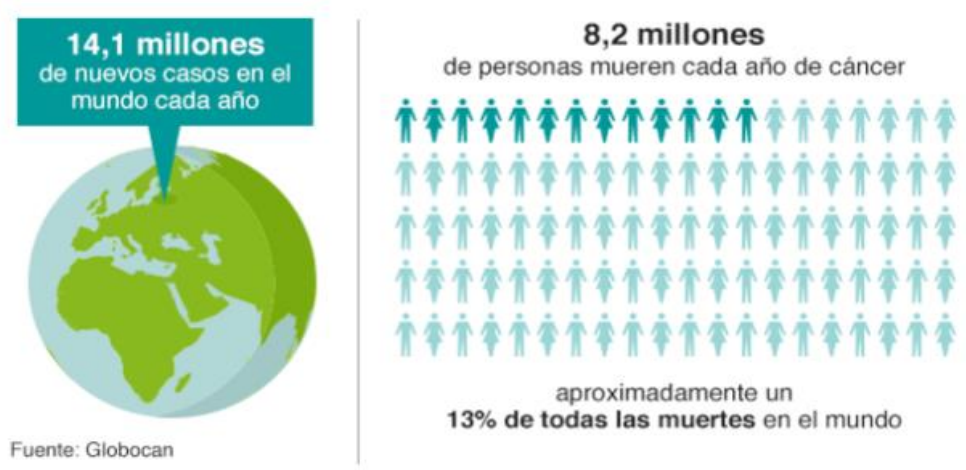


Figura 1. Estadísticas de mortalidad a nivel mundial para el año 2012 (GLOBOCAN)

Entre los tipos de cáncer más comunes se encuentra el cáncer de pulmón, tan solo superado en cuanto a porcentajes de incidencia por el cáncer de mama si tenemos en cuenta únicamente la población femenina.

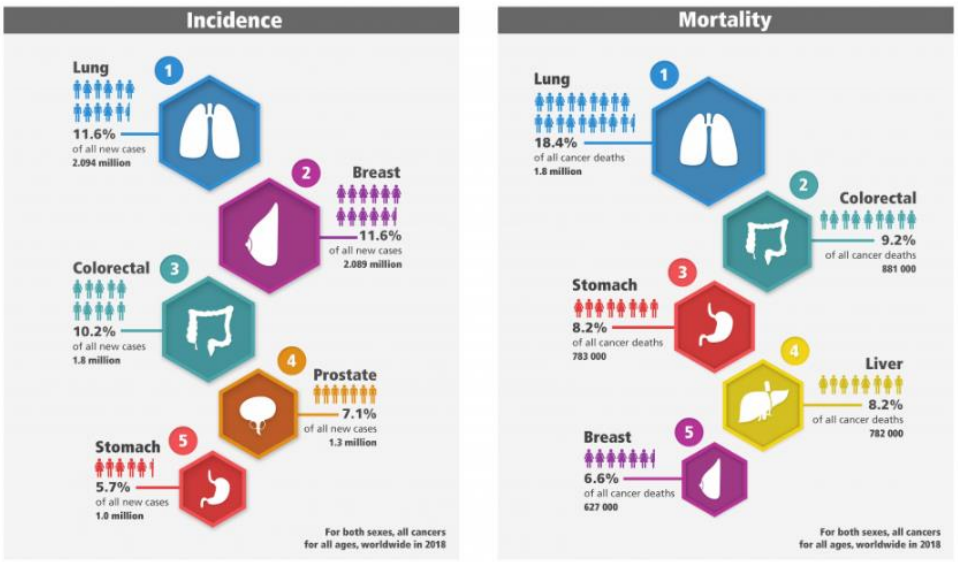


Figura 2. Tipos de cáncer con mayor mortalidad e incidencia según GLOBOCAN 2018

[2] World Health Organization, 2018 Cancer Statistics: https://gco.iarc.fr/today/data/factsheets/cancers/39-All-cancers-fact-sheet.pdf

Por desgracia, este tipo de cáncer viene acompañado de la mayor tasa de mortalidad para este tipo de enfermedad, superando la suma de los siguientes tres cánceres con mayor porcentaje de aparición (de seno, colorrectal y de próstata).

Únicamente en España, se calcula que desde el año 2015 se diagnostica esta enfermedad a más de 28.000 personas anualmente. Además, alrededor de 18.000 personas fallecen cada año debido a esta enfermedad, con un porcentaje de mortalidad bastante más elevado para los hombres en nuestro país, según los datos de la Asociación Española Contra el Cáncer (AECC)[3].

Los estudios recientes realizados por la Asociación Española Contra el Cáncer [AEEC. 2018] coinciden con las estadísticas publicadas por la Organización Mundial de la Salud, indicando que las incidencias diagnosticadas para esta enfermedad continuarán aumentando durante los próximos años. Se espera un aumento de hasta un 60% en el número de casos encontrados para el año 2040.

El cáncer de pulmón se caracteriza por su alta agresividad y su rápido crecimiento, por tanto, la detección temprana de esta enfermedad resulta crucial para aumentar las posibilidades de supervivencia. Sin embargo, el estado actual del sistema médico provoca que se cometa un mayor número de errores en el diagnóstico. Los neumólogos que deben revisar estas radiografías cuidadosamente tienen una carga de trabajo demasiado grande, provocando fatiga y pérdida de rendimiento que puede ocasionar fallos en los diagnósticos, como explica el radiólogo Luke Oakden en su artículo "*Exploring the ChestRay14 dataset*"[4].

1.2 Machine Learning en Medicina

A lo largo de las últimas décadas, el campo de la medicina ha estado apoyándose en el análisis de imágenes médicas [Litjens et al. 2017] procedentes de técnicas como las radiografías [Shen et al. 2017], resonancias magnéticas, ultrasonido o tomografías para detectar, diagnosticar y encontrar tratamientos efectivos para las distintas enfermedades.

Debido al enorme rango de patologías existentes y la gran cantidad de factores que pueden llevar a los humanos a equivocarse en su diagnóstico [Bruno M. et al. 2017], como pueden ser la fatiga en el trabajo o un simple error de juicio, los doctores e investigadores se están comenzando a apoyar cada vez más en las nuevas tecnologías, capaces de facilitar su trabajo enormemente.

A lo largo de este documento nos centraremos en aprovechar al máximo las tecnologías existentes para ayudar a los especialistas con tareas relacionadas al análisis de imágenes médicas. Para ello, debemos introducir las tecnologías principales y conceptos básicos que utilizaremos.

1.3 Deep Learning

Machine Learning es un campo de la inteligencia artificial que tiene como objetivo la creación de sistemas que posean la habilidad de aprender y mejorar a partir de experiencias previas, sin necesidad de ser programados explícitamente. Este proceso de aprendizaje se basa en la observación de una gran cantidad de datos, instrucciones o experiencias previas de manera que se puedan localizar patrones y similaridades que puedan ser utilizadas en casos futuros [Brownlee. 2019].

[3] https://www.aecc.es/es/todo-sobre-cancer/tipos-cancer/cancer-pulmon/evolucion-cancer-pulmon
[4] https://lukeoakdenrayner.wordpress.com/2017/12/18/the-chestxray14-dataset-problems/

Deep Learning es un subcampo de Machine Learning centrado en la estructura y el funcionamiento del cerebro. Según Andrew Ng., miembro del equipo de Google Brain (equipo de investigación de Google centrado en el campo de la inteligencia artificial), define Deep Learning como el uso de simulaciones del cerebro para:

- Hacer que los algoritmos de aprendizajes sean más fáciles de usar
- Conseguir grandes avances en el campo de la IA
- Acercar los equipos informáticos a la racionalidad humana

Esta técnica se beneficia enormemente de los avances en capacidad de computación y obtención de una extensa cantidad de datos, que permiten el entrenamiento de sistemas cada vez más complejos.

A diferencia de los anteriores sistemas, Deep Learning consigue obtener las características principales a partir de los datos introducidos [Brownlee. 2019, Wang. 2019], sin necesidad de la extracción de las mismas por parte de expertos en la materia. Esta característica, conocida como "Automatic feature extraction", permite que equipos no expertos en la materia estudiada puedan realizar sus pruebas e investigaciones.

Como se ha comentado anteriormente, este campo está inspirado profundamente por la arquitectura del cerebro humano. En él, se encuentran alrededor de 10^{11} neuronas, cada una de las cuales tiene cerca de 100.000 sinapsis, es decir, conexiones con otras neuronas que tienen un peso asociado. Mediante este peso, se mide la importancia de la información transmitida por cada neurona, obteniendo un impulso final de salida [Hartung. 2016]. Además, se ha demostrado que cada parte de nuestro cerebro contiene neuronas especializadas en distintos tipos de tareas.

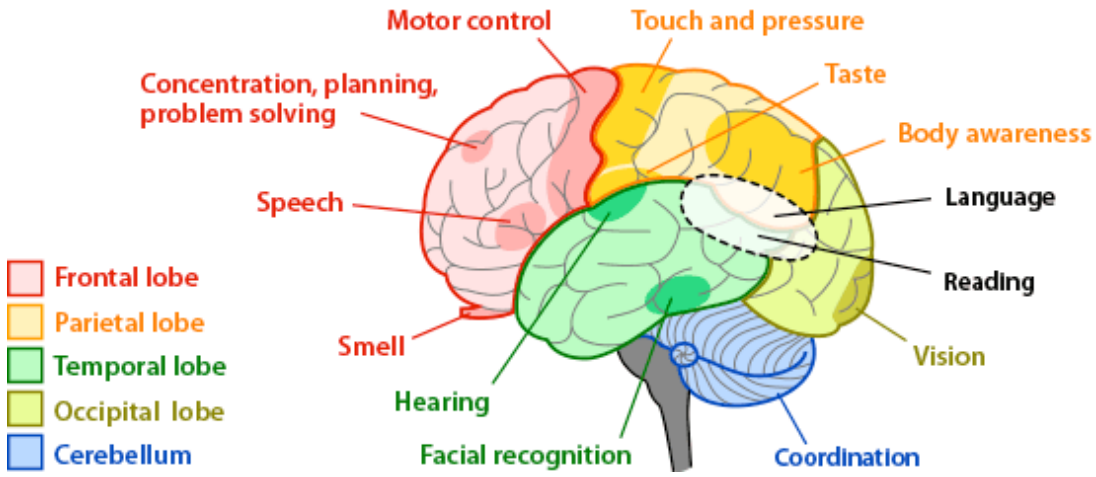


Figura 3. Distribución funcional del cerebro humano

Los sistemas actuales de Deep Learning se basan en neuronas artificiales que imitan este comportamiento. Estas neuronas procesan la información recibida y producen una salida. Como se puede observar en la Figura 4, en primer lugar, se realiza una suma pesada de los valores recibidos. Tras esto, se pasa el resultado de la suma por una función no lineal conocida como función de activación para obtener el valor de salida de la neurona.

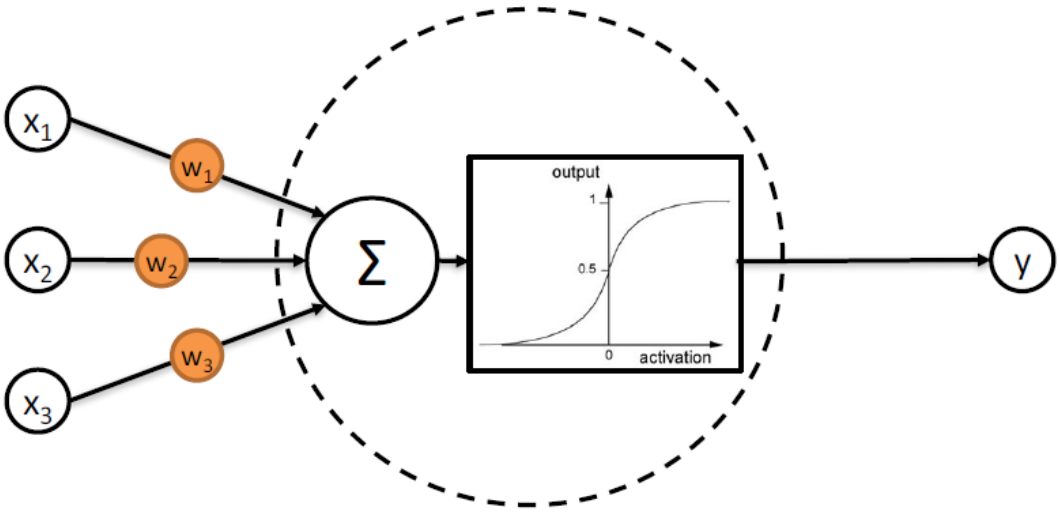


Figura 4. Estructura básica de una neurona artificial con activación softmax

Estas neuronas se combinan en estructuras jerárquicas compuestas por capas, conocidas como redes neuronales artificiales [Hartung, 2016]. Estas estructuras están compuestas de conjuntos de neuronas, comunicadas entre sí, con el objetivo de transformar la información de entrada de manera progresiva hasta obtener representaciones cada vez más abstractas que permitan alcanzar el objetivo del sistema (en nuestro caso, detectar si existen hallazgos de enfermedades en imágenes médicas).

A continuación, nos centraremos en explicar el funcionamiento de estos sistemas apoyándonos en dos tecnologías:

- Los perceptrones multicapa, popularizados a finales de los 80 y utilizados ampliamente hasta el descubrimiento de tecnologías que ofrecen mejores resultados.
- Las redes convolucionales, utilizadas ampliamente para resolver problemas en la actualidad.

1.3.1 Perceptrón multicapa (MLP)

El perceptrón multicapa (Multi Layer Perceptron) es una estructura que consiste en una capa de entrada [Razzak et al. 2018], seguida de una o varias capas ocultas encargadas de realizar transformaciones de los datos iniciales. Los resultados obtenidos en estas capas están conectados con una capa de salida, que produce el output final del sistema. El flujo de información siempre recorre la red de izquierda a derecha. El proceso de entrenamiento de un perceptrón multicapa consiste en los siguientes pasos:

1. Se inicializan aleatoriamente los pesos
2. Se propaga la entrada hacia adelante
3. Se calcula el error cometido y se vuelve atrás
4. Se adaptan los pesos iniciales
5. Se vuelve al paso 2 hasta que se obtienen los valores deseados

El principal problema de estos sistemas consiste en que los perceptrones multicapa no son capaces de extrapolar correctamente. Es decir, si no se realiza un entrenamiento correcto de la red o la cantidad de iteraciones (épocas) no es suficiente, se pueden obtener salidas imprecisas.

Además, estas redes dependen de un gran número de capas para alcanzar potencias suficientes. Un gran número de capas ocultas implicará que las primeras capas del sistema tendrán un impacto mucho menor, debido a que los algoritmos de propagación reducirán el error obtenido en su camino hacia el principio de la red.

Sin embargo, no todo son desventajas en estos sistemas. Los MLP cuentan con un gran número de pros. Entre ellos, se trata de sistemas tolerantes a fallos, fácilmente extensibles y capaces de aproximar cualquier función con el grado de precisión deseado (aproximador universal). Estas ventajas serán aprovechadas por tecnologías posteriores, como las redes convolucionales profundas o CNNs.

1.3.2 Redes Convolucionales Profundas

A diferencia de los MLP, las redes convolucionales se entrenarán de manera inteligente, eligiendo los pesos iniciales y las funciones de activación cuidadosamente con el objetivo de obtener los mejores resultados, y no de forma aleatoria como en el pasado.

La arquitectura de estas redes es la misma que la utilizada en los MLP, sin embargo, el funcionamiento varía de manera considerable. En las redes convolucionales [Freidman. 2017], se trata de extraer características o "*features*" de las imágenes iniciales, teniendo en cuenta la estructura espacial de la imagen (sistemas utilizados para procesar imágenes).

Esto quiere decir que no importa en qué lugar de la imagen inicial se reconozca un patrón característico. Si se detecta una estructura, la red aprenderá a detectar dicha estructura en cualquier otro lugar de la imagen inicial (y del resto de imágenes). Esta característica se basa en el principio de permanencia espacial.

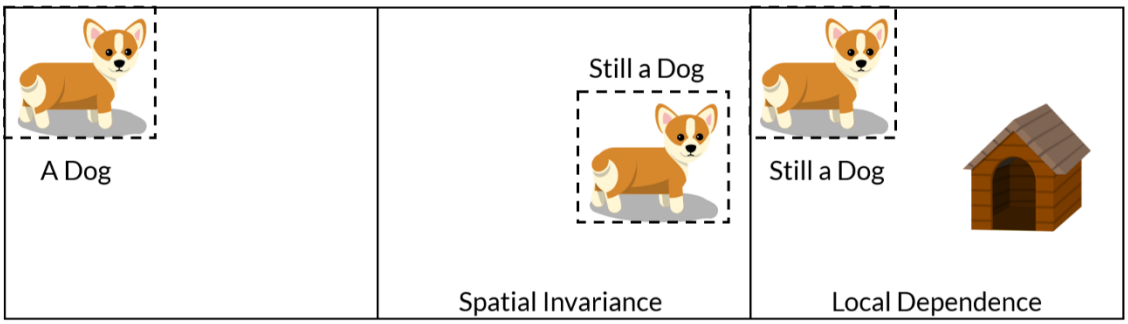


Figura 5. Permanencia espacial y dependencia local

Además, estos sistemas serán capaces de extraer la información relevante, pudiendo reconocer patrones a través de cambios en la iluminación, rotaciones, aumentos o disminuciones de tamaño y otras transformaciones básicas.

Para conseguir este objetivo, las redes convolucionales utilizan una estructura formada por la combinación de capas convolucionales seguidas de capas "*pooling*". Finalmente, se completa la arquitectura con capas densas al igual que en los sistemas anteriores (capas de salida).

1.3.2.1 Capas convolucionales

Este tipo de capa recibe su nombre de la operación conocida como convolución [Freidman. 2017]. En esta operación, la imagen de entrada en forma matricial (una matriz en caso de imágenes blanco y negro o tres matrices representando cada color en el caso de RGB) es filtrada utilizando una máscara conocida como filtro convolutivo.

Esta máscara viene acompañada de una matriz de pesos cuadrada conocida como "*kernel*", típicamente de tamaño mucho más pequeño que la imagen de entrada. Utilizando los pesos del kernel, se recorre la imagen de entrada, realizando operaciones matemáticas (productos escalares) para generar una nueva matriz de salida. El movimiento del kernel se realiza de izquierda a derecha, de arriba abajo hasta procesar todas las posiciones posibles.

En los sistemas actuales, se utilizan un gran número de filtros convolucionales, capaces de detectar elementos de la imagen como bordes, esquinas o detectar movimiento en caso de procesamiento de vídeo.

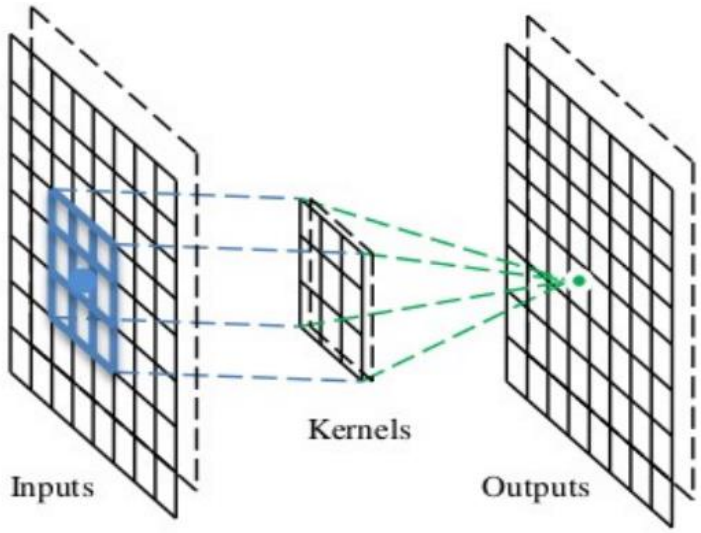


Figura 6. Comportamiento de una operación convolucional

1.3.2.2 Subsampling

Debido a la gran cantidad de operaciones realizadas, las capas convolucionales aumentan de manera considerable el número de neuronas. En una imagen en blanco y negro de 32x32, si decidiésemos utilizar 16 filtros convolucionales, obtendríamos 16.384 neuronas de salida tras realizar las operaciones descritas anteriormente.

Por tanto, si decidiésemos introducir una nueva capa convolucional a partir de la salida de nuestra capa anterior, el número de neuronas aumentaría dramáticamente, forzando a nuestro sistema a disponer de una capacidad de procesamiento inmensa. Para evitar esto, se realiza un proceso conocido como *subsampling* [Saha. 2018] a partir de las salidas resultantes. Este proceso tiene como objetivo disminuir el tamaño de nuestras imágenes filtradas sin eliminar ninguna de las características más importantes detectadas por cada uno de los filtros convolucionales utilizados.

Existen varios tipos de *subsampling*. En este documento nos centraremos en el tipo más utilizado en redes actuales (utilizado en todas nuestras arquitecturas), conocidas como capas "*Pooling*".

Las capas de tipo *pooling* tienen como objetivo reducir la dimensionalidad de las salidas de las capas convolucionales, conocidas como "Convolved features", sin eliminar las características principales obtenidas. Estas características deben ser invariables, permitiendo al sistema detectar la entidad que representan a través de cambios en la posición, rotación, iluminación y demás modificaciones.

Para mantener estas características, existen dos opciones distintas. En la Figura 7, podemos observar los resultados que se obtendrían utilizando capas Max-pooling y Average-pooling de tamaño 2x2 sobre una imagen de tamaño 5x5.

Como podemos observar, las capas Max-Pooling toman el mayor valor encontrado en el conjunto de celdas (de tamaño 2x2 en este caso). Este tipo de cada será el que utilizaremos para nuestras arquitecturas a lo largo de este documento.

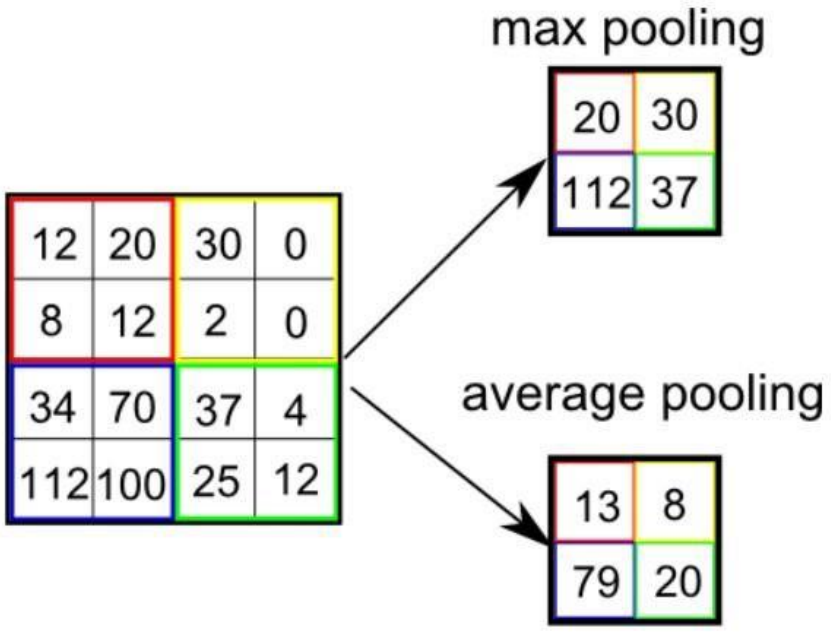


Figura 7. [Saha 2018], Resultados obtenidos utilizando capas Max y Average pooling de tamaño 2x2

Por otra parte, las capas Average-Pooling calculan el valor medio de todas las celdas que se tienen en cuenta en cada momento. Al igual que las capas convolucionales, las capas de tipo *pooling* recorren las matrices iniciales de izquierda a derecha y desde arriba hacia abajo.

1.3.2.3 Reducción del sobreajuste

El sobreajuste u "*overfitting*" es una característica no deseable en cualquier sistema de Deep Learning. Se puede decir que, cuando un sistema tiene un alto nivel de sobreajuste, las características que está aprendiendo están fuertemente correlacionadas con los datos introducidos.

Cuanto mayor sea el overfitting, menor capacidad tendrá nuestro modelo para generalizar, es decir, ser capaz de realizar predicciones utilizando datos completamente nuevos a partir de las características extraídas de los datos a partir de los que ha aprendido.

Por suerte, existen multitud de métodos capaces de reducir el sobreajuste en nuestros modelos. En este apartado, nos centraremos en las capas Dropout, Batch Normalization como métodos para reducir el sobreajuste.

Figura 8. Overfitting, underfitting y situación deseada para las predicciones de un sistema

1.3.2.3.1 Capas Dropout

Las capas Dropout [Srivastava et al. 2017] tratan de reducir el sobreajuste de una red neuronal mediante la incapacitación de múltiples neuronas, seleccionadas aleatoriamente. En una capa Dropout con un coeficiente de 0.5, el 50% de las neuronas se "congelarán", evitando que contribuyan propagando valores a las siguientes capas, e impidiendo la actualización de los pesos en las mismas.

Este funcionamiento cobra sentido gracias a una característica de las redes neuronales conocida como especialización. A medida que el sistema aprende, surgen grupos de neuronas cuyos pesos tienden a estabilizarse, especializándose en la detección de un conjunto determinado de características. Las neuronas adyacentes pueden llegar a depender completamente de esta especialización, pudiendo resultar en un modelo que se ajusta demasiado a los datos de entrenamiento.

Cuando se congela una gran cantidad de neuronas asociadas a una tarea específica, otras neuronas deberán compensar esta carga de trabajo, resultando en el aprendizaje de múltiples conjuntos de características. Esto produce que la red no sea tan sensible a los pesos de ciertos conjuntos de neuronas, provocando que se generalice mejor y se reduzca el sobreajuste.

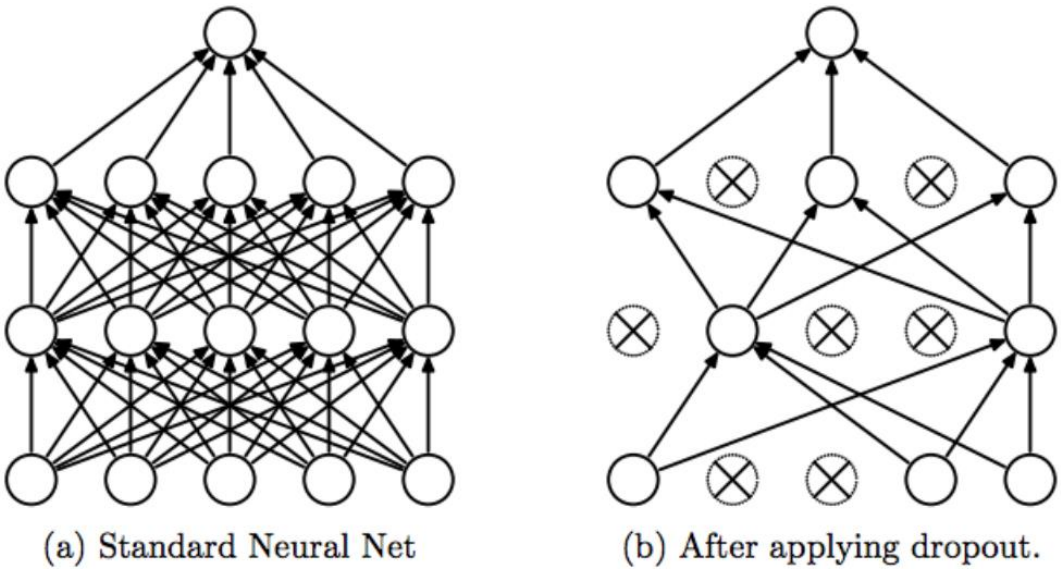


Figura 9. [Maklin 2019], Visualización del comportamiento de las capas Dropout

1.3.2.3.2 Capas Batch Normalization

Las capas Batch Normalization surgen en el año 2015, con el paper "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", creado por los

investigadores Sergey Ioffe y Christian Szegedy. En él, se propone un nuevo sistema capaz de obtener la misma precisión que las tecnologías más novedosas del momento, en un número de épocas de entrenamiento 14 veces menor.

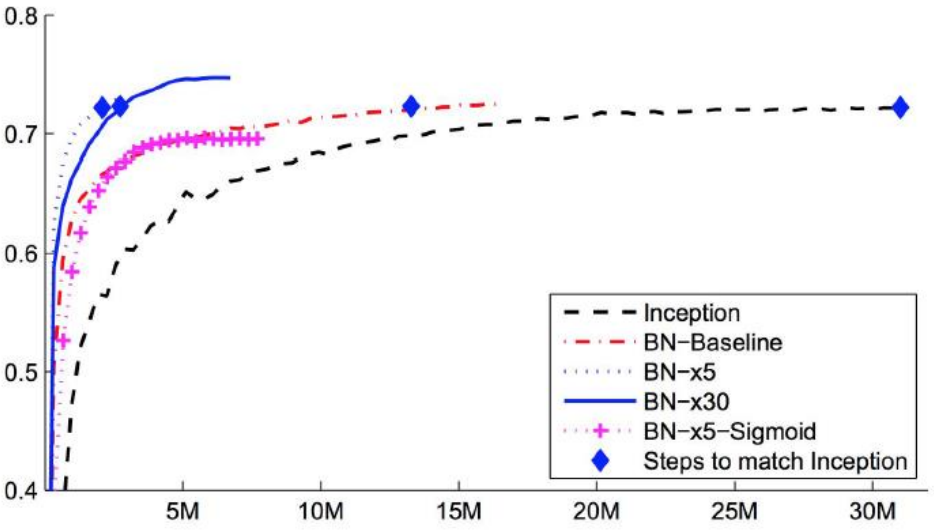


Figura 10. **[Sergey Ioffe, Christian Szegedy, 2015]** Reducción del número de épocas necesario utilizando capas Batch Normalization en el modelo Inception

El funcionamiento de estas capas se basa en el concepto de desplazamiento de la covarianza, es decir, una variación de la distribución inicial de nuestro sistema. Esta variación se produce debido al ajuste de los distintos parámetros en cada época del entrenamiento, ralentizando enormemente este proceso e impidiendo el uso de mayores tasas de aprendizaje.

Para resolver este problema, se propone incorporar un paso intermedio, normalizando la red y equilibrando la media y la varianza. Las ventajas principales, entre muchas otras, que se obtienen mediante el uso de estas innovadoras capas son las siguientes:

- Posibilidad de usar mayores tasas de aprendizaje (necesidad de menor número de épocas).
- Reducción del sobreajuste ya que estas capas actúan como regularizadores. En ocasiones, pueden eliminar la necesidad de capas Dropout.

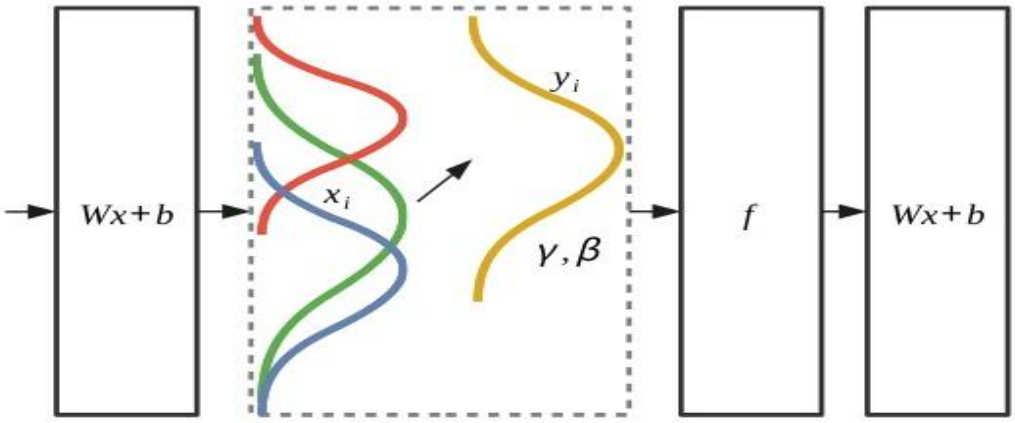


Figura 11. Esquema del comportamiento de las capas Batch Normalization.

Fuente: 岳華 杜, “Taiwan center for Disease Control”

1.3.2.4 *Estructura típica de una red convolucional*

En el año 2010, se lanza la competición de reconocimiento de imágenes de gran escala de ImageNet[5] (ILSVRC). El objetivo de esta competición consiste en detectar objetos, animales y demás estructuras que se pueden observar a simple vista en el mundo real.

Apenas dos años después de la aparición de esta base de datos de gran escala, se hace público el artículo "ImageNet Classification with Deep Convolutional Neural Networks" [Krizhevsky et al. 2012]. En él, se propone una arquitectura de red convolucional profunda conocida posteriormente como AlexNet, debido al nombre de su creador.

Esta arquitectura supuso un paso gigantesco en el campo del Deep Learning, consiguiendo superar la precisión promedio obtenida por un humano al enfrentarse a este problema de clasificación. La precisión de AlexNet era de un 84,7%, números muy superiores a los ofrecidos por los sistemas existentes en aquella época. El sistema más cercano a AlexNet había sido capaz de obtener una precisión de tan solo 73,8%.

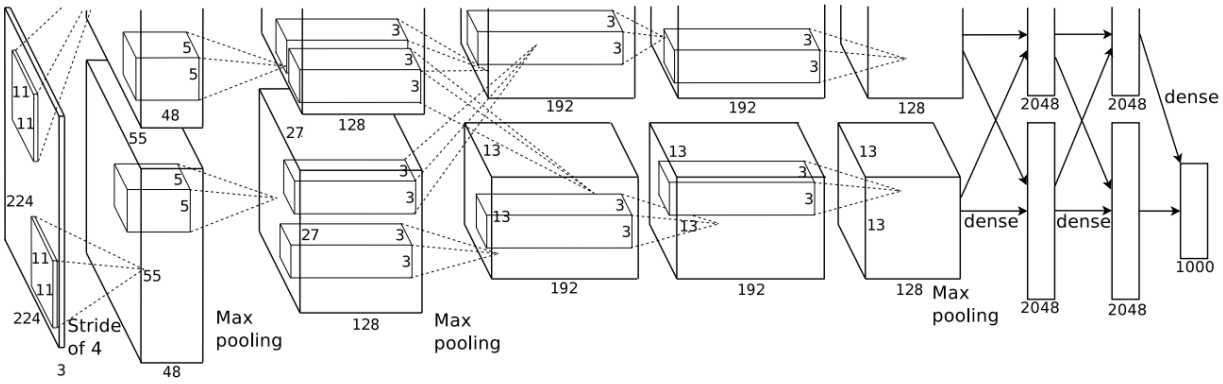


Figura 12. Arquitectura de AlexNet. Fuente: [Krizhevsky et al. 2012]

AlexNet utilizaba una arquitectura bastante mayor que los anteriores sistemas basados en CNNs. Esta arquitectura, consistente de 5 capas convolucionales, seguidas de capas de *subsampling,* y, finalmente, tres capas densas utilizadas para clasificar las 1000 clases existentes en el conjunto de ImageNet, utilizaba alrededor de 60 millones de parámetros y 650.000 neuronas. El entrenamiento del sistema fue realizado en poco más de cinco días, utilizando dos de las tarjetas gráficas más potentes del momento (GTX 580).

En la actualidad, se han creado sistemas capaces de producir resultados mucho mejores que AlexNet, basándose en nuevos tipos de arquitectura como los bloques Residual o Inception.

Sin embargo, el estudio y optimización de arquitecturas para mejorar las redes convolucionales profundas aún está en pleno crecimiento, centrándose principalmente en la mejora de resultados y el aprovechamiento de la creciente capacidad de computación.

1.4 Análisis de frameworks existentes

Antes de comenzar el desarrollo y las pruebas, se debe realizar un estudio de los *frameworks* y librerías de AI más populares e interesantes del momento, con el objetivo de seleccionar los más adecuados para nuestro caso de estudio: análisis de imágenes médicas.

[5] Base de datos ImageNet: http://www.image-net.org/

1.4.1 Theano

Gran cantidad de los investigadores en el campo de Deep Learning han utilizado o siguen utilizando Theano. Esto se debe a las numerosas ventajas que ofrece, siendo uno de los primeros *frameworks* Python en ofrecer soporte para CPU y GPU. Al tratarse de uno de los principales *frameworks* durante años, existe gran cantidad de documentación y ejemplos reales de uso de esta tecnología.

Además, se han implementado varias librerías que permiten abstraerse de la programación a bajo nivel, como podrían ser Keras o Lasagne.

Sin embargo, este *framework* ha dejado de ser mantenido desde septiembre de 2018, apostando por nuevas librerías como TensorFlow. Estas nuevas librerías no sufren gran parte de los principales problemas que nos encontrábamos con Theano, como los largos tiempos de compilación cuando se utilizan modelos de gran tamaño.

Ventajas:

+ Soporte CPU y GPU

+ Documentación y ejemplos

+ Programación mediante Python y Numpy

+ Librerías de alto nivel

Inconvenientes:

- Una única GPU

- Mucho tiempo de compilación en modelos grandes

- No preparado para modelos ya entrenados

- Gran cantidad de bugs

1.4.2 Tensorflow

Tensorflow nace como la herramienta creada por Google para remplazar a Theano. Se trata de dos tecnologías muy similares, ya que ambas están basadas en una API Python que funciona sobre un motor escrito en C y C++ para mejorar su rendimiento.

Algunas de las principales ventajas de TensorFlow con respecto a otras tecnologías son la gran cantidad de herramientas de visualización del aprendizaje, como TensorBoard, además de traer grandes mejoras en los tiempos de compilación (aunque sigue siendo más lento que otras tecnologías como PyTorch) y soporte para sistemas multi-GPU.

Ventajas:

+ Soporte CPU y multi-GPU

+ Gran comunidad, foros de ayuda y documentación

+ Uso de tensores

+ Herramientas de visualización

Inconvenientes:

- Más lento que otras tecnologías
- Grafo de computación lento (generado en Python puro)
- No hay soporte comercial

1.4.3 CNTK

CNTK es el *framework* de Deep Learning ofrecido por Microsoft. Ofrece soporte para utilizar y combinar gran cantidad de los modelos más relevantes actualmente, como podrían ser DNNs (Deconvolutional Neural Network), CNNs (Convolutional Neural Networks) o redes neuronales recurrentes (tanto RNNs como LSTMs).

Al igual que los anteriores *frameworks*, ofrece una API Python pese a estar desarrollado íntegramente en C++. Esta tecnología puede ofrecer mejores tiempos en comparación con Tensorflow, especialmente para modelos pesados en sistemas con múltiples GPUs.

CNTK utiliza una licencia MIT (licencia permisiva) para algunas partes de su código. Sin embargo, no permite el uso comercial para su innovador método de cálculo del gradiente (One-bit SGD).

Ventajas:

+ Soporte CPU y multi-GPU

+ Facilidad de implementación para modelos populares

+ Cálculo eficiente del gradiente mediante One-bit SGD

Inconvenientes:

- Menor documentación que otras tecnologías
- Distintas licencias (no permisivas en parte)
- Poco soporte de herramientas auxiliares (visualización)

1.4.4 Keras

Considerada por muchos la mejor librería de Deep Learning creada hasta la fecha. Esta librería, creada por Francois Chollet, ofrece un alto nivel de abstracción para los programadores. Permite la programación de operaciones complejas en cuestión de unas pocas líneas.

Además, ofrece soporte para múltiples *backends*, como son TensorFlow, Theano, DeepLearning4J y CNTK, permitiendo cambiar la tecnología utilizada de manera sencilla.

Una de las muchas ventajas del uso de esta librería es la gran comunidad que la consolida, creando gran cantidad de guías y tutoriales, que se complementan con la gran cantidad de documentación ya existente en la página oficial de la librería.

Ventajas:

+ Programación en alto nivel

+ Varias tecnologías como *backend*

+ Excelente documentación y comunidad

1.4.5 PyTorch

Al igual que Tensorflow, PyTorch es un *framework* de Deep Learning basado en tensores (estructuras de datos similares a los *arrays* de Numpy pero más eficientes). Está desarrollado por Facebook, y, pese a su escasa documentación oficial (clara falta de información para algunos aspectos), cuenta con una comunidad en constante crecimiento que ofrece ayuda mediante el foro encontrado en su plataforma.

Se trata de una tecnología en crecimiento debido a la gran cantidad de posibilidades que ofrece a los programadores. La creación de las redes es de más bajo nivel que otras tecnologías, pero ofrece ventajas como la modificación de la arquitectura en *runtime* o el cálculo automático del gradiente mediante la herramienta Autograd.

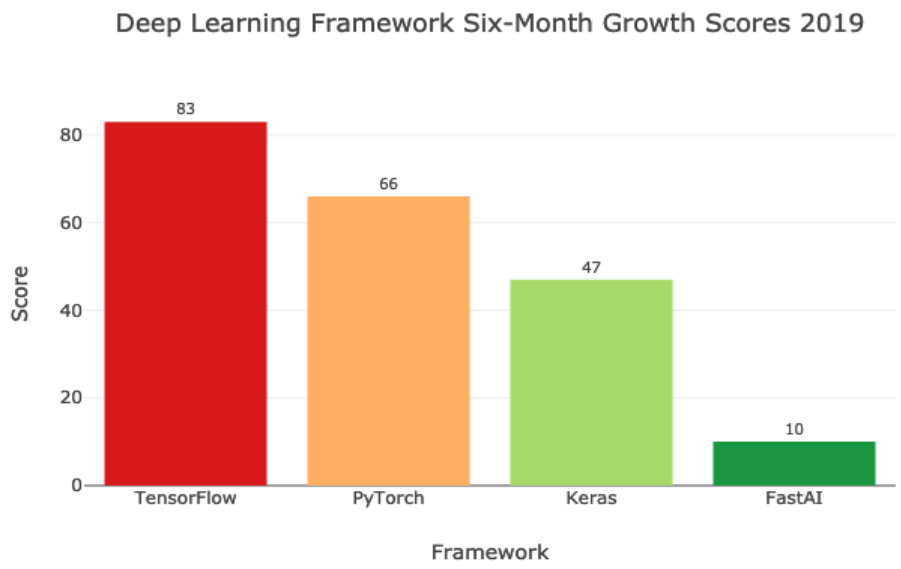


Figura 30. Crecimiento herramientas de DeepLearning en 2019

Ventajas:

+ Optimizado para GPU y TPU

+ Foro de ayuda y comunidad en crecimiento

+ Uso de tensores

+ Gran velocidad

+ Flexibilidad para el programador

+ Gran cantidad de modelos ya entrenados

Inconvenientes:

- Escasa documentación

- Programación de bajo nivel

- No hay soporte comercial

1.4.6 Caffe

Caffe es una librería basada en la implementación de Matlab para las redes convolucionales. Se trata de una librería enfocada al trabajo con imágenes (clasificación mediante redes convolucionales), ya que no contempla otras facetas del Deep Learning como podrían ser el procesamiento de texto, sonido o TSDs (datos ordenados temporalmente).

Ventajas:

+ Soporte CPU y GPU

+ Enfocado a trabajar con imágenes

Inconvenientes:

- Menor documentación que otras tecnologías
- Desarrollo lento y comunidad poco activa
- Única GPU
- Código poco extensible
- Necesidad de escribir en C++ y CUDA
- No cuenta con soporte comercial

1.4.7 Tensorflow 2.0

A principio de junio de 2019, sale de su fase Alpha la segunda versión de TensorFlow. Pese a estar en Beta, ya se han anunciado una gran cantidad de mejoras con respecto a la primera versión de TensorFlow.

Gran parte de estas mejoras acercan el framework a PyTorch, permitiendo customizar los modelos y bucles de entrenamiento mediante una API flexible que permite control completo al programador. Además, se añade soporte para TPUs y optimizan las herramientas existentes de visualización (mejora del *workflow* y ajuste de parámetros para TensorBoard).

Ventajas:

+ Optimizado para GPU y TPU

+ En desarrollo, aún se esperan más mejoras

+ Flexibilidad para el programador

+ Script de migración TF1 -> TF2

+ Recomienda su uso conjunto con Keras

Inconvenientes:

- Versión Beta
- Programación de más bajo nivel

2 Metodología

2.1 Bases de datos utilizadas

2.1.1 Lung Node Malignancy

La primera base de datos que vamos a utilizar para este proyecto procede de Kaggle, una plataforma que celebra competiciones relacionadas con el campo de la inteligencia artificial. En ellas, los participantes tratan de obtener el mayor porcentaje de acierto para los distintos problemas propuestos, con la opción de ganar grandes premios si terminan en las primeras posiciones.

Esta base de datos, llamada Lung Node Malignancy[6], cuenta con 6691 imágenes etiquetadas. Cada una de estas imágenes contiene un fragmento de una radiografía de pulmón, con una resolución de 64x64.

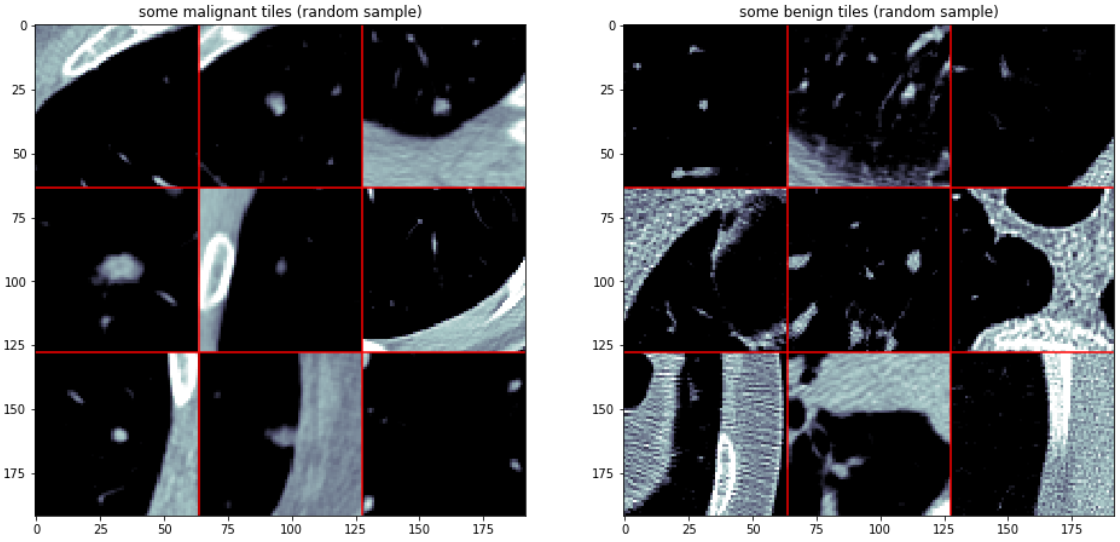


Figura 13. Ejemplo de nódulos malignos (izquierda) y benignos (derecha)

En ella, se seleccionan imágenes utilizadas en la competición Lung Nodule Analysis, celebrada en 2016. Todas estas imágenes contienen nódulos de pulmón, seleccionados por un conjunto de radiólogos expertos (al menos 3 de 4 radiólogos deben confirmar que se trata de un nódulo de pulmón válido para que la imagen sea aceptada).

2.1.2 ChestX-Ray14

La base de datos que vamos a utilizar en este apartado es una extensión de la primera base de datos de imágenes médica a gran escala (escala de hospital), conocida como ChestX-Ray8.

En ella se encuentran 112.120 radiografías de tórax frontales procedentes de 30.805 pacientes únicos. Estas radiografías están etiquetadas mediante catorce etiquetas, que representan las posibles patologías que pueden ocurrir en los pacientes. El grupo de posibles hallazgos (*findings*) contemplados en nuestro dataset es el siguiente:

[6] https://www.kaggle.com/kmader/lungnodemalignancy

1, Atelectasia

2, Cardiomegalia

3, Efusión

4, Infiltración

5, Masa pulmonar

6, Nódulo de pulmón

7, Neumonía

8, Neumotórax

9, Consolidation

10, Edema

11, Emphysema

12, Fibrosis

13, Mesotelioma pleural

14, Hernia

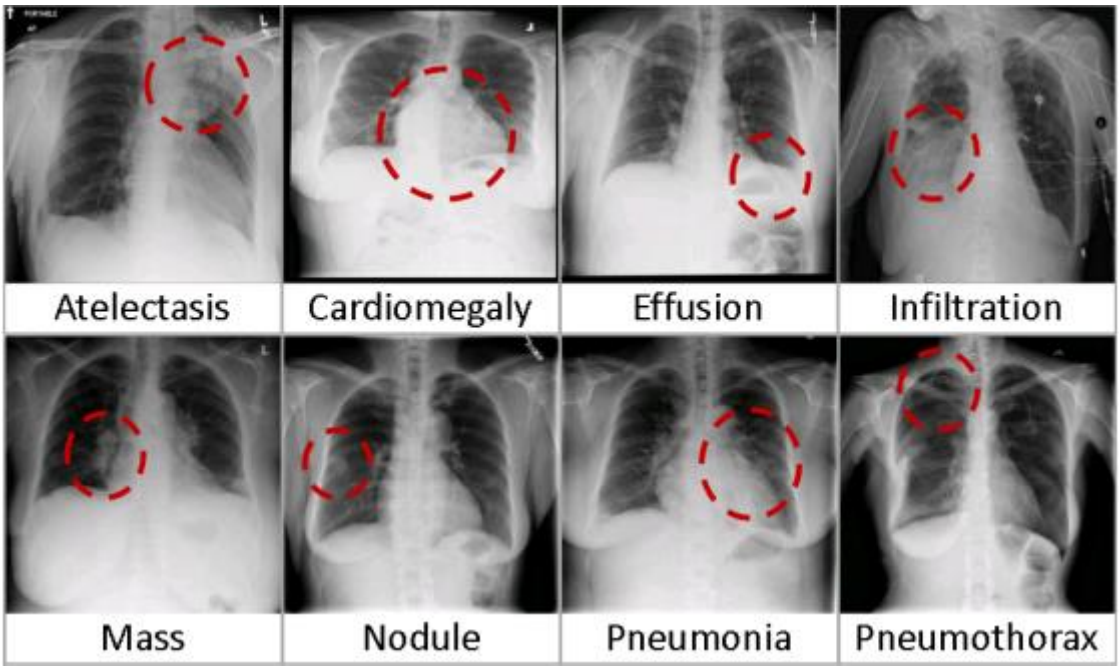


Figura 14. Ejemplos de radiografías con distintas patologías

El proceso de etiquetado se ha llevado a cabo utilizando técnicas de procesamiento de lenguaje natural (NLP), a partir de las descripciones obtenidas por los médicos y radiólogos expertos del hospital en el que se han recolectado las imágenes. En el artículo introductorio del conjunto de datos, Xiaosong Wang reporta que se ha obtenido una precisión en el etiquetado de las radiografías de aproximadamente un 92%. En el Anexo 2, se puede observar gráficamente la distribución y las estadísticas de ocurrencia simultánea de los distintos *findings* etiquetados.

2.1.2.1 Descripción de los datos

Como hemos mencionado anteriormente, la base de datos ChestX-Ray14 contiene más de 110.000 imágenes. Para el desarrollo de este TFM, se han podido utilizar tres equipos distintos:

- El ordenador portátil utilizado en la anterior sección, con un Intel Core i7 (2,6GHz) y una Nvidia Geforce 960M
- Un ordenador de sobremesa con un Intel Core i7 (3,6GHz) y una Nvidia Geforce 2080Ti
- Un servidor proporcionado por la empresa Cognodata que dispone de 12 CPUs

Debido a las características de los equipos disponibles, se ha decidido utilizar una muestra reducida de la base de datos inicial, conteniendo 5606 imágenes procedentes de 4230 pacientes únicos. De esta manera evitaremos que los tiempos de entrenamiento del conjunto no se disparen para poder realizar las pruebas necesarias.

Además, el tamaño original de las imágenes no podrá ser utilizado (por motivos de tiempo de entrenamiento). La resolución utilizada para el entrenamiento será de 128x128.

Como se puede observar en la Figura 32, los datos están bastante balanceados, obteniendo un accuracy de tan solo un 54% en caso de utilizar un clasificador que prediga siempre la clase más probable.

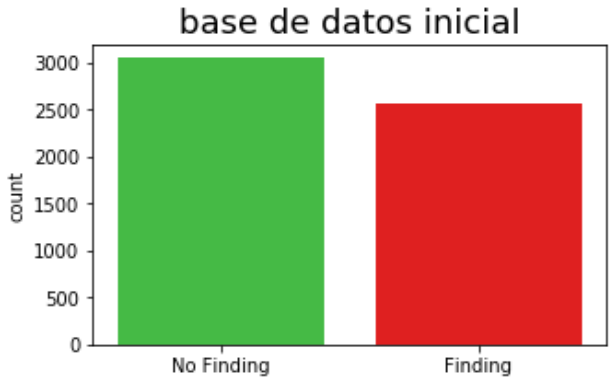


Figura 15. Distribución de las imágenes en la muestra seleccionada

En cuanto a las edades de nuestro conjunto de datos, una gran parte de las radiografías provienen de pacientes entre 40 y 70 años, como se puede observar en el histograma a continuación. Las edades de los pacientes están comprendidas entre uno y 94 años.

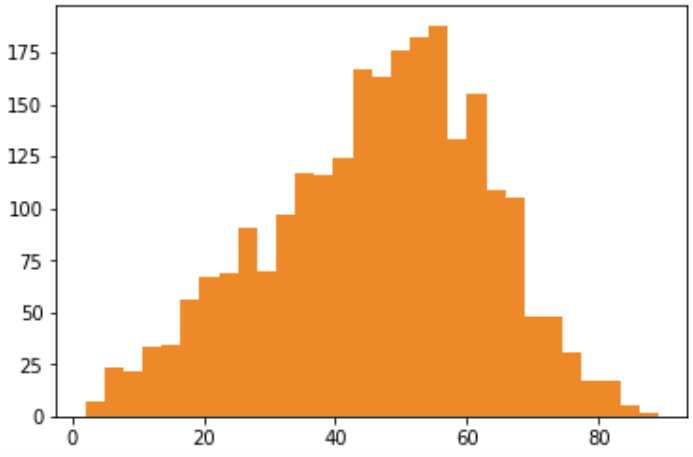


Figura 16. Histograma de la distribución por edades en nuestra muestra de datos

2.1.3 Imágenes reales de un hospital español

El último conjunto de datos que vamos a utilizar servirá para validar los resultados obtenidos a partir de la optimización de nuestros modelos obtenidos a partir de las primeras dos bases de datos, comprobando que se generaliza de manera correcta.

Se trata de un conjunto de 16 radiografías reales, obtenidas a lo largo del año 2018 en un hospital de Vigo (el nombre del hospital no se puede revelar en este documento por motivos de confidencialidad).

Al igual que las imágenes de la base de datos ChestX-Ray14, estas radiografías han pasado por un proceso de normalización y rescalado para cumplir con las características de entrada de los modelos obtenidos, reduciendo su tamaño a 128x128.

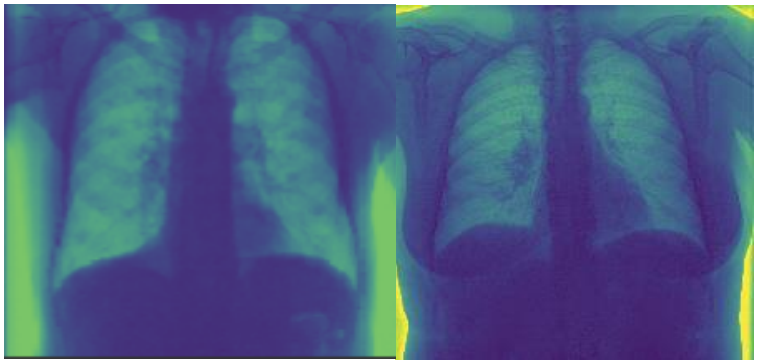


Figura 17. Ejemplos de radiografías procedentes del hospital español

2.2 Preprocesado de las imágenes

El siguiente paso que debemos realizar una vez seleccionadas las imágenes a utilizar para el entrenamiento de los distintos modelos es el preprocesado y limpieza del conjunto. En esta fase se llevarán a cabo tres tareas principales:

- Eliminar las imágenes inválidas
- Normalizar el brillo
- Utilizar técnicas de data augmentation

2.2.1 Limpieza de imágenes inválidas

Tras haber entrenado varios modelos utilizando la base de datos ChestX-Ray14, durante la fase de análisis de resultados nos hemos dado cuenta de que nuestra red neuronal se estaba fijando en partes de algunas imágenes que estaban completamente en negro.

Esto se debe a que algunas radiografías del conjunto de datos seleccionado estaban mal recortadas o escaneadas de manera incorrecta, dejando huecos negros alrededor de las imágenes, que, posteriormente serían interpretados incorrectamente por el sistema.

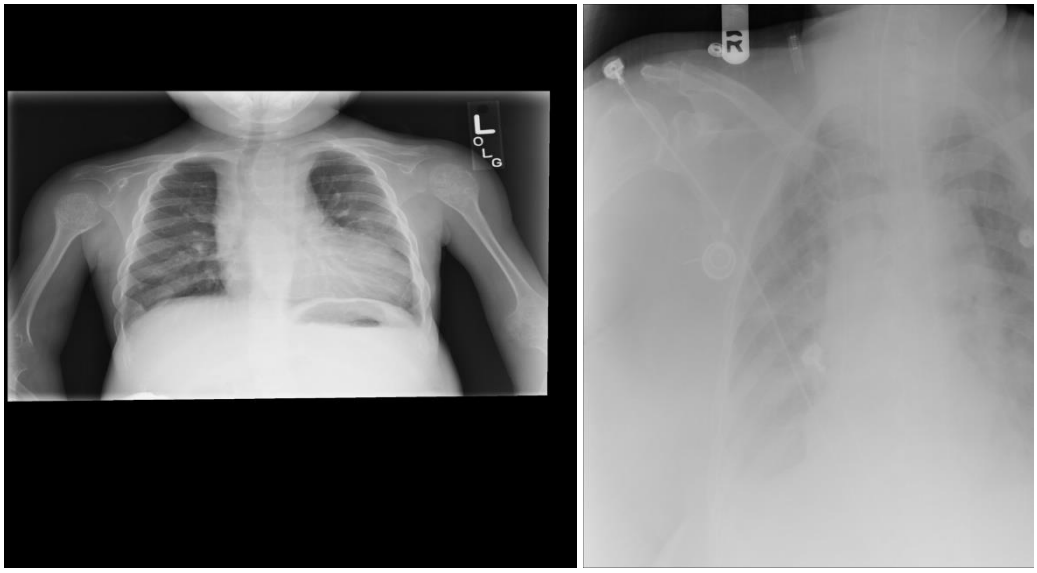


Figura 18. Ejemplo de imágenes recortadas incorrectamente o con espacios innecesarios

Antes de eliminar estas imágenes, debemos comprobar que la distribución de las clases que habíamos planteado inicialmente no varía de manera significativa (se mantiene un cierto balanceo entre las imágenes con y sin *findings*[7]).

Para ello, visualizamos la distribución en las imágenes erróneas, comprobando que se corresponde con la del conjunto inicial. De esta forma, podemos asegurar que no se eliminarán muchas más imágenes de una clase determinada, provocando desbalanceo. Una vez comprobado que este es el caso, podemos eliminar las imágenes detectadas como erróneas.

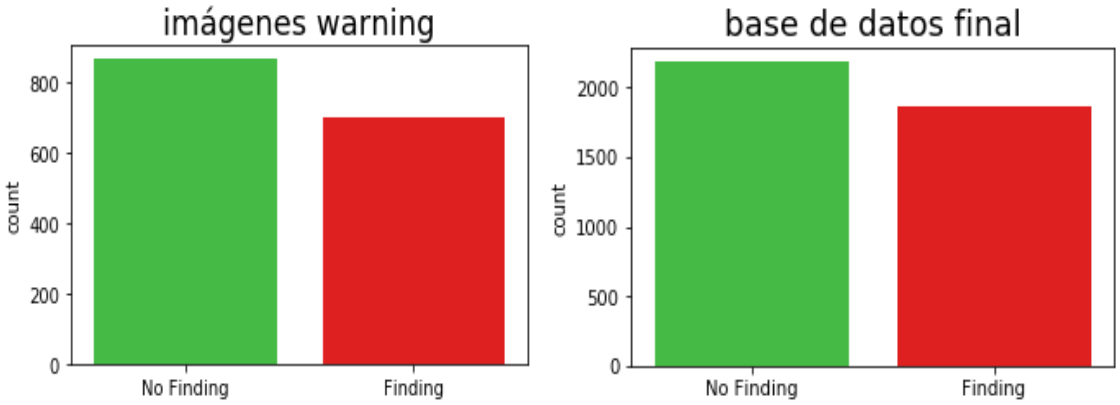


Figura 19. Distribución de clases en las imágenes erróneas y en el conjunto final

2.2.2 Normalización del brillo en las imágenes

Al tratarse de imágenes escaneadas a partir de radiografías tomadas a lo largo de mucho tiempo, las características de estas podrían variar considerablemente de una muestra a otra. Por tanto, debemos llevar a cabo una normalización del brillo en las imágenes utilizadas, tratando de reducir las diferencias causadas por el método de escaneado o las distintas máquinas de rayos X utilizadas.

[7] Detección de una patología en la imagen seleccionada

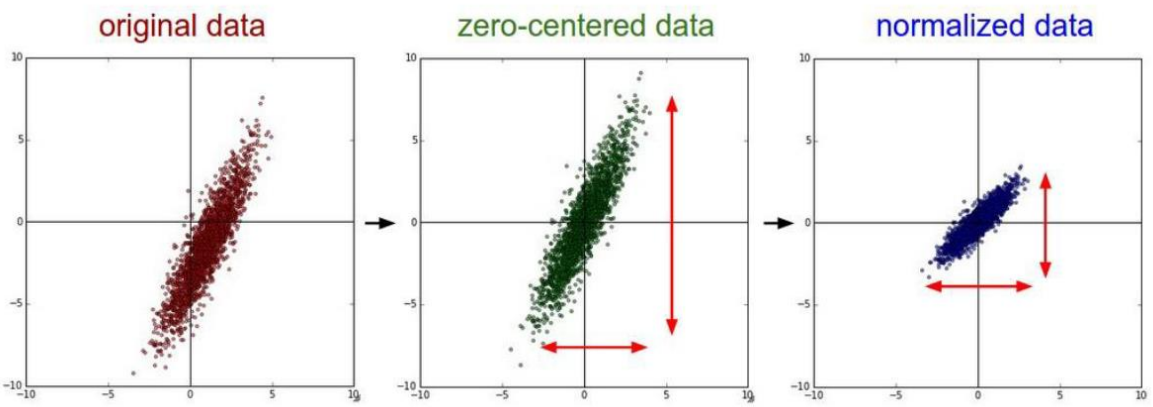


Figura 20. Representación del efecto de la normalización. Fuente: Stanford CS class CS231n.

Mediante la normalización de las imágenes a utilizar, se transforman los valores del brillo en nuestras imágenes a una escala común, sin modificar la diferencia en el rango global de valores. Este es un proceso muy recomendable en cualquier CNN que utilice valores numéricos (como la intensidad del brillo).

Figura 21. Ejemplo de imágenes tras la normalización del brillo

2.2.3 Data augmentation

Al utilizar un conjunto de datos limitado debido a la capacidad de procesamiento disponible, puede producirse sobreajuste, es decir, el modelo aprenderá a reconocer una imagen determinada pero no será capaz de generalizar correctamente para cualquier imagen introducida.

La técnica de Data augmentation consiste en generar nuevas imágenes realistas a partir de las imágenes existentes mediante transformaciones semialeatorias. En Keras, se puede llevar a cabo utilizando la clase ImageDataGenerator, que ofrece una gran cantidad de parámetros configurables, a partir de los que el sistema generará nuevas imágenes.

En su libro “Deep Learning with Python”, François Chollet, ingeniero de Google y creador de Keras, demuestra cómo configurar de manera sencilla un sistema de Data Augmentation. A partir de este ejemplo inicial, configuramos los distintos parámetros a utilizar, reduciendo de manera considerable las rotaciones, *zooms* y distorsiones que se realizan en un comienzo, para obtener imágenes de radiografías realistas.

Figura 22. Ejemplo de imágenes generadas mediante Data Augmentation

Gracias a esta técnica, utilizaremos imágenes muy similares a las originales, sin embargo, se introducen pequeños cambios, evitando que el modelo se corresponda más de lo necesario a las imágenes utilizadas para el entrenamiento, es decir, se trata de favorecer la generalización del modelo y reducir el sobreajuste.

2.3 Métricas utilizadas

En nuestro caso, el riesgo de obtener un falso positivo (persona no enferma a la que se le diagnostica una enfermedad) es muy alto como en cualquier problema de diagnóstico médico. Aun así, estas falsas predicciones no llegarían a causar daños en la mayoría de los casos, ya que la enfermedad sería fácilmente descartada mediante pruebas adicionales y el seguimiento médico que se llevaría a cabo.

		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

Figura 23. Estructura de una matriz de confusión

Sin embargo, si se produce un falso negativo (persona enferma a la que no se le diagnostica la enfermedad) se podría estar poniendo la vida de los pacientes en juego.

Las características de las enfermedades diagnosticadas serían la necesidad de detección temprana, ya que, en fases iniciales existe un alto porcentaje de que sea posible ofrecer un tratamiento u operación capaz de curar al paciente. En caso de encontrarnos con un paciente en estado avanzado, este porcentaje disminuye dramáticamente, pudiendo llegar a tratarse de una enfermedad incurable o con unas posibilidades mínimas.

Debido a esto, se prestará especial atención al valor del *recall*, cuya fórmula podemos ver a continuación:

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

Se implementa una función personalizada para calcular esta métrica y utilizarla en el futuro tanto en Keras como en PyTorch, sustituyendo el uso previo del accuracy (porcentaje de acierto):

```
#Created recall metric for keras
def recall(y_true, y_pred):
    '''Calculates the recall, a metric for multi-label classification of
    how many relevant items are selected.
    '''
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + K.epsilon())
    return recall
```

Figura 24. Implementación de la función de cálculo de *recall* en Keras

Además de la precisión o *accuracy* y el *recall*, utilizaremos a lo largo de este documento el área bajo la curva ROC como métrica (AUC). Se trata de una medida que representa gráficamente el rendimiento de nuestro modelo a la hora de diferencia entre ambas clases. Cuanto mayor sea el área bajo la curva de nuestro modelo, mejor podrá diferenciar los casos con o sin presencia de hallazgos (*findings*).

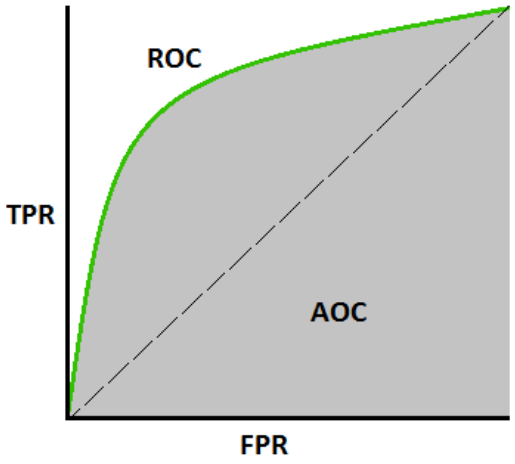


Figura 25. Curva ROC: [Narkhede. 2018] Verdaderos positivos frente a falsos positivos

2.4 Funciones de activación

La función de activación, o función de transferencia de una red neuronal es el encargado de determinar el output de nuestro sistema, es decir, devolver un resultado comprendido en un determinado rango de valores, dependiendo de la función utilizada. Se trata de funciones ascendentes, continuas y diferenciables.

En la actualidad, la función de activación RELU es la más utilizada en la mayoría de las redes convolucionales, y, en muchos casos ofrece los mejores resultados. Este tipo de función de activación cobró importancia tras la publicación del artículo introductorio de AlexNet y la observación de sus excelentes resultados. Sin embargo, resulta interesante fijarnos en sus alternativas más populares: las funciones *sigmoid* y *tanh* (sigmoide escalada):

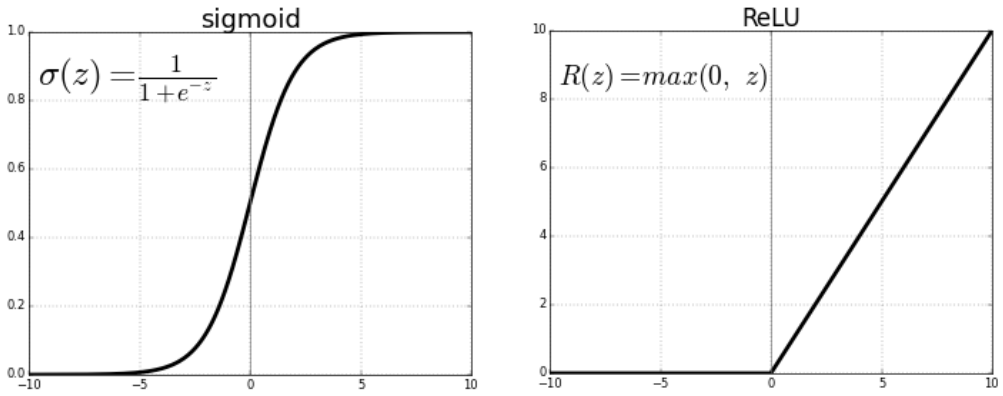


Figura 26. Funcionamiento de las activaciones *sigmoid* y ReLU

Debido a sus características, la función *sigmoid* podría ser interesante para problemas de clasificación con dos clases. Como se puede observar, entre los valores 5 y -5 del eje X, se produce una gran variación en el eje Y. Esto provoca que la función tenga tendencia a devolver valores de la Y muy cercanos al 0 o al 1, es decir, los valores deseados para un clasificador.

Al igual que la función *sigmoid,* la función *tanh*, que toma su nombre de Tangente Hiperbólica, es una función de tipo sigmoidal, es decir, en forma de letra S. La principal ventaja de esta función es que los inputs negativos se relacionarán con valores negativos, manteniendo la tendencia de la función *sigmoid* de devolver valores cercanos a 0 para entradas cercanas a 0.

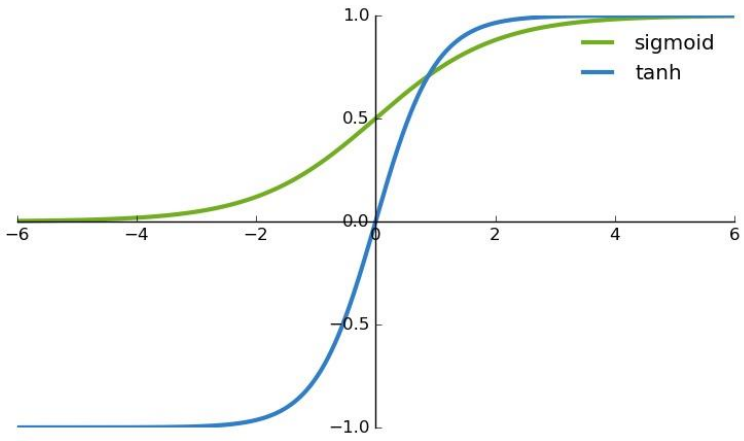


Figura 27. Representación de las funciones *sigmoid* (azul) y *tanh (naranja)*

2.5 Transfer Learning

El entrenamiento de una red convolucional consiste en encontrar los valores adecuados para una serie de parámetros, de manera que, en la última capa de nuestra red, se active un conjunto determinado de neuronas capaz de predecir la clase correcta.

Este proceso se puede realizar de manera sencilla para proyectos pequeños, sin embargo, cuando contamos con modelos muy complejos o conjuntos de datos de gran tamaño, el tiempo requerido y la capacidad de computación necesaria se convierten en dos grandes problemas.

Transfer Learning es una técnica utilizada frecuentemente para solucionar estos problemas. Esta técnica consiste en reutilizar información de modelos entrenados anteriormente, por lo general en problemas de clasificación de imágenes de gran escala (datasets de gran tamaño, ej: ImageNet). Se basa en que un modelo entrenado con un dataset suficientemente grande e informativo, será capaz de aprender y generalizar conceptos del mundo real. Estos conceptos podrán ser reutilizados para problemas en los que las clases a predecir no tienen por qué coincidir con aquellas utilizadas en el problema original.

Por ello, podemos considerar Transfer Learning como una técnica para optimizar nuestros sistemas, obteniendo mejor tiempo y rendimiento si se utiliza en las condiciones apropiadas. En el caso ideal, el uso de Transfer Learning ofrecería las siguientes ventajas simultáneamente:

- Mejora del valor inicial: El modelo comenzará realizando mejores predicciones que un modelo aleatorio.
- Mayor pendiente: El modelo será capaz de llegar a los mejores resultados de manera más rápida.
- Mejor rendimiento: Se obtendrán mejores resultados que en un modelo tradicional.

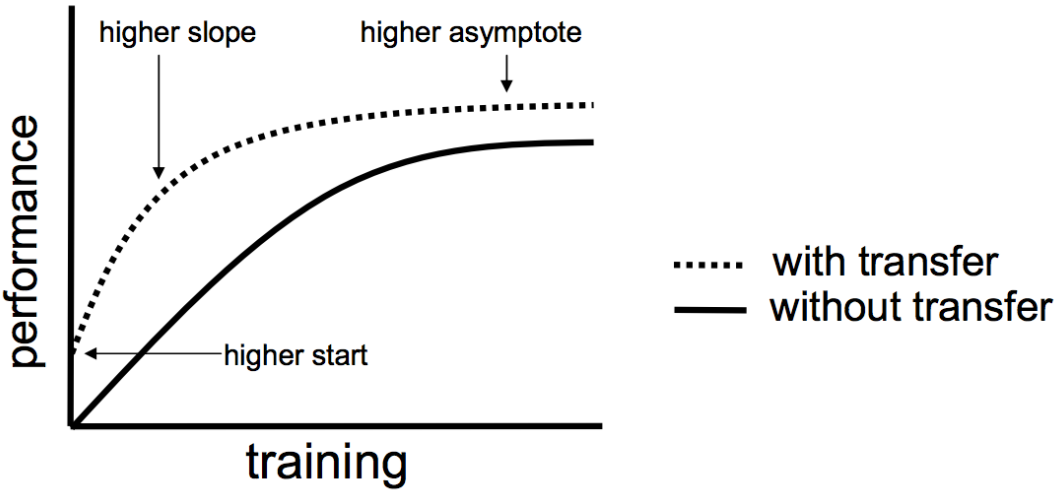


Figura 28. [Yosinski 2014] Ventajas de Transfer Learning sobre el aprendizaje

El motivo por el que esta técnica funciona tan bien está relacionado con el funcionamiento de las redes convolucionales de reconocimiento de imágenes. Mediante visualización del comportamiento de estas redes (que estudiaremos más adelante), se puede comprobar qué partes de la imagen se activan tras cada capa:

- Las primeras capas son capaces de reconocer aspectos básicos como colores o líneas horizontales y verticales.

- Las siguientes capas se fijan en formas creadas a partir de estas líneas y colores.
- Tras esto, se comienzan a reconocer texturas y partes de objetos, como podrían ser los ojos de un animal, la nariz de una persona, etc.
- Por último, la activación de las últimas capas reconoce objetos completos, como podrían ser un perro o un gato.

Gracias a esta estructura, los sistemas utilizan características de las primeras capas de modelos previos, entrenando únicamente unas cuantas capas densas para adaptar estas características a nuestro problema. Esto agiliza enormemente el proceso de entrenamiento y permite reducir la cantidad de información necesaria de manera significativa cuando se trata de problemas similares.

Sin embargo, las ganancias obtenidas mediante Transfer Learning están fuertemente relacionadas con las características de las imágenes iniciales que se pueden reutilizar para nuestro problema. Por ejemplo, se podrían obtener muy buenos resultados reutilizando un sistema de clasificación de animales (perros, gatos, pájaros) para un sistema que clasifique distintas razas de un determinado perro. En cambio, reutilizar un sistema de clasificación de objetos del mundo cotidiano para un problema de diagnóstico médico es complicado, ya que las imágenes apenas guardan semejanza entre sí.

En la actualidad, los sistemas más utilizados para Transfer Learning utilizan el conjunto de datos ImageNet, que contiene imágenes etiquetadas para objetos y elementos del mundo real (animales, objetos cotidianos…). Varios de estos modelos vienen incluidos en Keras, como se puede observar en la documentación de keras[8], permitiendo precargar los pesos resultantes del entrenamiento en ImageNet.

La pregunta que nos debemos plantear sería: ¿resulta beneficioso extraer características del mundo natural para problemas de diagnóstico médico? Este es el tema principal del artículo "*Convolutional Neural Networks for Medical Image Analysis"* [9], que concluye que se pueden obtener ventajas a partir de Transfer Learning a pesar de tratarse de imágenes y situaciones muy poco relacionadas. Para ello, se deben seleccionar cuidadosamente qué capas y pesos se van a reutilizar, consiguiendo adaptar los modelos previos a nuestro problema.

Transfer Learning puede dividirse en dos técnicas complementarias: Feature Extraction y Fine Tuning.

En el caso de Feature Extraction, se toman las partes de la red más genéricas, obteniendo una mayor probabilidad de que se identifiquen características positivas para el nuevo problema. A partir de la base convolucional del primer modelo y nuestro conjunto de datos, se extraen los pesos que utilizaremos para nuestro nuevo clasificador.

Fine Tuning consiste en mantener una gran cantidad de capas del modelo base (el modelo antiguo a partir del que realizamos Transfer Learning) congeladas, únicamente permitiendo entrenar las capas superiores del modelo. Tras esto, se añade un conjunto de capas (nuestro nuevo modelo) sobre las capas congeladas. Este proceso permite ajustar las representaciones obtenidas por el modelo inicial, asociándolas en medida de lo posible al nuevo problema con el que estamos tratando. Un proceso estándar de Fine Tuning cuenta con los siguientes pasos:

[8] Aplicaciones, redes preentrenadas en Keras: https://keras.io/applications/
[9] Nima Tajbakhsh, Jae Y. Shin et al. https://arxiv.org/pdf/1706.00712.pdf

- Añadir tu propia red sobre el modelo preentrenado
- Congelar las capas base de la red inicial
- Entrenar únicamente tu propia red
- Descongelar las capas que queramos volver a entrenar del modelo inicial
- Entrenar estas capas juntos a la red que has añadido

2.6 Visualización del aprendizaje

En muchos casos, se interpreta el Deep Learning como una técnica de caja negra en la que nuestro sistema crea representaciones abstractas con el objetivo de resolver un problema determinado. Sin embargo, en las redes convolucionales es posible analizar el comportamiento de nuestras redes, observando de manera visual los conceptos que aprende el sistema.

En el libro Deep Learning with Python [Chollet, 2017], se cubren tres formas de análisis del comportamiento de nuestros sistemas:

- Visualización de las salidas intermedias: permite observar las transformaciones realizadas por las distintas capas de nuestro sistema.
- Visualización de los filtros convolucionales: permite entender los patrones que se están tratando de reconocer en cada filtro utilizado por la red. Se pueden diferenciar fácilmente los filtros que reconocen bordes de la imagen, líneas verticales, etc.
- Visualización de los heatmaps de activación de clase: se trata del método de visualización en el que nos centraremos en este documento. Permite observar gráficamente aquellas partes de la imagen que nuestro sistema ha utilizado para tomar la decisión entre las clases a predecir.

2.6.1 Generación de heatmaps

El conjunto de técnicas utilizado para generar los heatmaps se conoce como CAM (Class activation map). Esta técnica se basa en generar una matriz de valores asociada con el output de una clase determinada.

Para cada clase que queramos predecir, existirá una matriz que indique qué partes de la imagen inicial son más importantes a la hora de predecir esta clase. En nuestro problema esperamos que este tipo de gráficos indiquen las partes determinadas de una radiografía en las que se pueden localizar los distintos hallazgos de que queremos reconocer.

Para el desarrollo del módulo de generación de heatmaps [Chollet. 2017], se ha utilizado la función *gradients* de Keras.backend para obtener el gradiente de la clase que se quiere representar. A partir de esta información, se generarán las matrices de valores asociados a la activación de cada clase. Una vez obtenidas estas matrices, solamente tendremos que sobreponerlas con la imagen inicial, ajustando la opacidad para poder visualizar de manera sencilla las zonas en las que se está fijando nuestro sistema.

Figura 29. [Chollet. 2017] Heatmap asociado a la detección de un elefante africano

2.7 Selección de frameworks

Teniendo en cuenta las distintas ventajas que nos ofrecen los *frameworks* y librerías mencionados en el estado del arte, se ha decidido realizar el estudio utilizando Keras y PyTorch.

Keras nos permite obtener tiempos y resultados para distintas tecnologías con tan solo un par de modificaciones en sus ficheros de configuración. Dentro de las tecnologías que Keras puede utilizar como *backend*, se ha decidido descartar Theano ya que ha sido descontinuada por Google en 2017, a favor del *framework* TensorFlow.

Además, se utilizará PyTorch para obtener comparativas a partir del *framework* que ha tenido uno de los crecimientos más espectaculares a lo largo de estos últimos años.

Utilizando estos frameworks, se generarán benchmarks utilizando la base de datos Lung Node Malignancy, con el objetivo de analizar los resultados obtenidos y los tiempos de ejecución en distintas configuraciones: Utilizando únicamente CPUs y utilizando GPUs.

Finalmente, se seleccionará un framework con el que realizar el estudio final, empleando la base de datos obtenida del NIH, ChestX-Ray14.

2.8 Creación de dashboard de apoyo médico

Para el desarrollo de la aplicación, se ha decidido utilizar Dash, un framework diseñado para generar aplicaciones analíticas en Python o R. Dash está construido utilizando Flask, Plotly.js y React, y se trata de una librería Open Source, bajo la licencia MIT (licencia permisiva).

Las principales ventajas que nos han llevado a elegir Dash sobre otras alternativas han sido las siguientes:

- Facilidad de aprendizaje: Se trata de una tecnología fácil de poner en marcha. Gracias a su gran cantidad de ejemplos de uso y su comunidad activa, resulta sencillo desarrollar una aplicación en Dash.
- Aplicaciones renderizadas en el navegador: Dash facilita el despliegue de tus aplicaciones, permitiendo compartirlas mediante URLs. El hecho de que se puedan visualizar directamente en los navegadores implica que las aplicaciones son multi plataforma y podrán funcionar en dispositivos móviles.
- Componentes prediseñados: Al igual que otras tecnologías como React, Dash proporciona al programador una gran cantidad de componentes prediseñados, que facilitan enormemente el proceso de desarrollo
- Potencial crecimiento de la aplicación: Pese a que no se va a utilizar en este documento, Dash permite al programador incluir diagramas y gráficos interactivos, que podrían ser útiles en futuras versiones de la aplicación. Por ejemplo, podría crearse una gráfica interactiva que muestre el progreso realizado en cada época.

2.8.1 Dash demos

Además de las ventajas que hemos nombrado anteriormente, Dash ofrece infinitas posibilidades a la hora de crear aplicaciones interactivas, actualizando los gráficos y estadísticas en tiempo real.

En la plataforma de Dash, existe una galería abierta al público en la que se pueden observar distintos ejemplos de aplicaciones profesionales creadas utilizando esta tecnología.

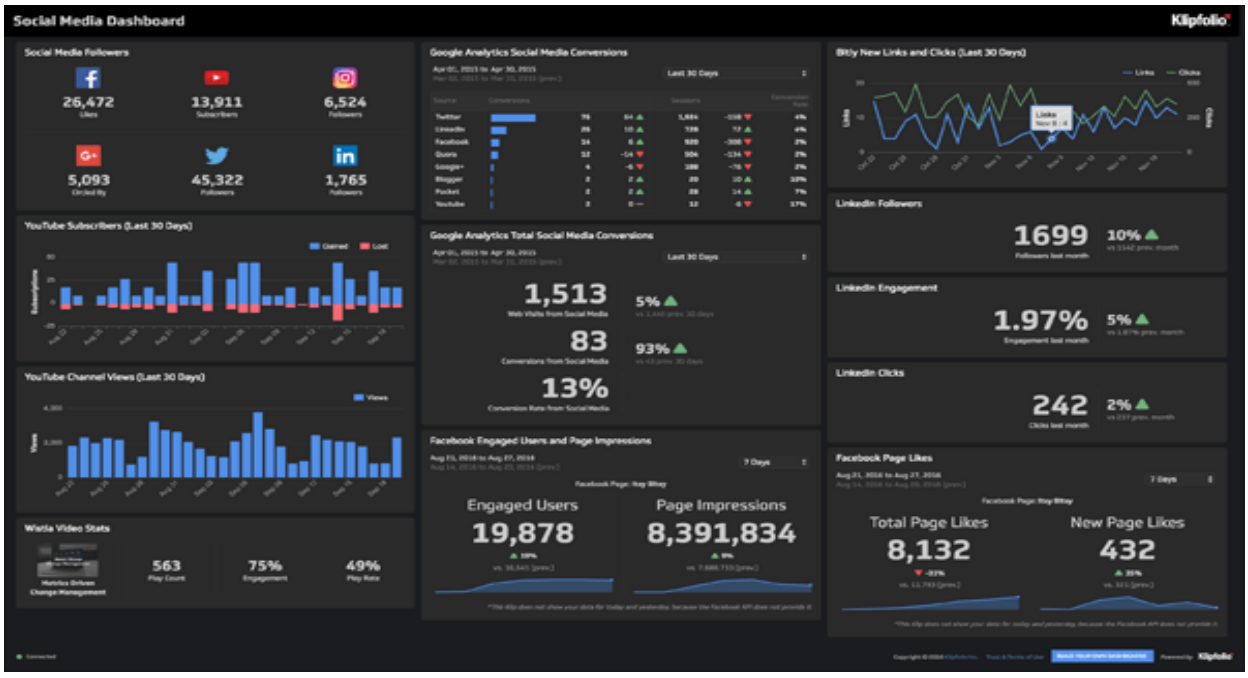


Figura 31. Dashboard actualizado en tiempo real creado mediante Dash

(https://dash-gallery.plotly.host)

3 Análisis de resultados

3.1 Problema Lung Node Malignancy

3.1.1 Estado Inicial

A continuación, vamos a tratar de realizar todas las mejoras posibles partiendo de un modelo inicial que cuenta con una configuración bastante utilizada en problemas de reconocimiento de imagen.

Se realizarán las pruebas y mejoras del modelo en Keras, utilizando el backend TensorFlow. Tras acabar de realizar las optimizaciones, se comprobará si estas modificaciones también provocan una mejora de resultados en PyTorch (debido a la distinta implementación) y se crearán benchmarks de tiempo y resultados entre las tres tecnologías seleccionadas.

3.1.2 Resultados iniciales

El modelo inicial comienza con dos capas convolucionales con activación ReLU. Tras esto, se utiliza una capa MaxPooling para extraer los *features* y una capa Flatten. Además, se añade una capa Dropout para evitar que el modelo deje de aprender y controlar el sobreajuste. Por último, se utilizan dos capas densas, con activaciones ReLU y softmax. La primera de estas capas utiliza una regularización de tipo L2, encargada de controlar el sobreajuste mediante una reducción de los pesos de salida (un valor bastante elevado para los pesos suele significar que nuestro sistema está sufriendo *overfitting*).

Inicialmente, se utiliza un optimizador RMSProp, y se computa la pérdida mediante la pérdida logística (*log loss*), una técnica utilizada comúnmente para clasificación de imágenes también conocida como pérdida multinomial o *categorical crossentropy*.

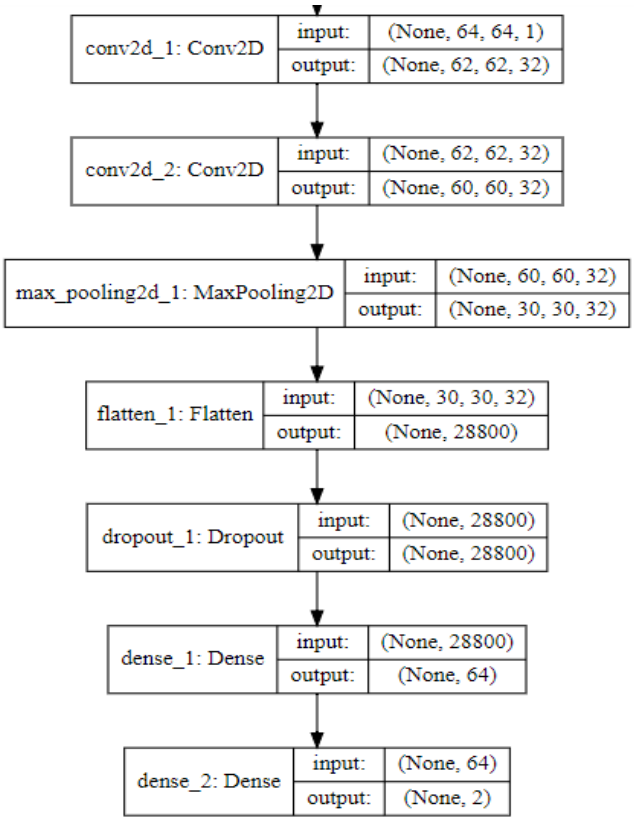


Figura 32. Visualización de las capas del modelo inicial

Los resultados obtenidos mediante este modelo se quedan bastante lejos de los valores que nos gustaría obtener para un problema de reconocimiento de imágenes médicas con este conjunto sencillo:

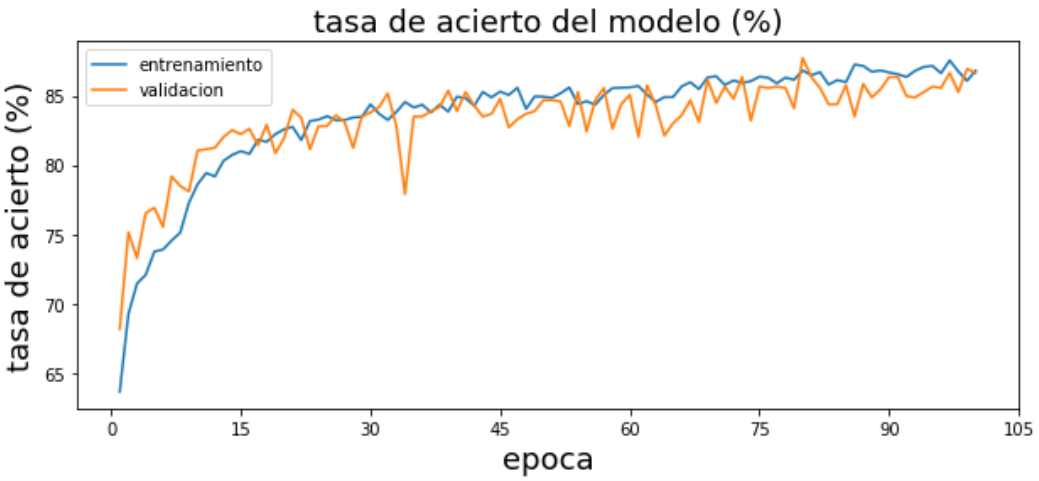


Figura 33. Resultados de entrenamiento y validación en las distintas épocas

Como se puede observar en la Figura 6., se obtienen valores bajos para la tasa de acierto (*accuracy*) al finalizar el entrenamiento durante 100 épocas. El mayor *accuracy* en test obtenido ha sido de 0,851, es decir, se ha clasificado correctamente un 85,1% del total de las imágenes. La matriz de confusión obtenida (representación gráfica de los valores predichos contra los valores reales) ha sido la siguiente:

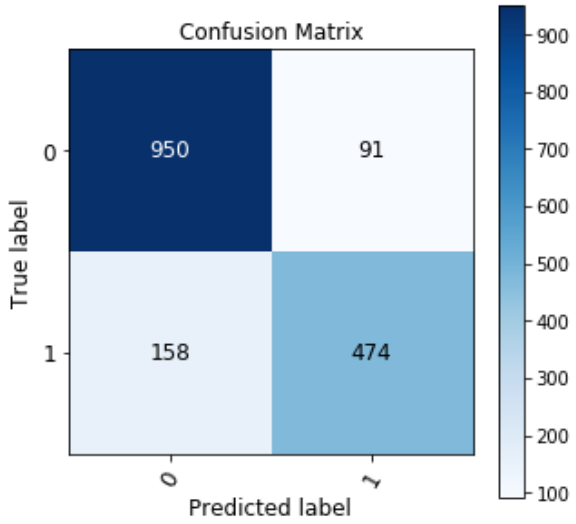


Figura 34. Matriz de confusión modelo inicial

En esta matriz se puede observar cómo se ha predicho incorrectamente que el paciente tenía cáncer en 91 radiografías. Además, se ha predicho que el paciente no padecía ninguna enfermedad en 158 casos en los que sí se trataba de cáncer de pulmón.

3.2 Pruebas y mejoras realizadas

3.2.1 Activación y optimizadores

Inicialmente, utilizamos ReLU como función de activación debido a sus numerosas ventajas descritas anteriormente. Sin embargo, puede resultar interesante comprobar si alguna de las funciones alternativas ofrece una mejora en los resultados de nuestro sistema. Las funciones utilizadas para el entrenamiento de los modelos alternativos han sido *sigmoid* y *tanh*, debido a que se trata de las siguientes funciones más utilizadas en la actualidad (después de ReLU) para problemas de clasificación binaria.

A pesar de las distintas características beneficiosas que tienen ambas funciones, tanto *sigmoid* como *tanh* empeoran considerablemente los resultados obtenidos por nuestro modelo, reduciendo la tasa de acierto considerablemente (se reduce aproximadamente un 10% en el caso de sigmoid y un 18% utilizando *tanh*) e incrementando de manera drástica el número de falsos negativos:

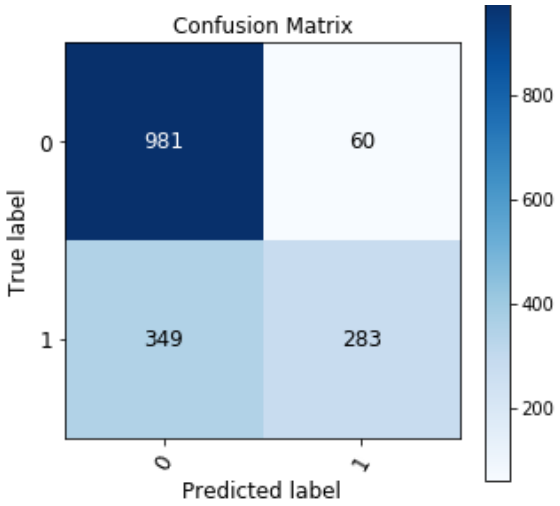


Figura 35. Matriz de confusión obtenida tras utilizar la activación *sigmoid*

Tras comprobar que no se obtienen mejoras modificando la función de activación, se tratará de mejorar los resultados modificando el algoritmo de optimización utilizado (inicialmente RMSProp):

Actualmente, se consideran tres algoritmos de optimización principales: RMSProp, SGD (Stochastic Gradient Descent) y ADAM (y sus distintas versiones modificadas). Por tanto, se realizarán pruebas utilizando tanto SGD como ADAMAX, una versión de ADAM basada en la norma de orden infinito que mejora considerablemente la estabilidad de este algoritmo.

El cambio de este algoritmo de optimización provoca un aumento considerable de la tasa de acierto tanto para SGD como para ADAMAX, como podemos comprobar en la tabla a continuación:

	RMSProp	SGD	ADAMAX
Accuracy	0,851	0,914	0,916
Recall	0,830	0,902	0,905
ROC Area	0,92	0,97	0,97

Los resultados mejoran considerablemente utilizando SGD y ADAMAX. Sin embargo, se produce un aumento del sobreajuste en la gráfica de entrenamiento. Este aumento es ligeramente menor utilizando ADAMAX. Por tanto, se ha decidido utilizar este algoritmo ya que también ofrece mejores resultados que SGD.

3.2.2 Minimización del sobreajuste: Dropout Y Batch Normalization

La mejora de los resultados obtenidos anteriormente viene acompañada por un aumento del sobreajuste, es decir, un aumento de la relación del modelo con el caso específico sobre el que se está entrenando.

Esto puede ser peligroso ya que, si un modelo se corresponde demasiado con el conjunto particular de datos utilizado, pueden producirse fallos en casos futuros con datos diferentes (no se generaliza correctamente).

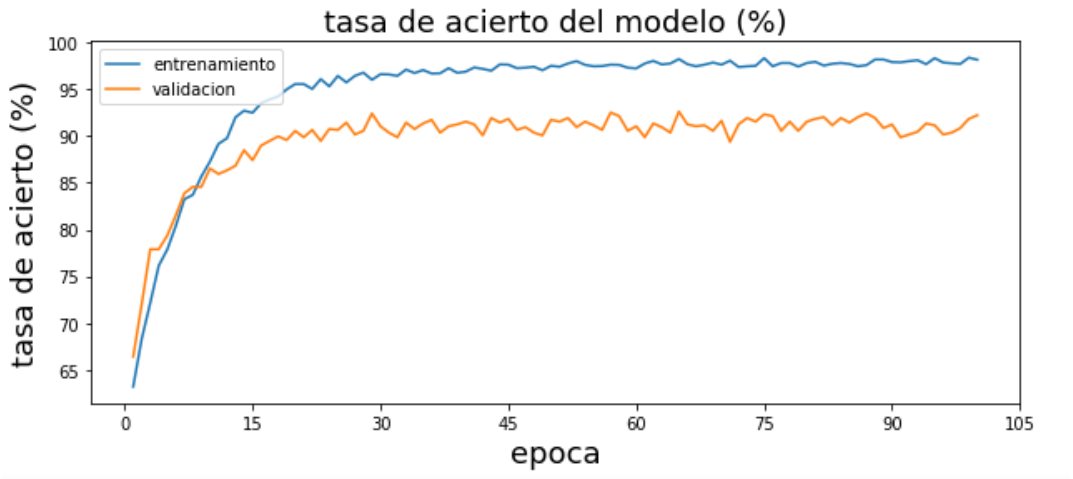


Figura 36. Overfitting introducido al pasar del optimizador RMSProp a ADAMAX

Para intentar disminuir el sobreajuste introducido (Figura 10.), trataremos de modificar el modelo, introduciendo en primer lugar capas Dropout con valores típicos para clasificación de imagen (0,5 y 0,7). Tras esto, probaremos distintos valores no tan tradicionales y sustituiremos estas capas por una alternativa que cobra bastante importancia en la actualidad, las capas Batch Normalization.

Al realizar estos cambios, nos fijaremos tanto en los resultados obtenidos (*accuracy*, *recall* y área bajo la curva ROC) como en el overfitting que se produce a lo largo del entrenamiento.

Las cuatro mejores configuraciones obtenidas en cuanto a resultados han sido las siguientes:

	Capas Dropout 0,7	Capas Dropout 0,5	Capas Dropout 0,5 y 0,3	Capas Batch Norm
Accuracy	0,915	0,917	0,937	0,939
Recall	0,90	0,912	0,925	0,921
ROC Area	0,96	0,97	0,98	0,98

Como podemos observar, se obtienen los mejores resultados utilizando capas Dropout de distintos valores (primera capa 0,5 y segunda capa 0,3) y utilizando capas Batch Normalization tras las capas convolucionales y densas en nuestra red.

Sin embargo, las configuraciones que reducen el *overfitting* de manera más significativa son aquellas que implementan capas de tipo Dropout, como se puede observar en la Figura 11.

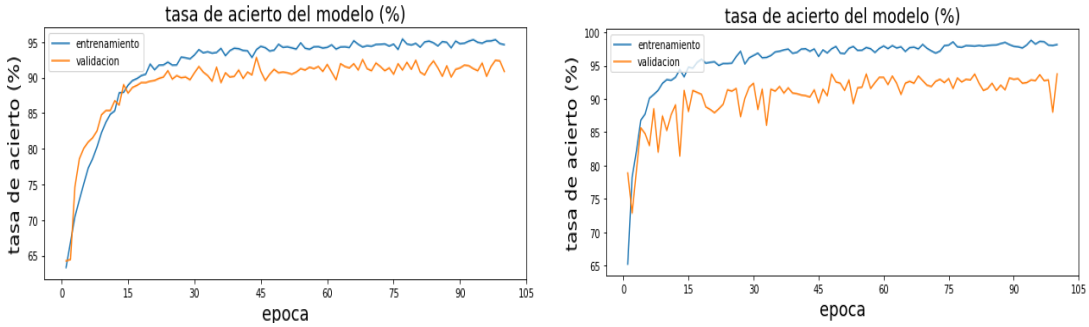


Figura 37. Reducción del *overfitting* mediante Dropout (izquierda) y Batch Normalization (derecha)

Además de reducir el sobreajuste de manera más efectiva, el uso de Capas Dropout con distintos valores resulta en un aumento del *recall*, una medida de especial interés para nuestro problema, como veremos a continuación.

3.2.3 Prueba modelo Xception

Se ha decidido realizar una prueba empleando un modelo utilizado para gran cantidad de problemas de reconocimiento de imagen. Este modelo, desarrollado por investigadores de Google, utiliza una arquitectura basada en convoluciones “depth-wise” que permite mejorar considerablemente los resultados obtenidos por los modelos desarrollados anteriormente (por ejemplo, InceptionV3).

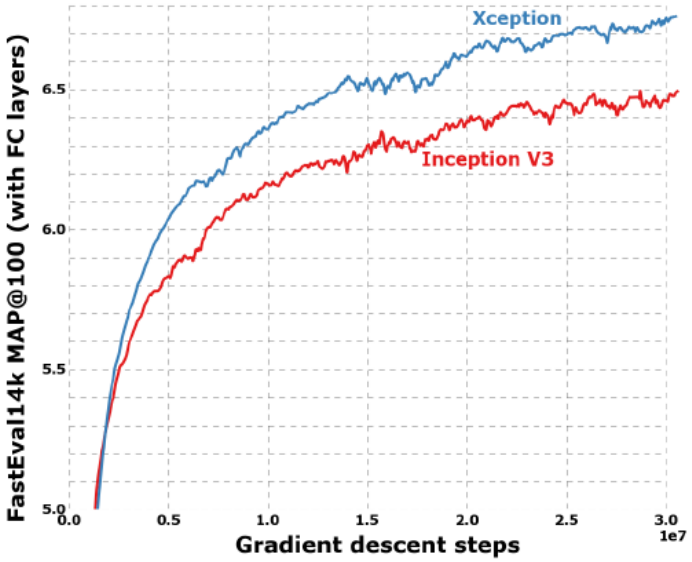


Figura 38. [Chollet. 2017] Mejora de rendimiento con respecto a modelos previos

Sin embargo, estos modelos conllevan unos tiempos de entrenamiento muy elevados, estando diseñados con equipos (granjas de GPUs) muy superiores al material disponible para este estudio. Por tanto, se ha decidido realizar la implementación en Keras de este modelo y entrenarla durante un número de épocas limitado para comprobar si los resultados podrían llegar a mejorar considerablemente. Esta configuración se puede encontrar en el Anexo 1. Configuración del modelo XCeption en Keras.

Los valores obtenidos al entrenar este modelo han sido los siguientes:

- Test accuracy: 93,65%
- Recall: 92,12%

La ejecución se ha realizado para 40 épocas, obteniendo un tiempo promedio por época de 106 minutos (ejecución realizada en aproximadamente 70 horas). A partir de esta ejecución podemos comprobar que este modelo ofrece resultados interesantes, a cambio de aumentar considerablemente el sobreajuste, como podemos observar en la Figura 15.

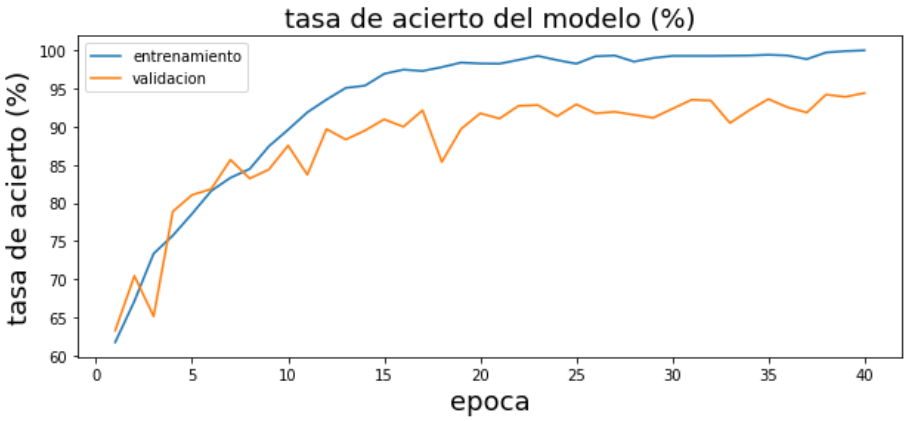


Figura 39. Entrenamiento XCeption en Keras para 40 épocas

3.2.4 Transfer Learning

Teniendo en cuenta las ventajas que nos podría traer utilizar la técnica de Transfer Learning en nuestro problema, vamos a comprobar su rendimiento en nuestra aplicación utilizando las redes preentrenadas que nos ofrece la librería Keras.

Inicialmente, nos planteamos utilizar un enfoque de Fine Tuning, ya que la mayoría de las redes ya entrenadas que incluye Keras (Xception, InceptionV3, VGG…) estarían utilizando pesos procedentes del entrenamiento en ImageNet[10], es decir, se trata de un tipo de problema de clasificación muy distinto al diagnóstico médico (identificación de objetos).

Este enfoque permite ajustar en medida de lo posible la red ya entrenada a nuestro sistema, pudiendo llegar a utilizar nuestro mejor modelo descubierto anteriormente sobre cualquiera de estas redes. Para ello, realizamos la implementación de nuestro sistema basado en Fine Tuning de la siguiente manera:

- Importamos el modelo que vamos a utilizar mediante la funcionalidad de Keras.applications, en nuestro caso, InceptionV3. Tras esto, podemos obtener nuestro modelo base con los pesos obtenidos tras su entrenamiento utilizando ImageNet. Como vamos a añadir nuestra propia capa densa, no incluiremos las capas "fully connected" del modelo ya entrenado (include_top = False).

```
# import InceptionV3 from keras applications
from keras.applications.inception_v3 import InceptionV3

# create the base pre-trained model
base_model = InceptionV3(weights='imagenet', include_top=False)
```

Figura 40. Import e inicialización del modelo ya entrenado sobre ImageNet, InceptionV3

- Tras esto, tan solo debemos implementar nuestro propio modelo, añadiéndolo sobre el output del modelo que acabamos de importar, como se puede observar en la Figura 18. En ella, podemos observar cómo se obtiene la salida del modelo InceptionV3, y, mediante la API funcional de Keras se añaden las distintas capas que vamos a entrenar para realizar el proceso de Fine Tuning.

```
# add a global spatial average pooling layer
x = base_model.output
x = GlobalAveragePooling2D()(x)

# my own convnet, used in the previous examples
x = Conv2D(filters=32, kernel_size=(3,3),
                activation='relu', input_shape = input_shape)(x)
x = Conv2D(filters=32, kernel_size=(3,3),
                activation='relu')(x)
x = MaxPooling2D(pool_size=(2,2))(x)
x = Dropout(0.5)(x)
x = Dense(64,activationr='relu',
               kernel_regularizer=regularizers.l2(l=0.1))(x)

# logistic layer, we pass the number of classes to
# predict (2 in this case)
predictions = Dense(num_classes,activation='softmax')(x)
```

Figura 41. Implementación del modelo utilizado anteriormente sobre InceptionV3

[10] ImageNet. Base de datos de imágenes del mundo real: http://www.image-net.org/

- Por último, congelamos las capas del modelo base, entrenamos nuestras capas durante un número pequeño de épocas (5 como se recomienda en el libro “Deep Learning with Python”) y desbloqueamos las capas en las que queremos hacer Fine Tuning antes de comenzar el entrenamiento final. En este caso, desbloqueamos los dos últimos bloques del modelo InceptionV3.

```
# example on how to freeze and unfreeze layers, we train the last
# two blocks on InceptionV3 model
for layer in model.layers[:249]:
   layer.trainable = False
for layer in model.layers[249:]:
   layer.trainable = True
```

Figura 42. Congelar y descongelar distintas capas del modelo base

Una vez diseñado el sistema, es el momento de probarlo y estudiar los resultados obtenidos. Sin embargo, nos encontramos con un problema que no se había tenido en cuenta: Las imágenes sobre las que están entrenadas todas las redes incluidas en Keras están en formato RGB. Por tanto, tenemos un sistema que está esperando imágenes codificadas en tres canales (rojo, verde, azul), en lugar de un único canal como nuestras imágenes (escala de grises).

Ante este problema, se nos ocurren dos posibles soluciones: Modificar el input en la primera capa del modelo ya entrenado para que acepte imágenes en escala de grises, y, modificar nuestras imágenes, transformándolas a RGB.

Descartamos la primera opción, ya que modificar la arquitectura del modelo acabaría con la integridad de los valores obtenidos para los distintos pesos. Esto se debe a que, como hemos mencionado anteriormente, las redes convolucionales utilizan valores procedentes de las primeras capas para extraer las características de mayor complejidad en los niveles superiores.

Por tanto, nos queda una única opción, modificar nuestras imágenes para que se ajusten al formato de entrada de los modelos ya entrenados en Keras: datos codificados en tres canales, en nuestro caso, RGB:

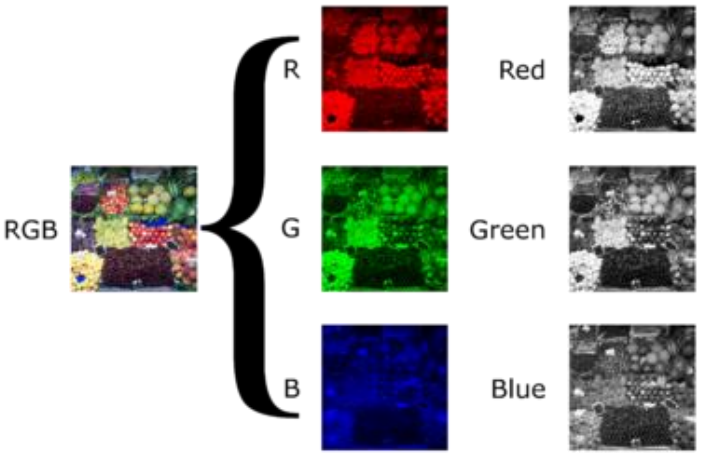


Figura 43. Codificación en tres canales RGB

La manera de implementar esta transformación es repetir la intensidad de nuestra escala de grises en los tres canales para crear una imagen RBG. Por ejemplo, si tenemos un valor de 50 en nuestra escala de grises, la imagen RBG tendría como estructura R=50, G=50, B=50. Esta solución se puede implementar de manera sencilla, utilizando la función ***repeat*** de numpy o la función ***merge*** de opencv.

Una vez adaptadas las imágenes al formato necesario, podemos ejecutar el modelo InceptionV3 con las capas que hemos añadido utilizando nuestra base de datos.

Se pueden observar beneficios del Transfer Learning como una mayor velocidad de convergencia (el modelo aprende más rápido) y una precisión inicial mayor. Pese a esto, aumenta considerablemente el sobreajuste con respecto a los modelos estudiados anteriormente y se obtienen resultados similares en cuanto a precisión (92%) y recall (90%), por tanto, se decide descartar el uso de esta técnica.

3.3 Benchmarking y mejora de tiempos

Una vez optimizado el modelo que vamos a utilizar y reducido todo lo posible el *overfitting*, vamos a medir el rendimiento de nuestro modelo, analizando el tiempo de ejecución tanto en CPU como en GPU, además de buscar el tamaño óptimo para el parámetro batch size (número de elementos que se utilizan para cada paso del entrenamiento).

Las pruebas se llevarán a cabo en el mismo dispositivo, un ordenador portátil con las siguientes características:

- CPU: Intel® Core™ i7-6600U 2,6GHz – 3.4GHz(turbo)
- GPU: Nvidia GeForce 930M

3.3.1 Tensorflow CPU vs GPU

Se miden los tiempos con la configuración obtenida tras minimizar el *overfit* (añadiendo las capas Dropout con factores 0,5 y 0,3):

Para entrenar nuestro modelo mediante CPU, simplemente utilizamos la librería por defecto de Keras, obteniendo un tiempo total de 2.483s. o 41,4 minutos.

El entrenamiento en GPU utiliza la librería Keras-GPU, que permite configurar los dispositivos en los que se realizará el entrenamiento (CPU, GPU, múltiples GPUs, etc). Una vez instalada, comprobamos que se están reconociendo correctamente todos los dispositivos y que el *backend* utilizado (en este momento Tensorflow) está configurado para utilizar la tarjeta gráfica.

```
from tensorflow.python.client import device_lib
print(device_lib.list_local_devices())
print(tf.test.is_gpu_available())
```

```
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 17980916701222184605
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 1462163865
locality {
  bus_id: 1
  links {
  }
}
incarnation: 17959606086301651081
physical_device_desc: "device: 0, name: GeForce 930M, pci bus id: 0000:03:00.0, compute capability: 5.0"
]
True
```

Figura 44. Reconocimiento de dispositivos de Tensorflow-GPU

Tras esto, medimos los tiempos de entrenamiento, obteniendo 1.456s o 24,2 minutos, una mejora considerable.

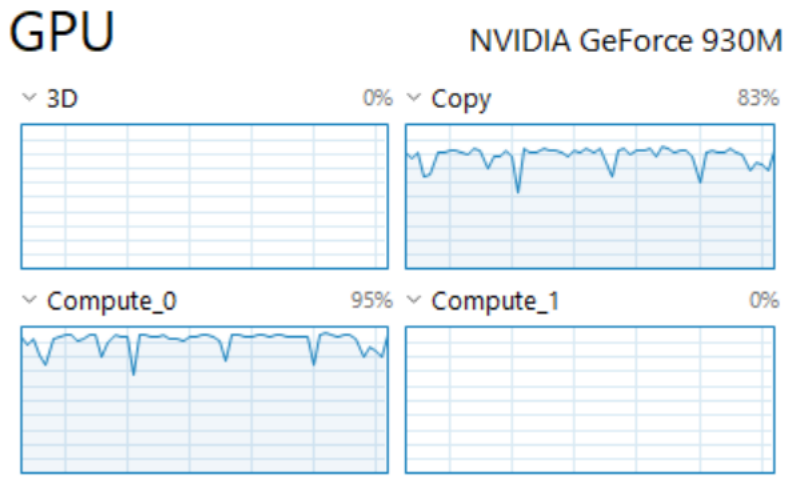


Figura 45. Uso de GPU durante la ejecución

3.3.2 CNTK CPU vs GPU

Las pruebas de CNTK se han realizado de la misma manera que las de Tensorflow, únicamente modificando el *backend* a utilizar en el fichero Keras.json, o bien asignando CNTK a la variable de entorno KERAS_BACKEND de la siguiente manera:

```
#Configurar el backend de keras para utilizar CNTK mediante
#la variable de entorno KERAS_BACKEND (alternativa: Keras.json)

import os
os.environ["KERAS_BACKEND"] = "cntk"
import keras; import keras.backend
if keras.backend.backend() != 'cntk':
    raise BaseException("This script uses other backend")
else:
    #keras.backend.set_image_dim_ordering('tf')
    print("Backend ok")
```

```
Using CNTK backend

Backend ok
```

Figura 46. Configuración backend CNTK mediante Keras

Por defecto, se utiliza el primer dispositivo encontrado, en este caso el CPU de nuestro portátil. Para realizar las pruebas con GPU se ha modificado el dispositivo a utilizar mediante el método *try_set_default_device* de CNTK.

Los tiempos obtenidos utilizando este backend han sido los siguientes:

- CPU: 6.226s o 103,7 minutos
- GPU: 1.173s o 19,5 minutos

3.3.3 PyTorch CPU vs GPU

La configuración de PyTorch es muy distinta a las tecnologías anteriores. Para poder realizar el entrenamiento, debemos indicar en qué dispositivo estamos almacenando nuestros elementos (modelo, predicciones…) para que puedan comunicarse entre sí de manera correcta.

Para ello, debemos seleccionar el dispositivo utilizando el método *torch.device* (CPU o GPU). Tras esto, simplemente movemos el modelo al dispositivo en el que queremos realizar el entrenamiento mediante el método *.to(*dispositivo a utilizar*).*

Los resultados obtenidos mediante esta tecnología son los siguientes:

- CPU: 7.155s o 119,25 minutos
- GPU: 870s o 14,5 minutos

3.4 Comparativa de tecnologías

A partir de las pruebas realizadas, se han sacado varias conclusiones acerca del funcionamiento de estas tecnologías y su rendimiento utilizando CPU y GPU:

- Tensorflow consigue unos resultados mucho mejores utilizando únicamente la CPU. Esto se debe a que tanto CNTK como PyTorch están diseñados para GPU en el caso de CNTK y para GPU o TPU en el caso de PyTorch.

- PyTorch obtiene los mejores tiempos utilizando la GPU, como era de esperar debido a su implementación y la gran cantidad de opciones que da al programador a la hora de

mejorar el rendimiento. Por otra parte, conlleva un desarrollo de bastante menor nivel que las tecnologías asociadas a Keras.

- CNTK no se queda demasiado lejos de PyTorch en tiempos utilizando GPU, permitiendo alternar entre *backends* fácilmente y utilizar esta tecnología junto a Tensorflow gracias a Keras.

A continuación, se puede observar la diferencia de tiempos obtenida mediante las distintas tecnologías:

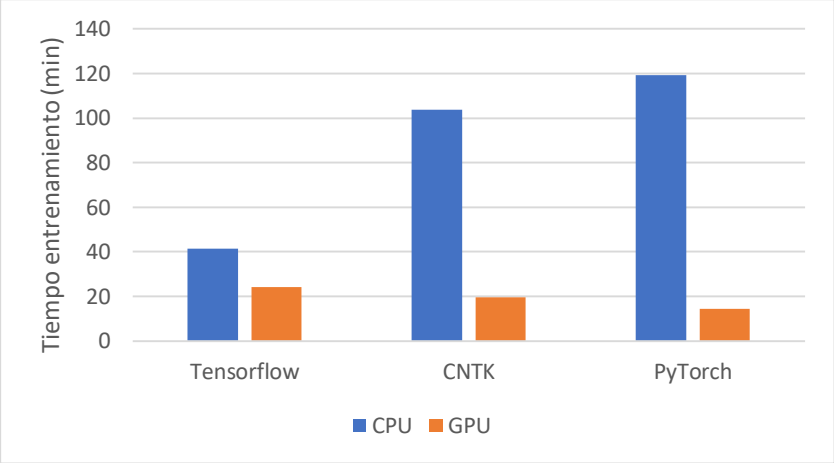


Figura 47. Comparativa CPU vs GPU en las tres tecnologías seleccionadas

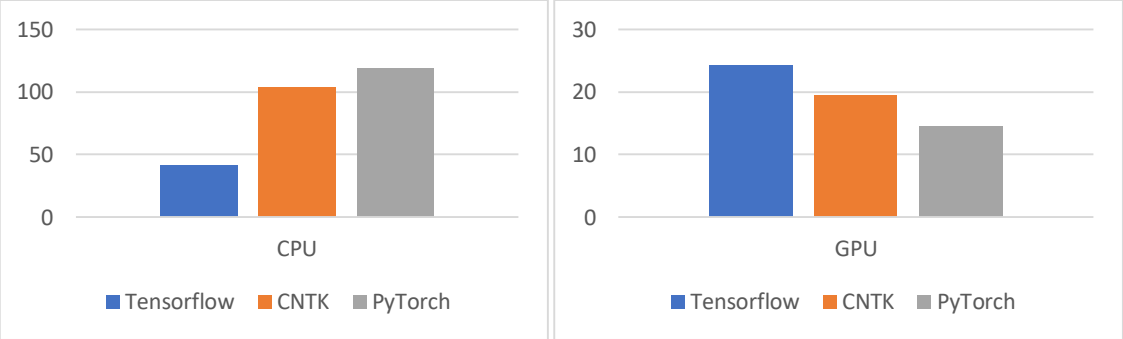


Figura 48. Comparación side-by-side del tiempo de ejecución en GPU y CPU

3.5 Selección de batch size

Tras comprobar los tiempos de ejecución de las distintas tecnologías utilizando distintos dispositivos, estudiaremos cómo afecta el parámetro batch size al tiempo de ejecución y los resultados obtenidos con el mejor modelo encontrado.

El valor inicial utilizado para este parámetro ha sido de 64. Se realizarán pruebas con potencias de 2 de mayor valor, para tratar de reducir el tiempo de ejecución manteniendo resultados de *accuracy* y *recall* similares. Estas pruebas se realizarán utilizando GPU, ya que se han obtenido los mejores tiempos en todas las tecnologías.

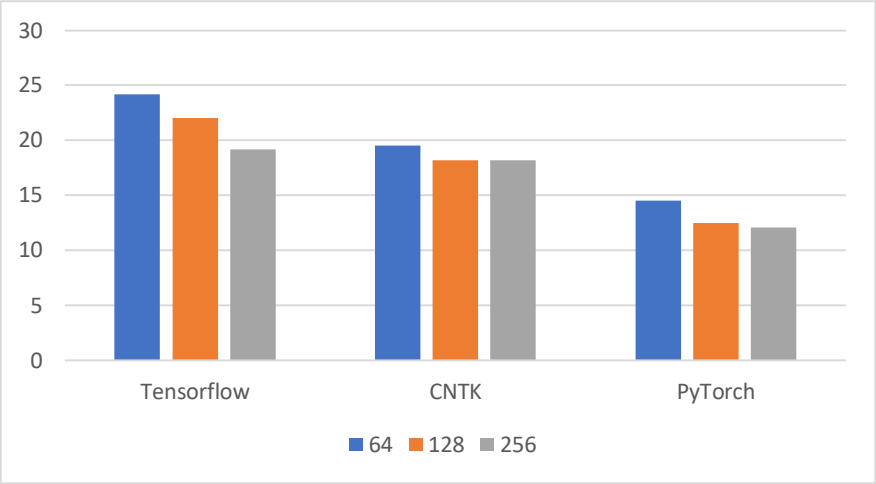


Figura 49. Comparación de tiempos para distintas batch sizes

Como se puede observar en la Figura 26., realizar cambios en el valor del parámetro batch size causa mayores cambios en el tiempo de ejecución en Tensorflow y PyTorch, siendo CNTK la tecnología que menos se beneficia de aumentar este número.

El aumento del tamaño de batch desde 64 unidades a 256 ocasiona una reducción del tiempo de entrenamiento de un 21% para Tensorflow, mientras que en el caso de PyTorch se reduce un 16,9% y tan solo un 7% en el caso de CNTK.

Sin embargo, un cambio en este parámetro puede ocasionar cambios significativos en los resultados del modelo (ya que el cálculo del gradiente se vuelve menos preciso para valores más pequeños). Por tanto, debemos estudiar cómo afectan los cambios en el batch size a los valores del *recall*, *accuracy* y área bajo la curva ROC como hemos hecho anteriormente:

3.5.1 Análisis batch size Tensorflow

	Accuracy	Recall	ROC Area
64	0,937	0,925	0,98
128	0,942	0,927	0,98
256	0,930	0,922	0,97

Como podemos observar, además de reducir considerablemente el tiempo de ejecución, se consiguen resultados similares aumentando el batch size de 64 a 128 unidades. Sin embargo, una vez se aumenta este valor los resultados empeoran significativamente.

3.5.2 Análisis batch size CNTK

	Accuracy	Recall	ROC Area
64	0,922	0,936	0,97
128	0,923	0,915	0,97
256	0,912	0,913	0,96

En el caso de CNTK, obtenemos unos resultados bastante similares a los de Tensorflow. Utilizando 128 unidades se consiguen resultados similares en cuanto a *accuracy* y área ROC, sin embargo, en este caso el valor del *recall* disminuye considerablemente. Por tanto, descartaríamos aumentar el tamaño de 64 a 128 pese a las reducciones del tiempo de ejecución.

3.5.3 Análisis batch size PyTorch

	Accuracy	Recall	ROC Area
64	0,941	0,935	0,98
128	0,945	0,935	0,98
256	0,945	0,940	0,98

PyTorch es la única tecnología que muestra unos resultados considerablemente distintos, mejorando tanto el *accuracy* como el *recall* al tomar como valor 128 y 256 unidades. En este último caso, se obtienen los mejores resultados mientras que en las tecnologías anteriores se empeoraban los valores del *accuracy* considerablemente.

3.5.4 Comparativa de tecnologías y umbral de decisión

Los mejores resultados tanto para el tiempo de ejecución utilizando GPUs como para el *recall* y *accuracy* se obtienen mediante PyTorch, utilizando 256 unidades como batch size. Sin embargo, para el desarrollo de este TFM hemos decidido continuar utilizando **Keras** con el backend **Tensorflow**, debido a que los sistemas disponibles para realizar el entrenamiento de modelos con la segunda base de datos no disponen de GPUs potentes con las que aprovechar la mejora de tiempos para esta tecnología.

La mejor configuración encontrada RELU como función de activación, ADAMAX como optimizador y una combinación de capas Dropout y regularizadores para reducir el sobreajuste. Además, el número de batchsize que ofrece los mejores resultados para el *recall* y el área bajo la curva ROC es 128. La matriz de confusión obtenida se puede observar en la Figura 28.

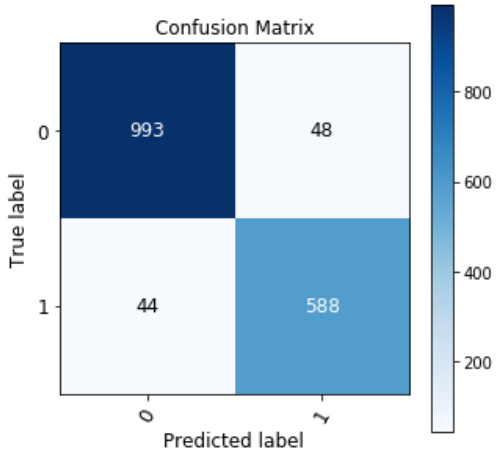


Figura 50. Matriz de confusión utilizando la mejor configuración encontrada para Tensorflow

Esta matriz se genera teniendo en cuenta 0,5 como el umbral mínimo de probabilidad para considerar que una imagen se deberá clasificar como maligna. Es decir, un valor de 0,6 será considerado maligno mientras que un porcentaje de 0,2 será etiquetado como benigno por el modelo.

En nuestro problema, podría ser beneficioso modificar este porcentaje para clasificar como maligno a partir de un umbral bastante menor, ya que, como hemos mencionado anteriormente, el riesgo de predecir que un paciente no tiene cáncer de pulmón cuando en realidad lo tiene es mucho mayor que lo contrario.

Utilizando valores menores de este umbral de decisión, se consigue reducir considerablemente la cantidad de falsos negativos encontrados. Sin embargo, se produce a su vez un aumento de los falsos positivos, y, en muchos casos, una disminución del *accuracy* obtenido.

Aun así, al tratarse de un sistema de apoyo a los médicos, podría ser aconsejable utilizar umbrales bastante más pequeños y conseguir eliminar una gran cantidad de casos (aquellos en los que el sistema obtiene un grado muy alto de seguridad para predecir que el alumno no tiene una enfermedad), aliviando el trabajo de los médicos. Por ello, se han obtenido valores utilizando 0,4 y 0,2 como valores para tomar las decisiones utilizando este último modelo:

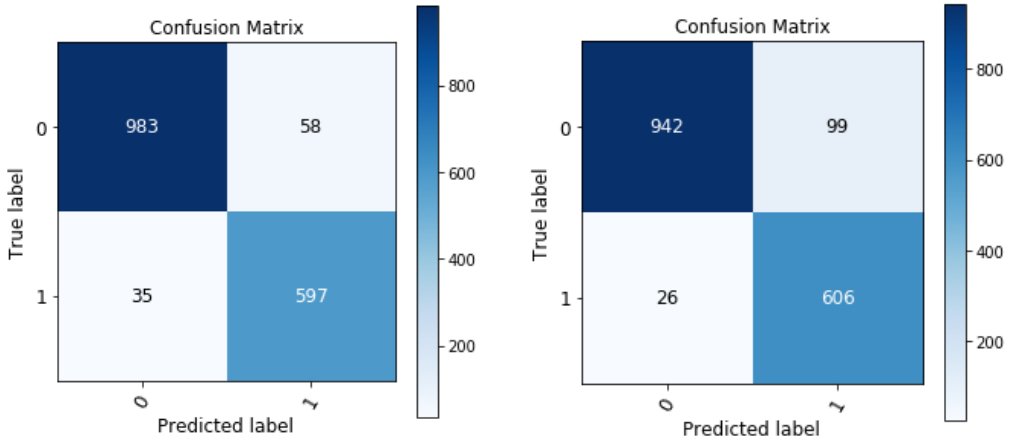


Figura 51. Umbrales de decisión: 0,4 (izquierda) y 0,2 (derecha)

Como podemos observar en la Figura 29, los falsos negativos se reducen de 44 a 35 utilizando 0,4 como umbral, provocando un aumento de falsos positivos de 48 a 58. Esto provoca una bajada (aunque muy pequeña) del *accuracy*. Sin embargo, reducir el número de falsos negativos es muy importante para nuestro sistema.

Este valor deberá ser considerado cuidadosamente, teniendo en cuenta cuánta importancia tiene un falso negativo con respecto a un falso positivo, ya que, como indica claramente la Figura 30, cuanto más se reduce el umbral de decisión mayor número total de falsas predicciones produce nuestro sistema (a cambio de reducir los falsos negativos).

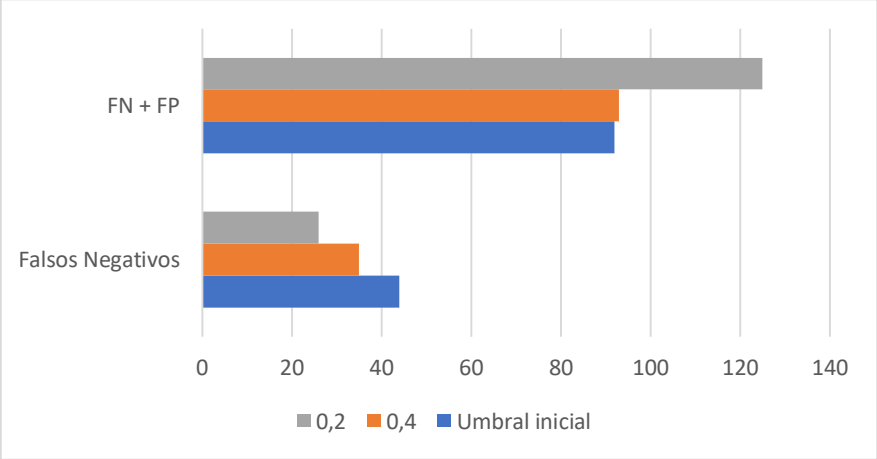


Figura 52. Aumento de las falsas predicciones con respecto al umbral de decisión

3.6 Problema ChestX-Ray14

3.6.1 Estado inicial

El primer modelo entrenado para realizar las pruebas con la segunda base de datos, ChestX-Ray14, está compuesto por una estructura muy tradicional en las redes convolucionales, explicada anteriormente en el apartado de metodología:

- Capa de entrada Batch Normalization
- Tres conjuntos de capas Convolucionales, MaxPooling y Batch Normalization. En las capas convolucionales se utilizarán 32 filtros y un tamaño de kernel variable, comenzando en 11x11 en la primera capa y reduciéndose progresivamente. En la segunda capa se utiliza un kernel de 7x7 y en la tercera capa se utiliza un kernel de 5x5.
- Capa de aplanamiento (Flatten).
- Capa Dropout con valor 0.5 para reducir el sobreajuste
- Dos capas Densas, la primera de las cuales utilizará activación relu con 64 unidades y un regularizador l2 con valor 0.5. La segunda será nuestra capa logística, encargada de obtener los resultados del modelo (predicción entre las dos clases existentes).

El optimizador utilizado en esta versión inicial es Adam, con un valor para el learning rate de 0,0001.

Utilizando esta arquitectura para entrenar nuestro modelo durante 200 épocas, obtenemos los siguientes resultados:

Train Accuracy	Validation Accuracy	Test accuracy	Recall	ROC Area
0,695	0,684	0,652	0,65	0,68

3.6.2 Optimización del modelo

La siguiente fase es la optimización del modelo. Partiendo de la configuración descrita anteriormente, trataremos de obtener los mejores resultados posibles para el conjunto de datos actual.

3.6.2.1 GridSearchCV

La librería de machine learning SKLearn nos ofrece una herramienta conocida como Grid Search que utilizaremos inicialmente para reducir el número de pruebas que vamos a realizar para tratar de optimizar los distintos parámetros.

Mediante esta herramienta, configuramos un conjunto de parámetros, conocido como *parameter grid*, introduciendo valores a probar. Una vez configurados estos valores, la herramienta realizará el entrenamiento del sistema utilizando todas las combinaciones posibles. Por ejemplo, si introducimos 3 valores para un primer parámetro y 5 valores para un segundo, GridSearch realizará pruebas con las 15 combinaciones resultantes.

Una gran ventaja de esta tecnología es que permite la paralelización total del proceso, aprovechando el número máximo de procesadores disponibles mediante su parámetro *n_jobs*. Por tanto, podremos hacer uso de los 12 procesadores disponibles en nuestro servidor.

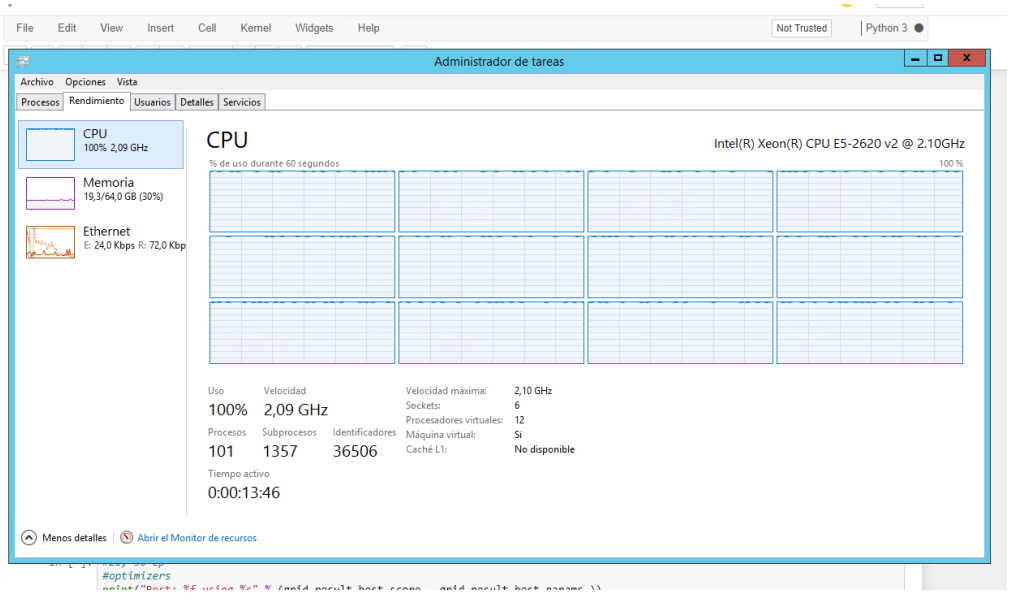


Figura 53. Aprovechamiento de los 12 procesadores en el servidor

Los parámetros que utilizaremos para configurar esta herramienta serán los siguientes:

- Optimizador (Adam, Adamax, SGD, RMSprop)
- Batch_size (32, 64, 128)

En la Figura 39 se puede observar gráficamente un resumen de los resultados obtenidos, representando visualmente el rendimiento de los distintos optimizadores utilizados.

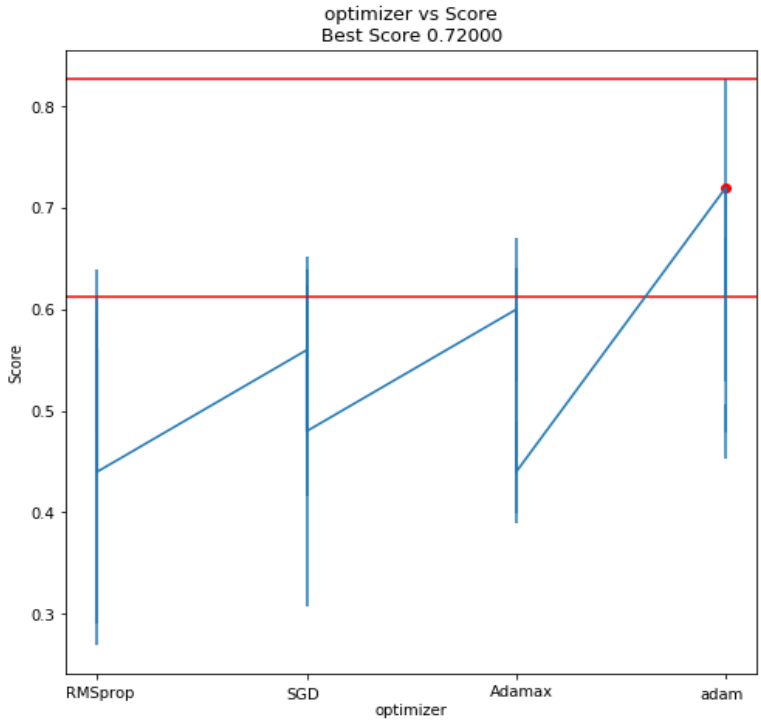


Figura 54. Resultados obtenidos utilizando el optimizador como parámetro más relevante

Tras ejecutar la búsqueda de parámetros, podemos extraer las siguientes conclusiones:

- Los optimizadores Adam y Adamax ofrecen resultados mucho mejores que SGD y RMSprop.
- Adam es el optimizador que ofrece mejores valores para el *accuracy*.
- Las tres configuraciones con mayores *accuracies* utilizan 64 como valor para el parámetro *batch_size*.

3.6.3 Pruebas realizadas

Una vez obtenidos los mejores valores para el optimizador y el *batch_size*, pasaremos a realizar pruebas modificando la arquitectura de la red convolucional y los distintos valores para el *learning_rate* en el optimizador y en los distintos regularizadores de kernel en nuestras capas densas.

Las pruebas a continuación han sido realizadas utilizando un número total de épocas de 200. En el ordenador portátil utilizado en los apartados anteriores, el tiempo medio de ejecución es de 8 horas y 20 minutos utilizando arquitecturas con cuatro conjuntos de capas Convolucionales, Max Pooling y Batch Normalization, y de aproximadamente 6 horas y 30 minutos realizando únicamente tres conjuntos.

Los resultados se dividirán en dos grupos distintos: Arquitecturas que utilizan tres y cuatro conjuntos de capas Convolucionales, de Pooling y Batch Normalization respectivamente. Se analizará el rendimiento de los cuatro mejores modelos encontrados para cada categoría.

Al igual que en las pruebas realizadas con la base de datos Lung Node Malignancy, analizaremos distintas métricas para evaluar el funcionamiento de nuestro modelo. Nos fijaremos en el accuracy obtenido para los conjuntos de training, validación y test, dándole especial importancia a la precisión obtenida en test (datos a partir de los que no se han aprendido características).

Además, tendremos en cuenta el recall como la métrica más importante en nuestro modelo, tratando de maximizarlo siempre y cuando no se produzca una gran disminución en la precisión o el área bajo la curva ROC.

3.6.3.1 Modelos con tres conjuntos de capas convolucionales

Partiendo de la arquitectura inicial, podemos observar mejoras en los resultados modificando el valor del regularizador de kernel L2 de 0,5 a 0,1 al utilizar el algoritmo Adamax. Esta composición, realizando dos sencillas modificaciones, es capaz de aumentar el área bajo la curva ROC con respecto al modelo inicial, manteniendo el mismo valor para el recall.

Otra observación interesante es que se consiguen mantener unos resultados muy similares al modelo inicial si disminuimos el tamaño de todos los kernels en las capas convolucionales a 3x3. Este proceso acelera enormemente el entrenamiento, llegando a reducir el tiempo consumido desde 6 horas y media hasta apenas 4 horas.

Modelo	Train acc.	Val acc.	Test acc.	Recall	ROC Area
Adam L2 0,1	0,666	0,652	0,630	0,625	0,68
Adam L2 0,5	0,672	0,655	0,623	0,61	0,68
Adam L2 0,1 Kernels 3x3	0,682	0,667	0,645	0,645	0,68
Adamax L2 0,1	0,672	0,668	0,645	0,65	0,69

3.6.3.2 Modelos con cuatro conjuntos de capas convolucionales

Al ejecutar las primeras pruebas, se puede observar rápidamente que la incorporación de las nuevas capas mejora considerablemente los resultados obtenidos por nuestros modelos.

Se han realizado pruebas utilizando distintas arquitecturas, entre ellas la incorporación de una tercera capa densa antes de la capa de salida, además de tratar de optimizar el valor del learning rate para el algoritmo Adamax, utilizando los valores recomendados en el paper "ADAM: A method for stochastic optimization".

Sin embargo, los mejores resultados que hemos podido encontrar se han obtenido utilizando una arquitectura relativamente sencilla: cuatro conjuntos convolucionales combinados con el optimizador Adam y regularización L2 con un valor de 0,1, como se puede observar en la tabla a continuación:

Modelo	Train acc.	Val acc.	Test acc.	Recall	ROC Area
Adam L2 0,1	0,728	0,686	0,670	0,67	0,71
Adam L2 0,1 2x Dense	0,749	0,686	0,655	0,655	0,70
Adamax L2 0,1 LR 0,0001	0,702	0,692	0,644	0,645	0,69
Adamax L2 0,1 LR 0,0002	0,707	0,688	0,645	0,640	0,69

Una vez ejecutadas estas pruebas, se ha decidido comprobar si resultaría beneficioso aumentar el tamaño de las imágenes utilizadas (actualmente rescaladas a 128x128 con el objetivo de obtener unos tiempos de ejecución razonables).

Para ello, se han entrenado los tres modelos con los mejores valores para el recall utilizando imágenes con el doble de tamaño (256x256). Debido a la pérdida de calidad y detalles de las imágenes al disminuir su tamaño, al aumentar la resolución se esperaría un aumento de la precisión del sistema.

Sin embargo, en todos los modelos entrenados con un mayor tamaño de imagen, nos hemos encontrado con una disminución significativa del rendimiento del sistema.

Utilizando la mejor arquitectura (Adam + Regularización L2 con valor 0,1) con imágenes de 256x256, se obtienen los siguientes resultados:

Train Accuracy	Validation Accuracy	Test accuracy	Recall	ROC Area
0,695	0,682	0,642	0,640	0,68

Esto supone una gran disminución tanto en el recall como en el área bajo la curva ROC. Además, el tiempo de entrenamiento aumenta considerablemente, pasando de las 8 horas y 20 minutos mencionadas con anterioridad a cerca de 26 horas.

Por tanto, se ha decidido descartar el aumento del tamaño de las imágenes y las pruebas con tamaños aún mayores, ya que se dispararía el tiempo de ejecución y sería muy probable que los resultados siguiesen empeorando.

3.6.4 Análisis de extremos

Los resultados obtenidos hasta el momento son similares a los que se detallan en estudios previos [Wang et al. 2017], sin embargo, nuestro modelo presenta características muy positivas que se han podido observar a la hora de analizar los resultados.

Una vez comprobados los valores finales para las precisiones en training, validación y test, además del recall y el área bajo la curva ROC, se ha realizado un análisis de los casos con mayores y menores probabilidades de enfermedad.

Este análisis muestra características muy interesantes de nuestro modelo, ya que, en los casos que se encuentran en ambos extremos, el modelo obtenido tiene un porcentaje bastante elevado de acierto, especialmente en aquellos casos con mayor % de haber encontrado un *finding*, es decir, casos de alto riesgo en los que un diagnóstico equivocado podría suponer grandes problemas.

	Porcentaje de acierto	
N casos	Menor % enfermedad	Mayor % enfermedad
10 Casos	80%	90%
20 Casos	70%	90%
25 Casos	75%	92%
50 Casos	70%	90%

3.6.5 Análisis de heatmaps

A continuación, se realizará un análisis de los heatmaps obtenidos para las dos clases a predecir: Finding y No Finding en varios de los casos encontrados en los extremos, es decir, aquellos casos en los que el sistema genera probabilidades más cercanas a 100% y 0%.

3.6.5.1 Casos con mayor probabilidad de finding

En el primer caso analizado, podemos ver como el sistema observa distintas zonas de la radiografía de tórax para cada una de las clases. Para la clase 0, está observando zonas que no se corresponden con los pulmones. Sin embargo, en la clase 1, el sistema parece estar prestando especial atención a la parte central de ambos pulmones, algo que concuerda perfectamente con la patología etiquetada para esta radiografía: Neumonía.

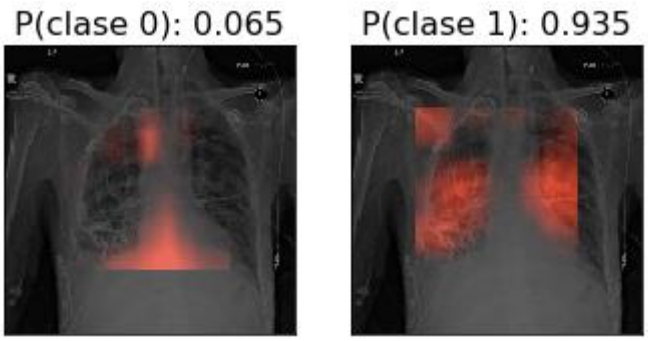


Figura 55. Heatmaps y probabilidades para ambas clases en el caso con mayor probabilidad detectado para el conjunto ChestX-Ray14

El segundo caso analizado se corresponde con la detección de un posible nódulo de pulmón de pequeño tamaño en la radiografía. Como podemos observar en la Figura 56, la clase 0 recorre la parte inferior del tórax y ambos pulmones, mientras que el heatmap para la clase 1 se centra en una zona específica, indicando que se ha encontrado una patología.

Figura 56. Heatmaps y probabilidades para ambas clases en el décimo caso con mayor probabilidad detectado para el conjunto ChestX-Ray14

3.6.5.2 Casos con menor probabilidad de finding

Los casos con las menores probabilidades muestran unos heatmaps muy diferentes a los estudiados anteriormente. En todos salvo uno de los casos generados para las diez radiografías con menores porcentajes, se muestra un heatmap vació para la clase 1. Es decir, el sistema no reconoce ninguna parte de la radiografía que se corresponda con las patologías aprendidas.

Es decir, se da un comportamiento ideal, deseado para nuestro sistema. Como podemos observar en la Figura 57, la clase 1 recorre gran parte de los pulmones, detectando zonas que el sistema considera sanas, mientras que la clase 1 apenas muestra señales.

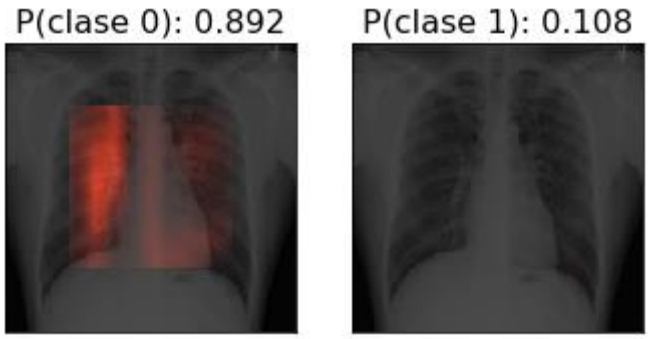


Figura 57. Heatmaps y probabilidades para ambas clases en el caso con menor probabilidad detectado para el conjunto ChestX-Ray14

El único caso en el que se puede apreciar la detección de una zona en la que exista una patología se da en la séptima imagen con menor probabilidad de presentar *findings*. Sin embargo, la intensidad del rojo mostrada en el heatmap es muy tenue, como se puede observar en las zonas resaltadas en la Figura 58.

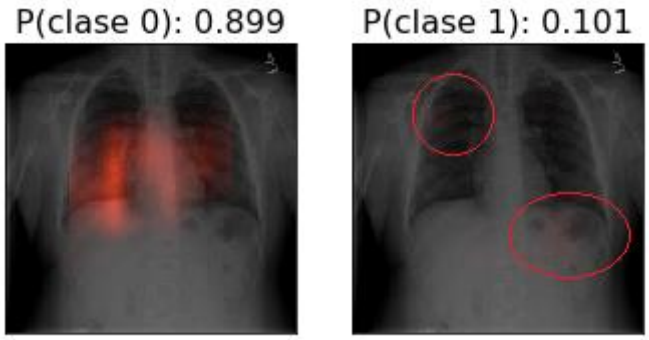


Figura 58. Heatmaps y probabilidades para ambas clases en el séptimo caso con menor probabilidad detectado para el conjunto ChestX-Ray14

3.7 Pruebas con imágenes reales

A partir de las radiografías utilizadas, se han generado heatmaps mostrando las partes de la imagen en las que se ha fijado la red para generar la probabilidad. En varios de los casos, se puede observar claramente que las zonas iluminadas coinciden con la descripción del radiólogo indicando la detección de uno o varios nódulos. Esto se puede observar en los casos 9 y 15:

- Caso 9: Silicosis pulmonar complicada con masas (conglomerados silicóticos) en lóbulo superior y segmento apical del LID. LID = Pulmón inferior derecho.
- Caso 15: Nódulo en LID.

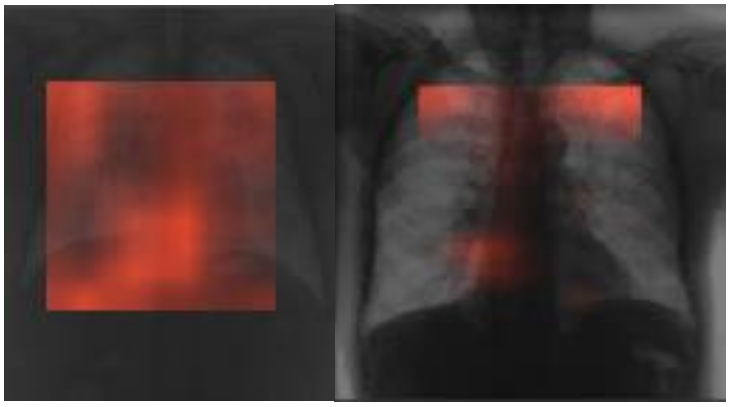


Figura 59. Heatmaps para la clase 1 en los casos 9 (izquierda) y 15 (derecha)

Tras la finalización de este TFM, se seguirá realizando un estudio de los *heatmaps* generados. En este momento se está colaborando con un experto en neumología, capaz de interpretar de manera adecuada radiografías de tórax. Este profesional será capaz de confirmar la eficacia del modelo y la generación de *heatmaps*, detallando si son de especial interés para el apoyo al diagnóstico médico.

3.8 Aplicación Dash

En la actualidad, el campo del aprendizaje automático y el Deep Learning está cobrando especial importancia debido a los avances en capacidad de computación. Multitud de universidades, empresas y centros de investigación están abordando problemas que en el pasado serían impensables [Litjens et al. 2017].

Sin embargo, nuestro objetivo no es crear un sistema que consiga superar o reemplazar a un médico especializado en la materia, sino proporcionarle toda la ayuda posible, para hacer más fácil el desarrollo de sus tareas y aliviar en medida de lo posible la enorme carga de trabajo.

Para ello, se ha decidido desarrollar una aplicación capaz de procesar cualquier radiografía introducida, devolviendo las predicciones obtenidas a partir de los modelos optimizados en apartados anteriores.

Además, se mostrará al especialista el heatmap producido por la red convolucional, es decir, aquellas partes de la imagen de entrada en las que se ha fijado nuestro sistema para realizar sus predicciones.

Esto puede ser utilizado por los especialistas para prestar especial atención a una parte de la imagen que puede indicar la existencia de una determinada enfermedad, facilitando su trabajo de manera considerable sin restarle importancia al factor más importante del diagnóstico, el criterio de los especialistas.

Una vez obtenido el modelo que vamos a utilizar para predecir si una radiografía contiene nódulos o masas de cáncer de pulmón, procedemos a implementar nuestra aplicación de ayuda al diagnóstico médico. El objetivo de esta aplicación es proporcionar ayuda a los médicos y especialistas que se encargan de analizar las radiografías.

Por tanto, nos centraremos en implementar la funcionalidad necesaria para hacer este trabajo más sencillo, restándole importancia a los aspectos gráficos de la aplicación.

3.8.1 Implementación y funcionalidad

Mediante los componentes prediseñados que nos ofrece Dash, configuramos un selector de ficheros, permitiendo seleccionar imágenes en formatos PNG o JPG.

Una vez seleccionadas una o varias radiografías, la aplicación evaluará las imágenes introducidas, mostrándolas en pantalla junto con las predicciones obtenidas, en forma de probabilidad con la siguiente estructura:

- Probabilidad finding: 0,10%
- Probabilidad no finding: 0,90%

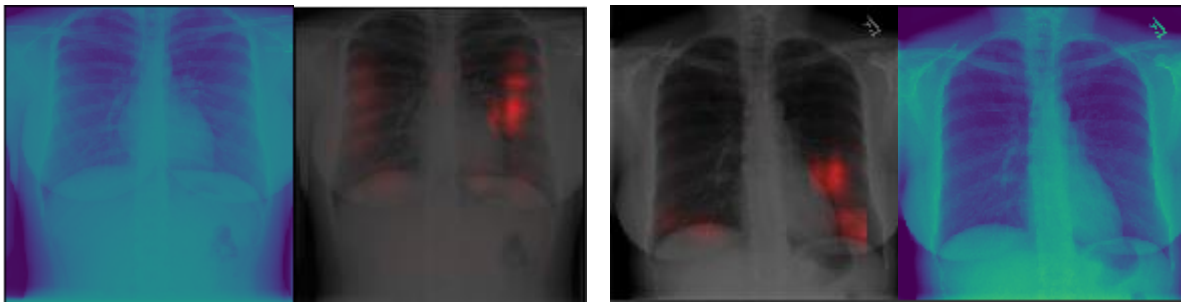


Figura 60. Heatmaps generados frente a las imágenes iniciales

Por último, las imágenes cargadas serán analizadas por el módulo de generación de heatmaps, mostrando gráficamente al usuario aquellas zonas de la imagen en las que nuestro modelo se ha apoyado para realizar la predicción, resaltando las partes de la imagen con mayor importancia.

La aplicación cuenta con un enlace que permite descargar los heatmaps producidos, de manera que se puedan observar detalladamente e incluso imprimirlos si hiciese falta analizarlos cuidadosamente.

4 Conclusiones y trabajo futuro

A partir del estudio realizado en las distintas partes de este documento, se han podido extraer las siguientes conclusiones:

- TensorFlow y PyTorch están muy por delante del resto de frameworks y tecnologías disponibles para crear sistemas basados en Deep Learning. Además, cada vez hay más usuarios que utilizan estas dos tecnologías, resultando en un crecimiento que se prevé que continúe durante los próximos años.
- TensorFlow 2.0, actualmente en estado beta, cuenta con características prometedoras, que cubren una gran parte de las ventajas que PyTorch ofrece actualmente con respecto a TensorFlow.
- TensorFlow ofrece los mejores tiempos de ejecución utilizando CPUs. Sin embargo, estos tiempos son mucho mayores que los obtenidos mediante el uso de PyTorch y CNTK en sistemas basados en GPUs.
- El uso de capas Batch Normalization resulta beneficioso para nuestros modelos, reduciendo el sobreajuste y minimizando la cantidad de épocas necesarias para el aprendizaje del sistema.
- La optimización de nuestros modelos, realizada mediante la búsqueda de parámetros óptimos y la prueba de distintas arquitecturas nos ha permitido obtener un modelo con resultados similares a los encontrados en la literatura actual en el campo de investigación de radiografías de tórax [Wang X. 2019].

4.1 Aspectos positivos

Utilizando el mejor modelo obtenido, haciendo uso del optimizador Adamax, las capas Batch Normalization y una arquitectura compuesta de cuatro capas convolucionales, hemos podido obtener un modelo que ofrece resultados muy positivos en los casos con mayores y menores porcentajes de haber encontrado un *finding*.

Estos resultados aportan credibilidad al sistema, permitiendo que los expertos aprovechen al máximo el sistema en aquellos casos en los que las probabilidades sean suficientemente extremas, es decir, cercanas al 100% o al 0% en caso de predecir que el paciente no padece ninguna enfermedad.

Además, se ha podido comprobar mediante los distintos casos de prueba que la aplicación es capaz de generar *heatmaps* en los que se visualizan las zonas más importantes de la radiografía de acuerdo al modelo utilizado. En el análisis de las radiografías con menores y mayores porcentajes de *finding*, se ha podido observar claramente el comportamiento de la red:

- En las radiografías con porcentajes menores, se puede observar como el sistema recorre los pulmones en el *heatmap* de la clase 0, asegurándose de que no existen hallazgos de ninguna patología.
- En las radiografías con mayores porcentajes, podemos apreciar las distintas zonas de interés de la imagen. En los casos reales del hospital de Vigo, se pueden observar varios casos cuyo *heatmap* coinciden con las descripciones de los expertos, asociadas con las distintas radiografías

Se ha podido llevar a cabo una prueba de concepto demostrando la utilidad de la aplicación creada en Dash. Accediendo a esta aplicación, actualmente alojada únicamente en local, los usuarios pueden introducir distintas radiografías, obteniendo las probabilidades detectadas por

el mejor modelo que hemos sido capaces de obtener. Además, esta aplicación permite consultar y descargar los *heatmaps*, que podrán ser interpretados para prestar especial atención a determinadas zonas de las radiografías, pudiendo detectar indicios de enfermedades que, en otro caso habrían pasado desapercibidos.

4.2 Extensiones futuras

Sin embargo, este estudio no cubre todos los aspectos posibles, pudiendo llevarse a cabo una cantidad de pruebas mucho mayor utilizando equipos especializados en el entrenamiento de modelos de Deep Learning.

Esto permitiría utilizar un porcentaje bastante más grande del segundo conjunto de datos estudiado (ChestX-Ray14). La selección de un mayor número de imágenes podría mejorar los modelos obtenidos, encontrando mejores características, y, resultando en un menor porcentaje de error.

Además, la aplicación desarrollada en Dash podría mejorarse, ofreciendo más opciones a los médicos, como por ejemplo introducir un conjunto de radiografías y mostrar casos similares, ordenar los distintos casos introducidos por porcentaje, o mostrar aquellas radiografías en las que se detecta una misma patología.

Una vez extendida, esta aplicación podría ser alojada en un servidor externo, permitiendo a los especialistas acceder de manera remota y utilizar el sistema desde cualquier lugar.

La interpretación de los *heatmaps* obtenidos es un trabajo en continuo crecimiento. Una vez entregado el TFM, se planea seguir colaborando con los expertos que nos han proporcionado las radiografías del hospital español para mejorar el sistema y verificar su eficacia, apoyándonos en el criterio de especialistas en el campo de la interpretación de radiografías de tórax.

En cuanto a la interfaz gráfica, no se ha optimizado la estética de la misma, ofreciendo únicamente un diseño minimalista, priorizando en todo caso la funcionalidad ofrecida sobre el aspecto de la aplicación. Sería aconsejable llevar a cabo un proceso de análisis y diseño, centrándose en cumplir los criterios de usabilidad y las guías de diseño para aplicaciones Web.

5 Anexos

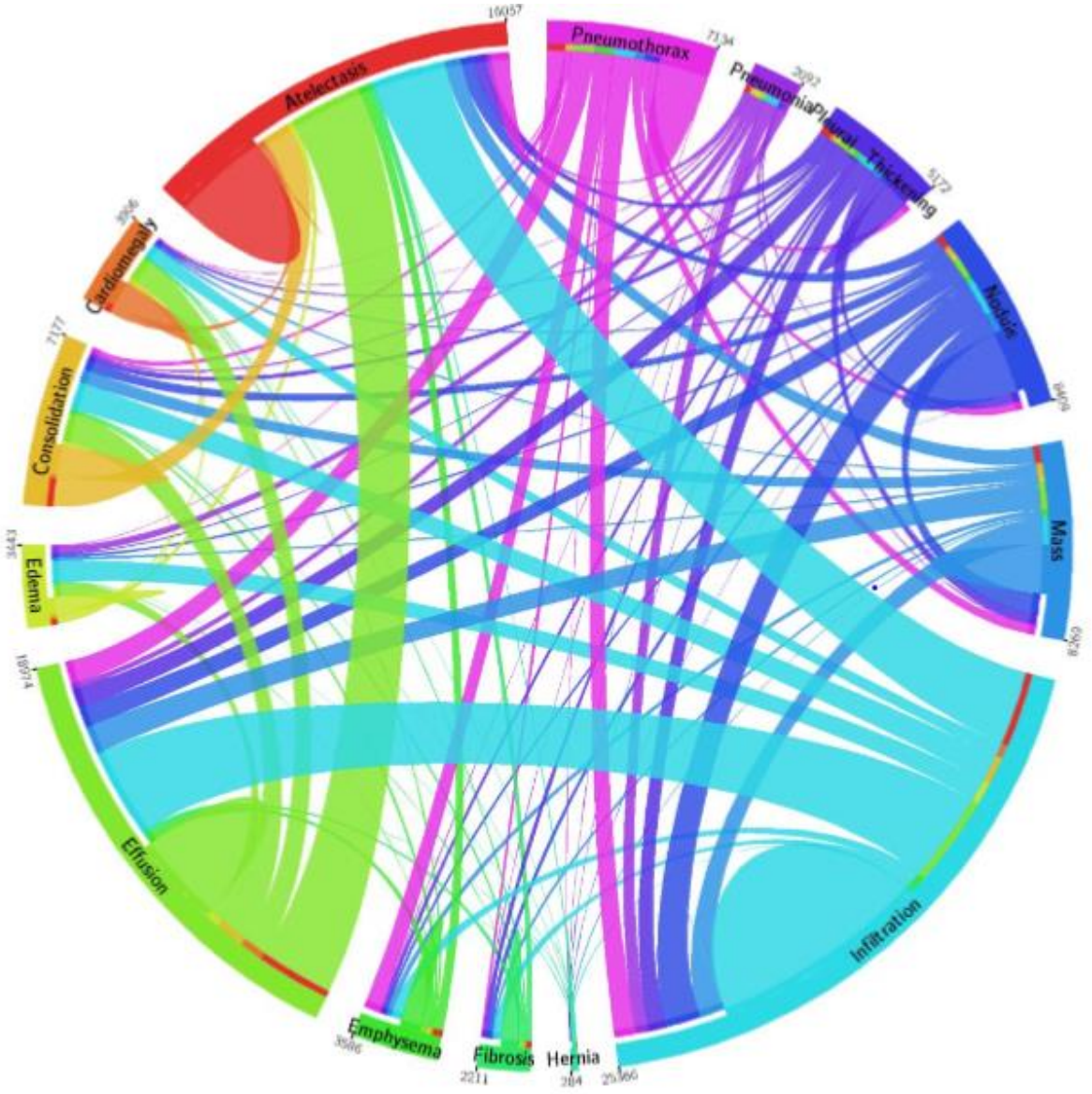
5.1 Anexo A: Configuración del modelo Xception en Keras

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 32, 32, 32)	320
batch_normalization_1 (Batch	(None, 32, 32, 32)	128
activation_1 (Activation)	(None, 32, 32, 32)	0
conv2d_2 (Conv2D)	(None, 32, 32, 64)	18496
batch_normalization_2 (Batch	(None, 32, 32, 64)	256
activation_2 (Activation)	(None, 32, 32, 64)	0
activation_3 (Activation)	(None, 32, 32, 64)	0
separable_conv2d_1 (Separabl	(None, 32, 32, 9)	1161
batch_normalization_3 (Batch	(None, 32, 32, 9)	36
activation_4 (Activation)	(None, 32, 32, 9)	0
separable_conv2d_2 (Separabl	(None, 32, 32, 9)	171
batch_normalization_4 (Batch	(None, 32, 32, 9)	36
max_pooling2d_1 (MaxPooling2	(None, 16, 16, 9)	0
activation_5 (Activation)	(None, 16, 16, 9)	0
separable_conv2d_3 (Separabl	(None, 16, 16, 728)	7361
batch_normalization_5 (Batch	(None, 16, 16, 728)	2912
activation_6 (Activation)	(None, 16, 16, 728)	0

s

```
activation_6 (Activation)    (None, 16, 16, 728)       0
separable_conv2d_4 (Separabl (None, 16, 16, 728)       537264
batch_normalization_6 (Batch (None, 16, 16, 728)       2912
activation_7 (Activation)    (None, 16, 16, 728)       0
separable_conv2d_5 (Separabl (None, 16, 16, 728)       537264
batch_normalization_7 (Batch (None, 16, 16, 728)       2912
activation_8 (Activation)    (None, 16, 16, 728)       0
separable_conv2d_6 (Separabl (None, 16, 16, 728)       537264
batch_normalization_8 (Batch (None, 16, 16, 728)       2912
activation_9 (Activation)    (None, 16, 16, 728)       0
separable_conv2d_7 (Separabl (None, 16, 16, 1024)      753048
batch_normalization_9 (Batch (None, 16, 16, 1024)      4096
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 1024)        0
activation_10 (Activation)   (None, 8, 8, 1024)        0
activation_10 (Activation)   (None, 8, 8, 1024)        0
separable_conv2d_8 (Separabl (None, 8, 8, 728)         755416
batch_normalization_10 (Batc (None, 8, 8, 728)         2912
activation_11 (Activation)   (None, 8, 8, 728)         0
separable_conv2d_9 (Separabl (None, 8, 8, 1024)        753048
batch_normalization_11 (Batc (None, 8, 8, 1024)        4096
max_pooling2d_3 (MaxPooling2 (None, 4, 4, 1024)        0
flatten_1 (Flatten)          (None, 16384)             0
dropout_1 (Dropout)          (None, 16384)             0
dense_1 (Dense)              (None, 64)                1048640
dropout_2 (Dropout)          (None, 64)                0
dense_2 (Dense)              (None, 2)                 130
=================================================================
Total params: 4,972,791
Trainable params: 4,961,187
Non-trainable params: 11,604
```

5.2 Anexo B: Distribución en el etiquetado de ChestX-Ray14



6 Referencias

Agarap, A. (2018). Deep Learning using Rectified Linear Units (ReLU). arXiv:1803.08375

Andrew Gabehart, R. (2003). Overwork and Stress: The Need for Policy on Working Hours in the Healthcare Professions. UCHC Graduate School Masters Theses. UCHC Graduate School Masters Theses 2003 - 2010.

Asociación Española Contra el Cáncer (2018). Evolución del cáncer de pulmón https://www.aecc.es/es/todo-sobre-cancer/tipos-cancer/cancer-pulmon/evolucion-cancer-pulmon (Último acceso 04/09/2019)

Brownlee, J. (2019). Deep Learning & Artificial Neural Networks https://machinelearningmastery.com/what-is-deep-learning/ (Último acceso 12/08/2019)

Bruno, M.; R. Duncan, J.; Bierhals, A.; Tappouni, R. (2018). Overnight Resident versus 24-hour Attending Radiologist Coverage in Academic Medical Centers. Radiology. vol. 289. https://doi.org/10.1148/radiol.2018180690

Bullock, J.; Cuesta, C.; Quera-Bofarull, A. (2018). XNet: A convolutional neural network (CNN) implementation for medical X-Ray image segmentation suitable for small datasets. Proceedings of SPIE Medical Imaging 2019: Biomedical Applications in Molecular, Structural, and Functional Imaging, vol. 10953. https://doi.org/10.1117/12.2512451

Chollet, F. (2017). Xception: Deep Learning with Depthwise Separable Convolutions. The IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 1800-1807. https://doi.org/10.1109/CVPR.2017.195

Chollet, F. (2017) Deep Learning with Python. Manning Publications Co.

Demner-Fushman, D.; Kohli, M.; B Rosenman, M.; E Shooshan, S.; Rodriguez, L.; Antani, S; R Thoma, G.; Mcdonald, C. (2015). Preparing a collection of radiology examinations for distribution and retrieval. Journal of the American Medical Informatics Association. vol. 23, pp. 304-310

Freidman, L. (2017). Lecture 3: Convolutional Neural Networks. MIT 6.S191 https://www.youtube.com/watch?v=v5JvvbP0d44 (Último acceso 04/09/2019)

Goodfellow, I. (2016). Master class on Deep Learning https://www.youtube.com/watch?v=vi7lACKOUao (Último acceso 04/09/2019)

Hartung, R. (2016). Membrane Potentials for beginners https://www.youtube.com/watch?v=VOh3pj0RsI0&t=499s (Último acceso 04/09/2019)

Heaton, J.; Goodfellow, I.; Bengio, Y.; Courville, A. (2017). Deep learning: The MIT Press, 2016, 800 pp, ISBN: 0262035618. Genetic Programming and Evolvable Machines. vol 19. https://doi.org/10.1007/s10710-017-9314-z

Hernandez-Garcia, A.; König, P. (2018). Further Advantages of Data Augmentation on Convolutional Neural Networks. 27th International Conference on Artificial Neural Networks, Rhodes, Greece. arXiv:1906.11052v1

Holländer, B. (2018). Accelerating Deep Learning Using Distributed SGD https://towardsdatascience.com/accelerating-deep-learning-using-distributed-sgd-an-overview-e66c4aee1a0c (Último acceso 25/08/2019)

Huang, G.; Li, Y.; Pleiss, G.; Liu, Z.; Hopcroft, E.; Weinberger, K. (2017): Snapshot Ensembles: Train 1, get M for free. arXiv:1704.00109

Ioffe, S.; Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. arXiv:1502.03167

Kingma, D.; Ba, J. (2014). Adam: A Method for Stochastic Optimization. International Conference on Learning Representations. arXiv:1412.6980

Krizhevsky, A.; Sutskever, I.; E. Hinton, G. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Neural Information Processing Systems (NIPS). vol 25. https://doi.org/10.1145/3065386

L. Causey, J.; Zhang, J.; Ma, S.; Jiang, B.; A. Qualls, J.; Politte, D.; Prior, F.; Zhang, S.; Huang, X. (2018). Highly accurate model for prediction of lung nodule malignancy with CT scans. Scientific Reports. vol 8. https://doi.org/10.1038/s41598-018-27569-w

Li, H.; Xu, Z.; Taylor, G.; Goldstein, T. (2018): Visualizing the Loss Landscape of Neural Nets. Intro to optimization in deep learning. Advances in Neural Information Processing Systems (NIPS) vol. 31, pp. 6389-6399

Litjens, G.; Kooi, T.; Ehteshami Bejnordi, B.; Setio, A.; Ciompi, F.; Ghafoorian, M.; van der Laak, J.; van Ginneken, B.; I. Sánchez, C. (2017). A Survey on Deep Learning in Medical Image Analysis. Medical Image Analysis. arXiv:1702.05747v2

London: Royal college of radiologists (2008). Standards for providing a 24-hour interventional radiology service. BFCR. vol. 27. https://doi.org/10.1007/s00270-016-1299-0

Maklin, C. (2019). Dropout Neural Network Layer In Keras Explained https://towardsdatascience.com/machine-learning-part-20-dropout-keras-layers-explained-8c9f6dc4c9ab (Último acceso 04/09/2019)

Narkhede, S. (2018). Understanding AUC – ROC Curve https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5 (Último acceso 04/09/2019)

Oakden-Rayner, L. (2017). Exploring the ChestXray14 dataset, problems https://lukeoakdenrayner.wordpress.com/2017/12/18/the-chestxray14-dataset-problems/ (Último acceso 22/08/2019)

Oskolkov, N. (2019). Deep Learning for Clinical Diagnostics. https://mc.ai/deep-learning-for-clinical-diagnostics/ (Último acceso, 04/09/2019)

Razzak, M.; Naz, S.; Zaib, A. (2018). Deep Learning for Medical Image Processing: Overview, Challenges and the Future. arXiv:1704.06825

Saha, S. (2018). A Comprehensive Guide to Convolutional Neural Networks https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 (Último acceso 04/09/2019)

Shao, Y.; Gao, Y.; & Guo, Y.; Shi, Y.; Yang, X.; Shen, D. (2014). Hierarchical Lung Field Segmentation With Joint Shape and Appearance Sparse Learning. Medical Imaging, IEEE Transactions. vol 33. https://doi.org/ 10.1109/TMI.2014.2305691

Sharma, S. (2017). Activation Functions in Neural Networks: Sigmoid, tanh, Softmax, ReLU. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6 (Último acceso 02/09/2019)

Shen, D.; Wu, G.; Suk, H. (2017). Deep Learning in Medical Image Analysis. Annual review of biomedical engineering. Annual review of biomedical engineering, 2017. https://doi.org/10.1146/annurev-bioeng-071516-044442

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research. vol. 15, pp. 1929-1958

Tajbakhsh, N; Shin, J.; Gurudu, S.; Todd, R.; Kendall, C.; Gotway, M.; Liang, J. (2016). Convolutional Neural Networks for Medical Image Analysis: Fine Tuning or Full Training?. IEEE Transactions on Medical Imaging. vol. 35, pp. 1299-1312

Tsetis, D.; Uberoi, R.; Fanelli, F.; Roberston, L.; Krokidis, M.; van Delden, O.; Radeleff, B.; Müller-Hülsbeck, S.; Szerbo-Trojanowska, M.; Lee, M.; Morgan, R.; Brountzos, E.; Maria Belli, A. (2016). The Provision of Interventional Radiology Services in Europe: CIRSE Recommendations. CardioVascular and Interventional Radiology. vol 39. https://doi.org/10.1007/s00270-016-1299-0

Wang, X.; Peng, Y.; Lu, L.; Lu, Z.; Bagheri, M.; Summers, R. (2017). ChestX-ray8: Hospital-scale Chest X-ray Database and Benchmarks on Weakly-Supervised Classification and Localization of Common Thorax Diseases. arXiv:1705.02315.

Wang, X.; Mao, K.; Wang, L.; Yang, P.; Lu, D.; He, P. (2019). An Appraisal of Lung Nodules Automatic Classification Algorithms for CT Images. Sensors. 19, vol 1. 194. https://doi.org/10.3390/s19010194

World Health Organization, (2018). All types of cancer fact sheet https://gco.iarc.fr/today/data/factsheets/cancers/39-All-cancers-fact-sheet.pdf (Último acceso 04/09/2019)

Yosinski, J.; Clune, J.; Bengio, Y.; Lipson, H. (2014). How transferable are features in deep neural networks?. Advances in Neural Information Processing Systems (NIPS). vol. 27, pp. 3320–3328