Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Clasificación de imágenes usando redes neuronales convolucionales en Python

Autor: Álvaro Artola Moreno
Tutor: José Antonio Pérez Carrasco

Dpto. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Clasificación de imágenes usando redes neuronales convolucionales en Python

Autor:
Álvaro Artola Moreno

Tutor:
José Antonio Pérez Carrasco
Profesor Contratado Doctor

Dpto. de Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2019

Trabajo Fin de Grado: Clasificación de imágenes usando redes neuronales convolucionales en Python

Autor: Álvaro Artola Moreno
Tutor: José Antonio Pérez Carrasco

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2019

El Secretario del Tribunal

A mis padres

Agradecimientos

A vosotros mis padres, Álvaro y Macarena que lo sois todo para mi y que os estaré eternamente agradecido pues sin vuestro apoyo esta meta hubiera sido imposible alcanzarla.

En especial a ti papá, que desgraciadamente no puedes estar aquí acompañandome en estos momentos de felicidad. Gracias por haber sido el mejor padre que haya podido tener. Gracias por transmitirme tu tesón, fortaleza y fuerza de voluntad. Te echaré siempre de menos.

A mis profesores por transmitirme la motivación de ser ingeniero y la pasión por las nuevas tecnologías. Gracias especialmente a mi tutor José Antonio por darme la posibilidad de adentrarme en un concepto que hasta ahora era totalmente nuevo para mí y que con su ayuda he logrado entender y llevar a la práctica aun no siendo nada sencillo.

Álvaro Artola Moreno

Sevilla, 2019

Resumen

En los último años se han generalizado en nuestro lenguaje y en numerosas acciones cotidianas que desarrollamos cuando accedemos a Internet o utilizamos dispositivos tecnológicos los conceptos de Inteligencia Artificial (IA), *Machine Learning* (ML) y *Deep Learning* (DL).

La Inteligencia Artificial (*Artificial Intelligence* o AI) "*es una rama de las ciencias computacionales encargada de estudiar modelos de cómputo capaces de realizar actividades propias de los seres humanos en base a dos de sus características primordiales: el razonamiento y la conducta*" [1].

De la IA surgen dos ramas más específicas que son el ML primero y el DL. Cuando hablamos de *Machine Learning*, nos referimos al aprendizaje automático que trata de crear programas capaces de generalizar comportamientos a partir de una información suministrada en forma de ejemplos. Esta capacidad de aprender y de anticipar comportamientos tiene múltiples usos como sistemas de reconocimiento facial, aprendizaje de idiomas o la posibilidad de generar diagnósticos médicos [2].

Con *Deep Learning* nos referimos a una función de la inteligencia artificial que imita el funcionamiento del cerebro humano en el procesamiento de datos y la creación de patrones para su uso en la toma de decisiones. Se compone de redes capaces de aprender sin supervisión a partir de datos sin estructurar o etiquetar [3].

El objetivo del presente proyecto es el de estudiar dichas redes neuronales, concretamente las redes neuronales convolucionales y la importancia que tienen en el análisis de imágenes.

Centrándonos en el objetivo de entrenar la red para conseguir un elevado porcentaje de precisión, sensibilidad y especificidad, se desarrollará una investigación de los lenguajes de programación más utilizados en el campo de la Inteligencia Artificial, bases de datos para explotar la información y el funcionamiento propio de las Redes Neruonales Convolucionales.

Abstract

In recent years, concepts such as Artificial Intelligence – AI –, Machine Learning – ML – and Deep Learning – DL – have become widespread in our language and in numerous daily actions that we develop when we access the Internet or use technological devices.

AI are the processes by which a machine imitates the cognitive functions that humans associate with other human minds, such as learning and solving problems.

From the AI, two more specific branches emerge: ML first and DL. When we talk about Machine Learning we are referring to create programs that are able to generalize behaviors from information provided by examples. This ability to learn and anticipate behaviors has multiple uses ranging from the most generalist, as facial recognition systems or the ability to respond and learn languages to more specialized uses such as the possibility of generating medical diagnoses.

On the other hand, Deep Learning is referring to an artificial intelligence function that imitates the workings of the human brain in preprocessing data and creating patterns for use in decision making.

The aim of the project is to study these neural networks, specifically the convolutional ones and the importance they have in image analysis.

Focusing on the objective of training the network to achieve a high percentage of precision, sensitivity and specificity, a research will be carried out on the most used programming languages in the AI field. Moreover, the most famous datasets used to exploit information and the Convolutional Neural Networks behavior will be investigated too.

Índice

Índice de Tablas

Índice de Figuras

Notación

CNN	Red neuronal convolucional
ML	Aprendizaje automatizado
AI	Inteligencia Artificial
DL	Aprendizaje profundo
CPU	Unidad de procesamiento central
GPU	Unidad de procesamiento gráfico
CUDA	Arquitectura Unificada de Dispositivos de Cómputo
ANS	Redes Neuronales Artificiales
RAM	Memoria de Acceso Aleatorio

1 INTRODUCCIÓN

Alguien inteligente aprende de la experiencia de los demás

-Voltaire-

Desde sus inicios, trabajar en modelos de computación conexionista o redes neuronales artificiales, referidas comúnmente como "redes neuronales" ha venido motivado por el hecho de apreciar que el cerebro humano computa de una forma análoga a los ordenadores. La mente humana es un ordenador altamente complejo, no lineal y paralelo – un sistema de procesamiento de la información – que tiene la capacidad de organizar sus constituyentes estructurales conocidos como neuronas con la finalidad de llevar a cabo ciertos cálculos: tareas de reconocimiento de patrones, control de dispositivos, clasificación de objetos, etc. a una velocidad mucho mayor que cualquier ordenador ordinario. Pongamos como ejemplo, la visión humana. Su función principal consiste en proporcionar una representación del entorno que nos rodea, y lo que es más importante, nos proporciona la información que necesitamos para interactuar con dicho entorno. Para ser más específicos, la mente rutinariamente desarrolla tareas de reconocimiento – reconocer una cara familiar en un escenario desconocido – en aproximadamente 100-200 ms mientras que tareas de mucha menos complejidad tardan hasta días en ser resueltas por cualquier ordenador convencional.

De estas observaciones surge la cuestión que supuso el nacimiento de las Redes Neuronales Artificiales o ANS (*Artificial Neural Systems*). ¿Qué parámetros hacen posible la supremacía del cerebro en unas determinadas tareas frente a los computadores?

Resulta llamativo saber que las neuronas son mucho más simples, lentas y menos fiables que una CPU, y sin embargo existen tareas que el cerebro resuelve eficazmente – respuestas ante estímulos del entorno, reconocimiento del habla, etc. – mientras que son difícilmente abordables por un ordenador.

En definitiva, la idea que subyace en las ANS es la de abordar todos los problemas que el cerebro humano resuelve con rapidez y eficiencia. Para ello, puede resultar conveniente construir sistemas que simulen la estructura de las redes neuronales biológicas con la finalidad de alcanzar una funcionalidad similar.

1. Objetivos

El objetivo principal detrás de este Trabajo Fin de Grado es el de estudiar el funcionamiento interno de las redes neuronales convolucionales y especialmente su utilidad en el análisis de imágenes. Para ello, se centrará en la consecución de los siguientes puntos:

El estudio de la neurona desde una inspiración biológica de manera que pueda comprenderse el mecanismo que estas llevan asociado y el cual desarrollan para llevar a cabo la tarea de aprender.

Un análisis del modelo de neurona artificial y la evolución hasta las redes neuronales convolucionales.

El análisis de los lenguajes de programación más destacados en la inteligencia artificial y el por qué de la elección de Python para el proyecto.

Una descripción de los conjuntos de datos más famosos para entrenar redes neuronales destacando aquellos relacionados con la medicina.

A partir de la consecución de estos puntos y mediante la comprensión de las bibliotecas y tecnologías implementadas en el desarrollo del aplicativo como son Keras y TensorFlow, se pretenden entrenar dos redes neuronales convolucionales distintas y ver qué porcentaje de precisión se obtiene con cada una de ellas al pasarle como entrada distintos conjuntos de datos.

Se aspira a que el porcentaje de acierto obtenido sea elevado y que la red evalue con precisión.

2 REDES NEURONALES

Todo hombre puede ser, si se lo propone, escultor de su propio cerebro.

- Santiago Ramón y Cajal-

Antes de llevar a cabo un estudio pormenorizado de las redes neuronales convolucionales y la repercusión que estas pueden llegar a tener en los estudios sobre distintas imágenes, resulta conveniente y de utilidad conocerlas de manera más exhaustiva.

Inspiración biológica

El sistema nervioso humano puede ser visto como un sistema de tres etapas, tal y como se muestra en la figura 1. En el centro se encuentra el cerebro representado por la red neuronal, que continuamente recibe información, la trata y a partir de esta toma determinadas decisiones. Los receptores convierten tanto los estímulos del cuerpo humano como los del entorno en impulsos eléctricos que transmiten información a la red neuronal – el cerebro –.

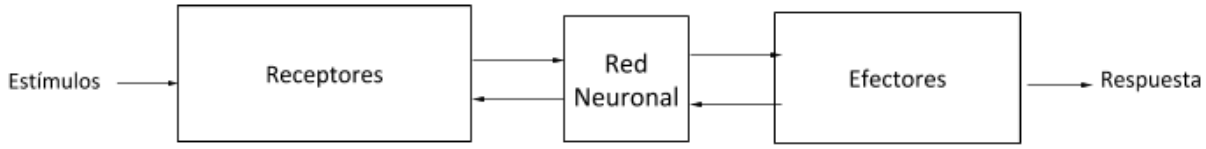


Figura 1 – Esquema simple de red neuronal.

La dificultad de entender el cerebro fue mucho menor gracias al trabajo pionero de Ramón y Cajal (1911), quien describió las neuronas como las estructuras constituyentes del cerebro. Generalmente, las neuronas son unos cinco o seis órdenes de magnitud más lentas que las puertas lógicas de silicio; los eventos en un chip de silicio ocurren en un rango de unos 10^{-9} segundos – nanosegundos – mientras que los eventos neuronales ocurren en el rango de los 10^{-3} segundos – milisegundos –. Sin embargo, el cerebro compensa esta diferencia gracias a la gran cantidad de neuronas interconectadas entre ellas. Se estima que aproximadamente hay 10 billones de neuronas en el cortex cerebral humano y 60 trillones de sinapsis o conexiones. La red resultante es una enorme estructura muy eficiente. Específicamente hablando, la eficiencia energética del cerebro se corresponde con aproximadamente 10^{-16}Julios – J – por operación por segundo mientras que el valor correspondiente en los mejores computadores de hoy en día se encuentra en torno a los 10^{-6} Julios por operación por segundo [4].

2.1.1 Anatomía de una neurona

Una neurona típica recoge señales procedentes de otras neuronas a través de un conjunto de delicadas estructuras llamadas dendritas. La neurona emite impulsos de actividad eléctrica a lo largo de una fina y larga fibra denominada axón, que se escinde en millones de ramificaciones. El órgano de cómputo es el soma. Lo interesante es conocer cómo se comunican las neuronas entre sí.

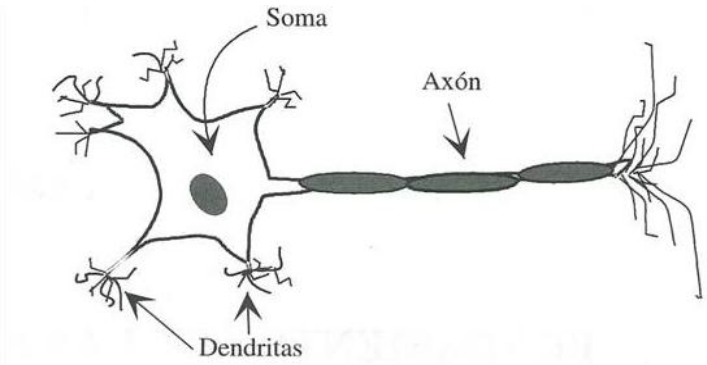


Figura 2 – Imagen de una neurona [4].

La sinapsis – del griego ύναψις [sýnapsis], 'unión', 'enlace'– es una aproximación intercelular especializada entre neuronas,ya sea entre dos neuronas de asociación, una neurona y una célula receptora o entre una neurona y una célula efectora. A pesar de que una sola neurona es capaz de realizar numerosas tareas por si misma, lo más interesante como ha sido comentado anteriormente, surge entre la interacción entre ellas [4].

2.1.2 Modelo de una neurona artificial

Los tres conceptos clave que una neurona artificial debe emular son [7]:

- Procesamiento paralelo
- Memoria distribuida
- Adaptabilidad al entorno

2.1.3 Modelo genérico de una neurona artificial

En general el modelo de una neurona artificial puede representarse de la siguiente forma:

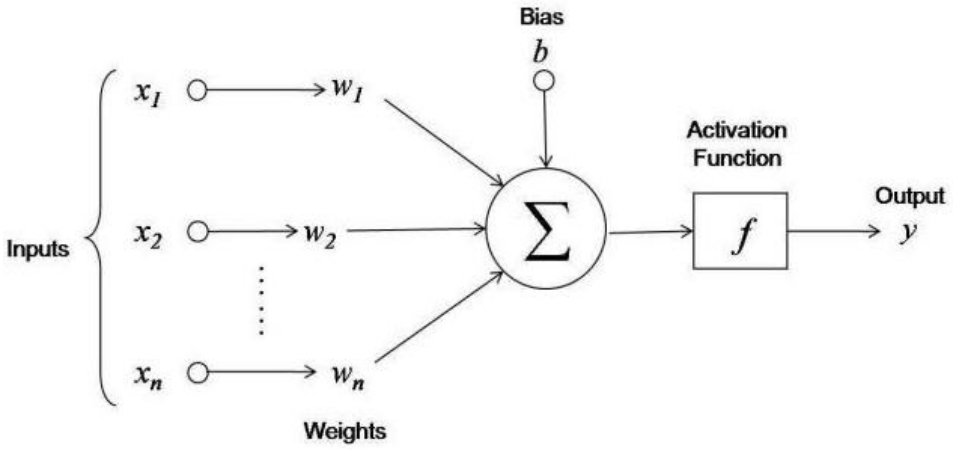


Figura 3 – modelo genérico de una neurona artificial [4].

En él podemos apreciar:

- Un conjunto de **sinapsis** o conectores, cada uno de los cuales se caracteriza por un peso o longitud propia. A diferencia de una sinapsis cerebral, el peso sináptico de una neurona artificial puede tomar un rango de valores que incluye tanto valores positivos como negativos.
- La **función de red** $\sum$ – de propagación – es la encargada de realizar la suma de las señales de entrada, ponderado por las respectivas sinapsis de la neurona; las operaciones descritas constituyen un combinador linear.

- Una **función de activación** para limitar la amplitud de la salida de una neurona. También se denomina función de aplastamiento pues limita el rango de amplitud permitido en la salida a un valor finito.
- La **salida,** que calcula la salida de la neurona en función de la activación de la misma. Normalmente únicamente se aplica la función identidad, y se toma como salida el valor de activación.

Por norma general, el rango de amplitud normalizada a la salida de una neurona se corresponde con el intervalo cerrado [0,1] o, de manera alternativa [-1,1]

2.1.4 Función de activación

La función de activación, denotada por y define la salida de la neurona en términos de campo local inducido. Generalmente, se considera determinista y en la gran mayoría de los modelos es monótona creciente y continua. La forma $y = f(x)$ de las funciones de activación más empleadas en los Artificial Neural Systems (ANS) se muestran en la siguiente tabla donde x representa el potencial postsináptico e y el estado de activación.

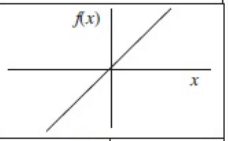
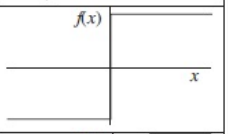
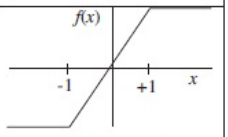
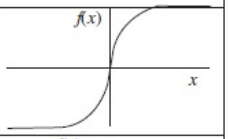
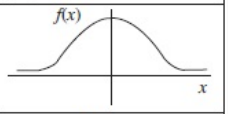
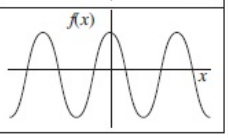
	Función	Rango	Gráfica
Identidad	$y = x$	$[-\infty,+\infty]$	
Escalón	$y = sign(x)$ $y = H(x)$	$\{-1,+1\}$ $\{0,+1\}$	
Lineal a tramos	$y = \begin{cases} -1, si\ x < -l \\ x, si + l \le x \le -l \\ +1, si\ x > +l \end{cases}$	$[-1,+1]$	
Sigmoidea	$y = \frac{1}{1+e^{-x}}$ $y = tgh(x)$	$[0,+1]$ $[-1,+1]$	
Gaussiana	$y = Ae^{-Bx^2}$	$[0,+1]$	
Sinusidal	$y = A\ sen(\omega x + \varphi)$	$[-1,+1]$	

Tabla 1 – Funciones de activación [5].

La **función sigmoidal o logística** es probablemente la función de activación más empleada por su plausibilidad fisiológica [4].

2.1.5 Redes síncronas y asíncronas

Es importante destacar la forma en la que las unidades computan su activación en relación al tiempo. Si en cada ciclo de aprendizaje se calcula la activación de todas las unidades de una capa, hablamos de redes síncronas. En caso contrario, se trata de redes asíncronas o probabilísticas. En ellas, cada unidad de proceso tiene una cierta posibilidad de computar su activación en cada ciclo de aprendizaje [8].

2.1.6 Métodos deterministas y estadísticos

La distinción entre métodos deterministas y métodos estadísticos está muy relacionada con la de las redes síncronas o asíncronas.

Las redes síncronas suelen emplear reglas de aprendizaje deterministas, mientras que los métodos estadísticos se aplican en las redes asíncronas.

Los métodos estadísticos, sin embargo, hacen cambios pseudoaleatorios en las conexiones y retienen los cambios sólo si mejoran la respuesta del sistema [8].

Arquitectura de las redes neuronales

2.2.1 Según el número de capas

- Redes neuronales monocapa y multicapa [6]:

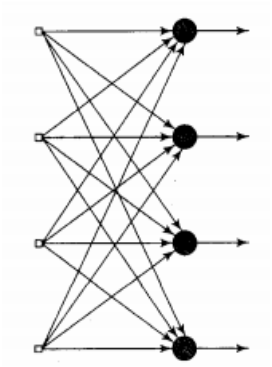


Figura 4 – Red neuronal monocapa [4].

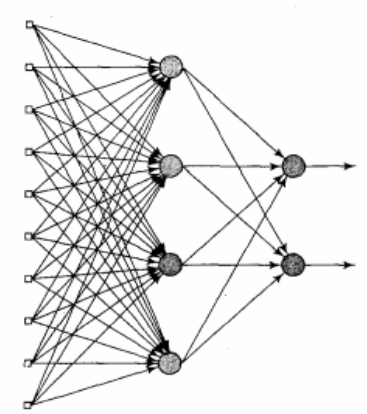


Figura 5 – Red neuronal multicapa [4].

2.2.2 Según la realimentación

- Redes neuronales no recurrentes: en este tipo de redes, la propagación de las señales se produce en un único sentido.
- Redes neuronales recurrentes: de manera análoga a las anteriores, estas redes se caracterizan por poseer lazos de realimentación. Dichos lazos, pueden ser entre neuronas de diferentes capas, de la misma capa o incluso de una neurona consigo misma [6].

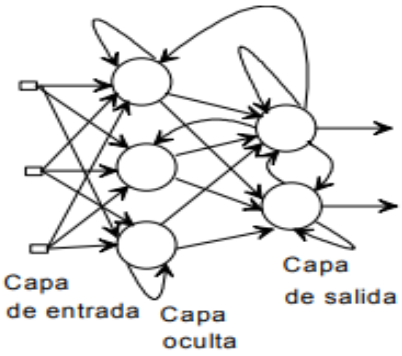


Figura 6 – Red neuronal recurrente [4].

2.2.3 Según el grado de conexión

- Redes neuronales totalmente conectadas: en ellas, todas las neuronas de una capa se encuentran conectadas con las de la capa siguiente –redes no recurrentes – o con las de la anterior – redes recurrentes –.
- Redes neuronales parcialmente conectadas: en esta ocasión, no se da la conexión total entre neuronas de diferentes capas. Esto ocurre cuando alguna de las sinapsis se pierde.

Modos de operación

Durante la operatoria de una red neuronal pueden distinguirse claramente dos fases o modos de operación: la fase de aprendizaje o entrenamiento y la fase de operación o ejecución.

2.3.1 Fase de aprendizaje

Por norma general, el proceso de aprendizaje parte de un conjunto de pesos sinápticos aleatorio y busca un conjunto de pesos que permitan a la red desarrollar correctamente una determinada tarea. Durante el proceso se va refinando iterativamente la solución hasta alcanzar un nivel de operación suficientemente aceptable [5].

El entrenamiento o aprendizaje, se puede desarrollar a dos niveles:

a) A través del modelado de las sinápsis.

b) A través de la creación o destrucción de neuronas.

Los algoritmos de aprendizaje se basan normalmente en métodos numéricos iterativos que tratan de reducir al máximo la función de coste. En ocasiones, esto genera problemas en la convergencia del algoritmo. Rigurosamente hablando, la convergencia supone una comprobación de una determinada arquitectura, pues junto con su regla de aprendizaje es capaz de resolver un determinado problema.

Los tipos de aprendizaje que pueden distinguirse son [9]:

- Supervisado.
- No supervisado.
- Híbrido.
- Reforzado.

2.3.1 Fase de recuerdo

Toda vez que el sistema ha sido satisfactoriamente entrenado, el aprendizaje se desconecta. Esto quiere decir, que los pesos y la estructura de la red quedan fijos, quedando la red neuronal dispuesta para el procesamiento de datos. Una de las principales ventajas de las redes neuronales es que la red en sí aprende la relación existente entre los datos, adquiriendo la capacidad de generalizar conceptos. De esta forma, una red neuronal tiene la habilidad de tratar con información desconocida para ella durante la fase de entrenamiento [5].

3 REDES NEURONALES CONVOLUCIONALES

"Nunca vayas por el camino trazado, porque conduce hacia donde otros ya han estado".

-Alexander Graham Bell -

Hasta ahora, se ha realizado una breve descripción de las redes neuronales más simples, sin embargo, el proyecto se centra en redes que van más allá de las características del perceptrón simple o el multicapa. Surge entonces, la necesidad de trabajar con las redes neuronales convolucionales –CNN–.

El objetivo de este tercer capítulo es el de a partir de los conocimientos adquiridos en el apartado anterior, comprender la necesidad de evolucionar a este tipo de redes, pues son el algoritmo utilizado en el *Machine Learning*.

Introducción

La CNN es un tipo de Red Neuronal Artificial que procesa sus capas imitando al cortex visual del cerebro humano para identificar distintas características en las entradas. Para ello, la CNN contiene varias capas ocultas especializadas y con una jerarquía: esto significa que las primeras capas detectan propiedades o formas básicas y se van especializando hasta llegar a capas más profundas capaces de reconocer formas complejas como un rostro o una silueta [10].

La red neuronal por sí misma ha de reconocer una gran cantidad de imágenes para que la pueda captar las características únicas de cada objeto y a su vez poder generalizarlo. Cada imagen se trata de una matriz de pixeles cuyo valor va de 0 a 255 pero se normaliza para la red neuronal de 0 a 1, como podrá observarse en la aplicación del proyecto.

Como punto de partida la red toma como entrada los píxeles de una imagen. En el caso de nuestro proyecto, las imágenes serán en escala de grises y las entradas serán de 50x50 píxeles de alto y ancho, por lo que habrá en total 2500 neuronas. En caso de que dispusiéramos de una imagen a color serían necesarios tres canales distintos: *red*, *green* y *blue*, utilizando entonces un total de 50x50x3 = 7500 neuronas de entrada para la capa de entrada de la red [11].

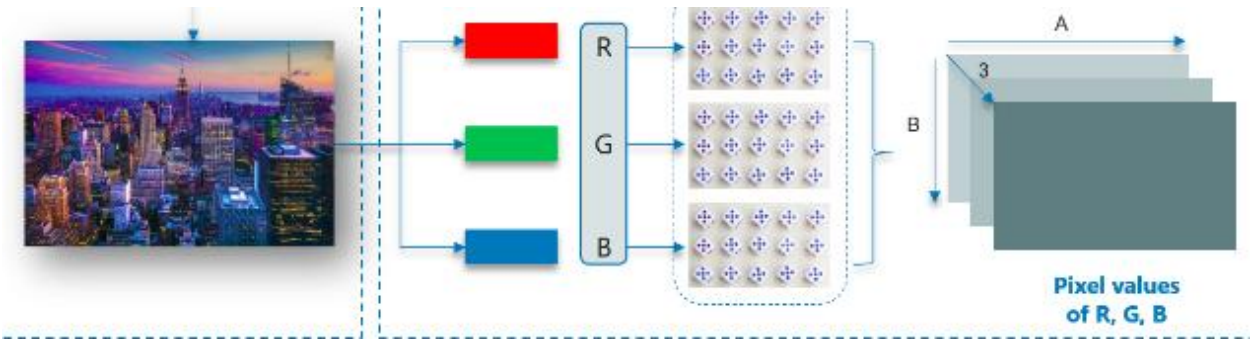


Figura 7 – División de la imagen original [11].

Antes de que la red sea alimentada, es conveniente normalizar los valores, tal y como se comentó anteriormente. Los valores de los pixels oscilan entre 0 y 255 por lo que se llevará a cabo una transformación de cada pixel en la forma: valor del pixel / 255 resultando siempre un valor entre cero y la unidad.

Tras estos primeros pasos comienza el procesado distintivo de las CNN, lo que se conoce como las convoluciones. Como tal, una convolución consiste en tomar grupos de píxeles cercanos de la imagen de entrada e ir operando matemáticamente contra una pequeña matriz a la que se denomina *kernel* conocida como filtro. Ese *kernel* recorre todas las neuronas de entrada – de izquierda a derecha y de arriba hacia abajo – y genera una nueva matriz de salida, que será la nueva capa de neuronas ocultas, y que también se conoce como la matriz de activación. La convolución será tal si y sólo si el kernel es real y simétrico [12].

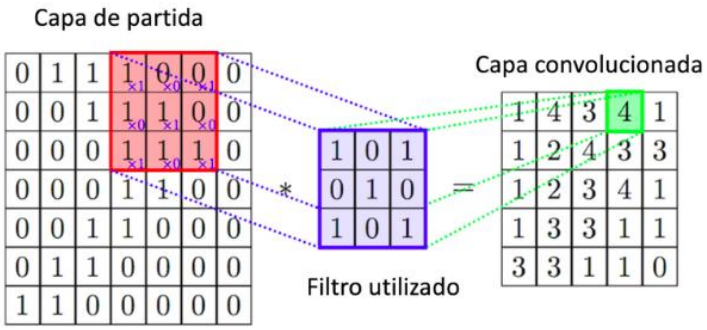


Figura 8 – Uso del filtro en la convolución [12].

Cabe destacar que durante las convoluciones no se aplica un solo filtro si no que habrá un conjunto de estos. Si por ejemplo se escogen 32 filtros, se obtienen en total 32 matrices de salida. A este conjunto se le denomina *feature mapping* .

A medida que el kernel se va desplazando obtenemos una nueva imagen filtrada por este. Una vez la imagen realiza una convolución con un *kernel*, aplica la función de activación. Esta función se encarga de devolver una salida a partir de un valor de entrada, normalmente en un rango determinado como (0,1) o (-1,1). En general, se buscan funciones cuyas derivadas sean simples para minimizar así el coste computacional.

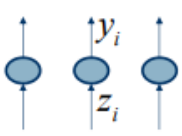
Funciones de activación

Durante el apartado 2.1.4 de desarrolló una descripción de las principales funciones de activación en las redes neuronales artificiales. Ahora, una vez se conocen las características de estas, se procederá a describir cómo las Redes Neuronales Convolucionales las utilizan, y en especial, las del Proyecto en si.

3.1.1 Softmax

La función softmax (función exponencial normalizada) es una generalización de la función sigmoidea y popularizada por las redes neuronales convolucionales. Se utiliza como función de activación de salida para la clasificación multiclase porque escala las entradas precedentes de un rango entre 0 y 1 y normaliza la capa de salida, de modo que la suma de todas las neuronas de salida sea igual a la unidad [13].

Se considera a esta función como una distribución de probabilidad categórica, lo que le permite comunicar un grado de confianza en las predicciones de su clase. Matemáticamente:



$$y_i = \frac{e^{z_i}}{\sum_{j \in group} e^{z_j}} \quad (3\text{-}1)$$

$$\frac{\partial y_i}{\partial z_i} = y_i(1 - y_i) \quad (3\text{-}2)$$

3.1.2 ReLu

La unidad lineal rectificada, del inglés Rectified Linear Unit (ReLu) es la función de activación más utilizada en los modelos de aprendizaje profundo. La función devuelve 0 si recibe una entrada negativa, pero para cualquier valor positivo x devuelve ese valor. Por lo tanto, se puede escribir como $f(x) = \max(0, x)$ [14].

Gráficamente se ve así:

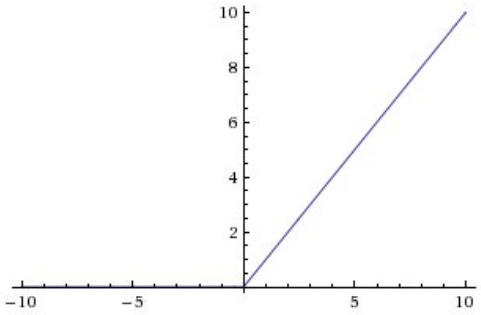


Figura 9 – Función ReLu [13].

Maxpooling

Una vez terminado el proceso anterior, se reduce la cantidad de neuronas antes de realizar una nueva convolución. Como se comentó anteriormente, partiendo de una imagen de 50x50 tenemos una primera capa de entrada de 2500 neuronas y justo tras la primera convolución obtenemos una capa oculta de 80.000. Si se llevara a cabo una nueva convolución a partir de esta capa, el número de neuronas de la próxima capa crecería exponencialmente implicando un mayor procesamiento. Para reducir el tamaño de la próxima capa se realiza el proceso de subsampling en el que se reduce el tamaño de las imágenes filtradas, pero donde prevalecen las características más importantes que detectó cada filtro.

Hay diversos tipos de subsampling, pero el más utilizado es el Max-Pooling. Si suponemos que se realiza una *Max-pooling* de tamaño 2x2 se recorrerán cada una de las 32 imágenes de características obtenidas anteriormente de 50x50 pixeles de izquierda a derecha, arriba-abajo, pero en lugar de a un solo pixel se toman de 2x2 – 2 de alto por 2 de ancho – y se preserva el valor más elevado de esos 4 pixeles, de ahí el termino *Max*. En este caso, usando 2x2 la imagen resultante se reduce a la mitad y quedará una de 25x25 pixeles. Después de este proceso de subsampling quedan 32 imágenes de 25x25, pasando de haber tenido 80.000 neuronas a 20.000. El descenso es considerable y teóricamente almacenan la información más importante para detectar las características deseadas [15].

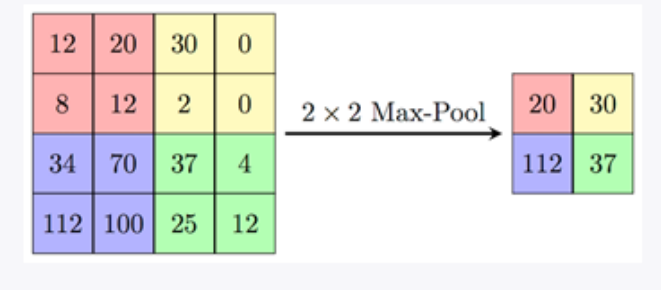


Figura 10 – Maxpooling [15].

La primera convolución es capaz de detectar características primitivas como líneas o curvas. A medida que se van realizando más capas con las convoluciones, los mapas de características serán capaces de reconocer formas más complejas.

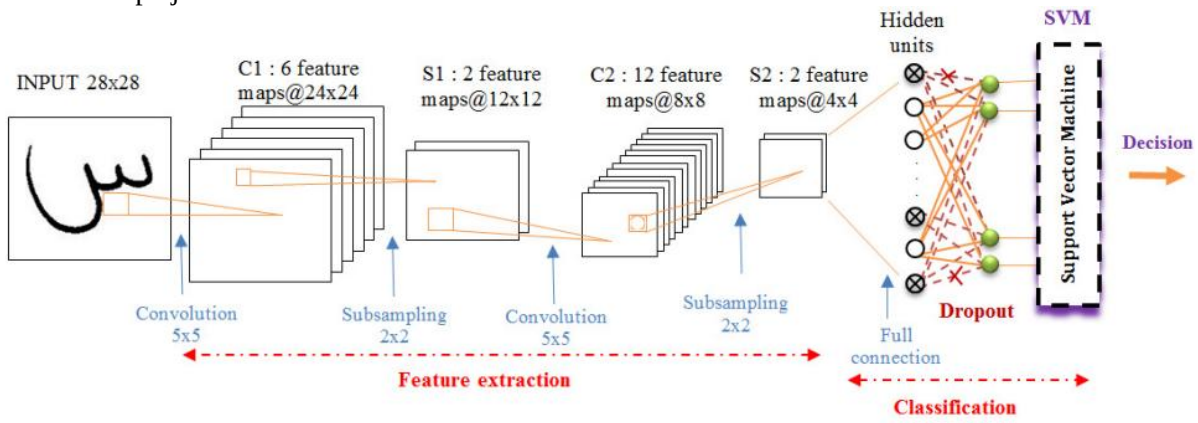


Figura 11 – Capas subsampling [18].

Como punto de finalización se toma la última capa oculta a la que se le realizó el proceso de subsampling, que se dice que es tridimensional y se procesa para que deje de serlo pasando a ser una capa de neuronas tradicionales. Entonces a esta última se le aplica la función de activación *Softmax* que conecta contra la capa de salida final que tendrá la cantidad de neuronas correspondientes con las clases que estamos clasificando.

Las salidas en el momento del entrenamiento tendrán el formato conocido como *one-hot-encoding* que resulta un grupo de bits cuyas combinaciones de valores válidas son solo aquellas en las que si aparece un bit a uno, el resto ha de estar a cero [19].

Si hablamos de imágenes de lesiones de piel benignas y malignas tendríamos: [1,0] y [0,1] respectivamente tras lo cual, la función Softmax se encarga de trasladar dichos valores a probabilidad a las neuronas de salida. Por tanto, si se obtiene una salida [0’2 0’8] esto quiere decir que con un 20% de probabilidad la lesión será benigna y con un 80% maligna.

Backpropagation

Backpropagation – retropropagación en español– [4] consiste en un método de cálculo que emplea un ciclo de propagación. Una vez aplicado un patrón a la entrada de la red como estímulo, este se propaga desde la primera capa a través de las capas siguientes de la red hasta generar una salida. La señal de salida se compara con la salida esperada y se calcula una señal de error para cada una de las salidas.

A medida que se entrena la red, las neuronas de las capas intermedias aprenden a reconocer las diferentes características del espacio total de la entrada. Tras el entrenamiento, cuando se presenta un patrón de entrada arbitrario que contenga ruido o que esté incompleto, las neuronas de la capa oculta responden con una salida activa si la nueva entrada contiene un patrón que se asemeje a las características que las neuronas individuales han aprendido a reconocer durante el entrenamiento.

Dropout

Dropout [21] es un método que desactiva un número determinado de neuronas en una red neuronal de forma aleatoria Las neuronas desactivadas no se tienen en cuenta para la propagación hacia delante ni para atrás, lo que obliga a las neuronas cercanas a no dependert tanto de las neuronas desactivadas. Este hecho ayuda a reducir el *overfitting* con lo que las neuronas necesitan trabajar mejor de forma solitaria para no depender tanto de las relaciones con las neuronas vecinas.

Dropout tiene un parámetro que indica la probabilidad de que las neuronas se queden o no activadas y que toma valores de 0 a 1. Cuando los valores son cercanos a 0 el *dropout* desactivará menos neuronas que si por el contrario, son cercanos a 1. Cabe destacar que el método *dropout* solo se usa durante la fase de entrenamiento.

Overfitting y underfitting

Las principales causas de obtener malos resultados en *Machine Learning* son el *overfitting* y el *underfitting* de los datos [22]. Cuando entrenamos un modelo la misión principal es hacer encajar – *fit* – los datos de entrada entre ellos y por supuesto, con la salida. Podemos traducir estos términos como sobreajuste y subajuste respectivamente. Hacen referencia al fallo en el modelo al generalizar o encajar el conocimiento que pretendemos que adquiera.

Si los datos de entrenamiento son mínimos la máquina será incapaz de generalizar el conocimiento y estará incurriendo en el *underfitting*. Por el contrario, si la máquina es entrenada con valores o imágenes demasiado específicas sobre lo que se quiere analizar será incapaz de discernir si se trata de una imagen, carácter, etc. u otro. Ambas situaciones son negativas puesto que la máquina no es capaz de generalizar el conocimiento y no será capaz de dar buenas predicciones.

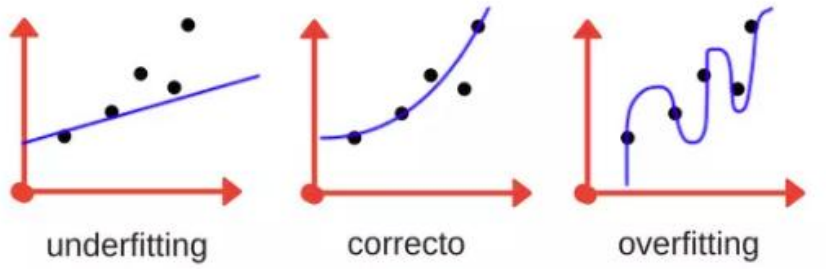


Figura12 – underfitting, correcto y overfitting [23].

4 REDES NEURONALES CONVOLUCIONALES PROFUNDAS

"El primer paso de la ignorancia es presumir de saber".

-Baltasar Gracián -

El diseño de las redes neuronales convolucionales profundas no se trata de una tarea sencilla, pues no existe un patrón definido para establecer el número de capas, los tipos de las mismas o las conexiones entre ellas. Además, tampoco existe un proceso que ayude en su definición.

Actualmente, y como solución a estos inconvenientes, lo que se hace es utilizar aquellas redes que han demostrado ser precisas, efectivas y con una tasa de acierto elevada en el reconocimiento de imágenes.

A continuación se van a detallar algunas de las más famosas.

LeNet

Desarrollada por Yann LeCun y sus colaboradores, fueron capaces de implementar una red neuronal convolucional capaz de detectar caracteres haciendo uso de los conceptos de *backpropagation* y *feedforward* [24]. Entre sus características más llamativas destacan:

- Gran canidad de capas escondidas.
- Numerosos mapas de unidades replicadas en cada capa.
- Agrupación de las salidas cuyas unidades replicadas estuvieran cercanas.

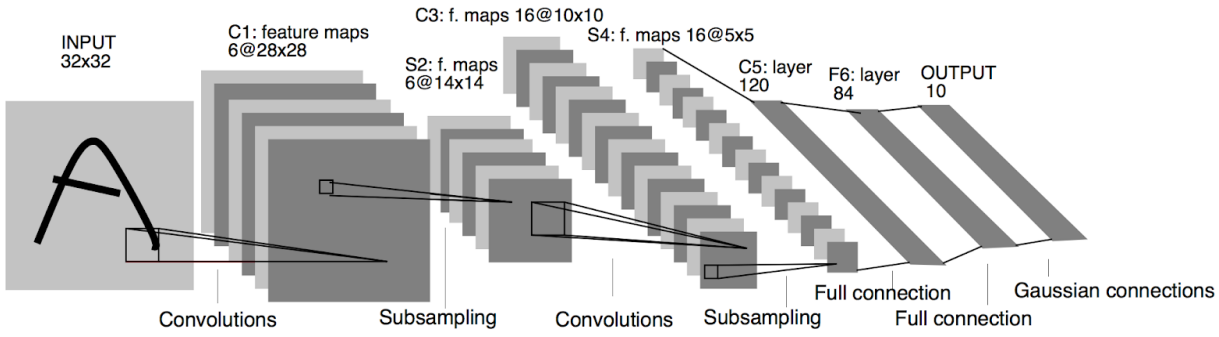


Figura 13 – LeNet [24].

AlexNet

AlexNet [25] ha tenido un gran impacto en el campo de *machine learning*, y especialmente en la aplicación del aprendizaje profundo al *machine vision*. Tiene una arquitectura muy similar a LeNet, pero se trata de una red más profunda, con mas filtros por capa y con capas convolucionales apiladas. Los puntos clave de esta red son:

- Uso de la función de activación ReLU en lugar de Tanh para no añadir linealidad. Esto hace que el proceso se acelere unas seis veces con la misma precisión

- Uso de *dropout* en lugar de regularización para tratar el *overfitting*. Sin embargo, el tiempo de entrenamiento se dobla con una tasa del 0.5 de *dropout.*

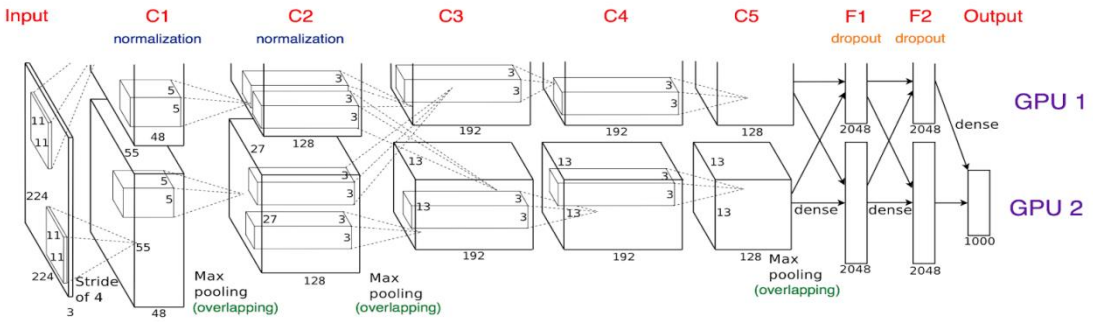


Figura 14 – AlexNet.

La arquitectura mostrada en la figura contiene 8 capas con sus respectivos pesos; las primeras cinco son convolucionales y las restantes están totalmente conectadas – *fully connected* – La función de activación ReLU es ejecutada justo después de cada capa totalmente conectada y cada capa convolucional. El dropout se aplica antes de la primera y la segunda capa *fully connected*. La red tiene 62,3 millones de parámetros y necesita 1,1 millones de unidades de computación en una pasada hacia delante.

VGG

Esta red [25] fue desarrollada por el Visual Geometry Group de la universidad de Oxford. En general existen varias redes bajo este nombre que difieren en el número de capas que poseen. Se trata de una red similar a AlexNet, pero con solo capas de convolución 3x3 y numerosos filtros.

Las más empleadas son:

- VGG-16 está compuesta por un total de 16 capas, 13 de ellas de convolución, 2 de ellas totalmente conectadas y la capa final una softmaxpara clasificar.

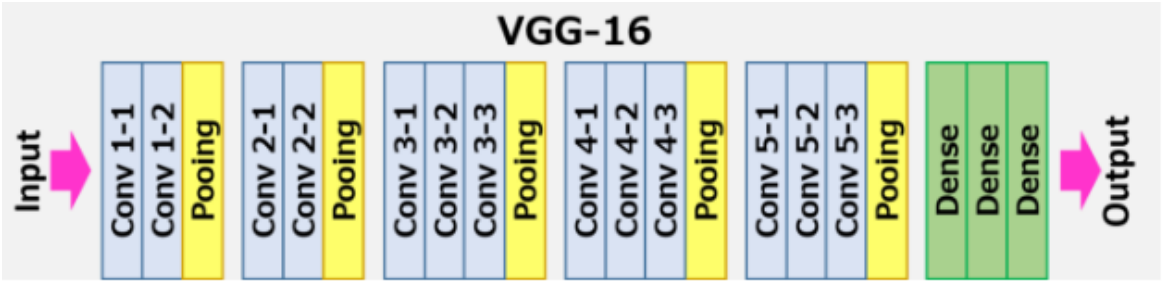


Figura 15 – VGG-16 [27].

- VGG-19 está compuesta por un total de 19 capas, 16 de ellas de convolución, 2 de ellas totalmente conectadas y la capa final una softmax para clasificar.

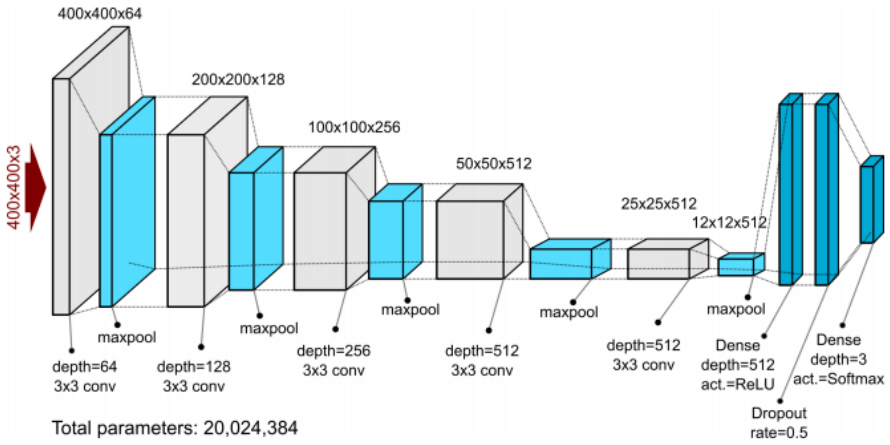


Figura 16 – VGG-19 [28].

GoogleNet

GoogleNet [26] está compuesta por 22 capas y casi 12 veces menos parámetros que por ejemplo AlexNet, lo cual la hace mucho más rápida y precisa. GoogleNet se compone de sus *inception layers* o capas de inicio. La idea principal de éstas, es cubrir un área mayor pero también preservar una buena resolución para una pequeña muestra de información sobre las imágenes.

Se pretende que una serie de filtros con diferentes tamaños manejen mejor las escalas de múltiples objetos, con la ventaja de que todos los filtros en la capa de inicio son más fáciles de aprender. La forma más sencilla de mejorar el rendimiento en el aprendizaje profundo es usar más capas y más datos. Consecuentemente, el problema es que más parámetros involucran una mayor probabilidad de sobreajuste por lo que para evitar dicho sobreajuste en el comienzo se lleva a cabo el uso de todas las técnicas de cuello de botella.

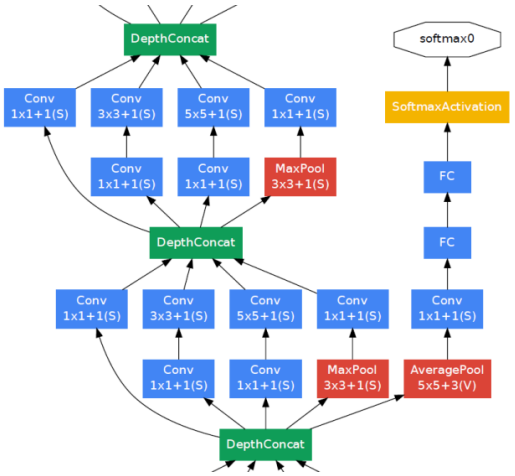


Figura 17 – GoogleNet [26].

ResNet

Las *residual networks* o redes residuales [26] son capaces de aprender funciones más complejas y consecuentemente conducir a un mejor rendimiento. Sin embargo, en ocasiones agregar más capas tuvo eventualmente un efecto negativo en el rendimiento final. Este fenómeno se conoce como el problema de degradación, aludiendo al hecho de que, si bien las mejores técnicas de inicialización de parámetros y la

normalización de lotes permiten que las redes más profundas converjan, en ocasiones convergen a tasas de error más altas que las menos profundas.

La solución que se propone a esta circunstancia es la de introducir bloques residuales en los que las capas intermedias de un bloque aprendan una función residual con referencia a la entrada del bloque. La función puede ser vista como un paso de refinamiento en el que se aprende a cómo ajustar el mapa de características de entrada para obtener mayor calidad. Esto se compara con una red completa donde se espera que cada capa aprenda nuevas características del mapa. En el caso en que no se necesite refinamiento, las capas intermedias pueden aprender a ajustar gradualmente sus pesos hacia cero tal que el bloque residual represente una función de identidad.

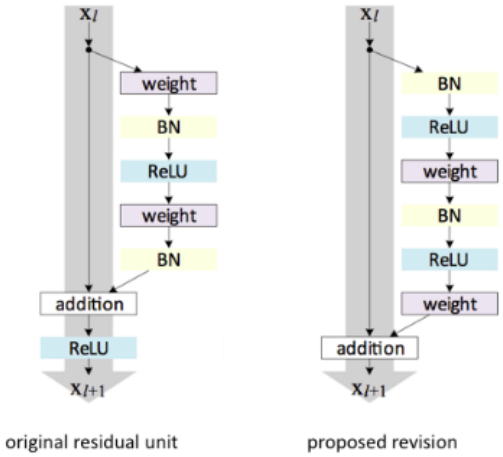


Figura 18 – ResNet [26].

DenseNet

Las redes de convolución densas [26] pueden ser útiles para referenciar mapas futuros desde el inicio de la red. Así, cada capa del mapa de características está concatenada con la entrada de cada capa sucesiva dentro de un bloque denso. Esto permite que las capas posteriores dentro de la red aprovechen directamente las características de las capas anteriores fomentando la reutilización de características dentro de la red. Concatenar los mapas de características aprendidos por diferentes capas incrementa la variación en la entrada de las capas subsecuentes mejorando la eficiencia. Debido a que la red es capaz de usar directamente cualquier mapa de características previo, se puede reducir considerablemente el número de parámetros necesarios.

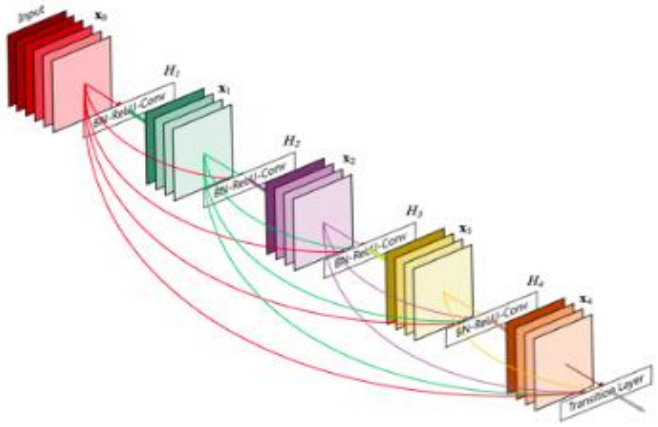


Figura 19 – DenseNet [26].

A modo de resumen se muestra una gráfica que compara la precisión en función del número de operaciones que realizan las distintas arquitecturas:

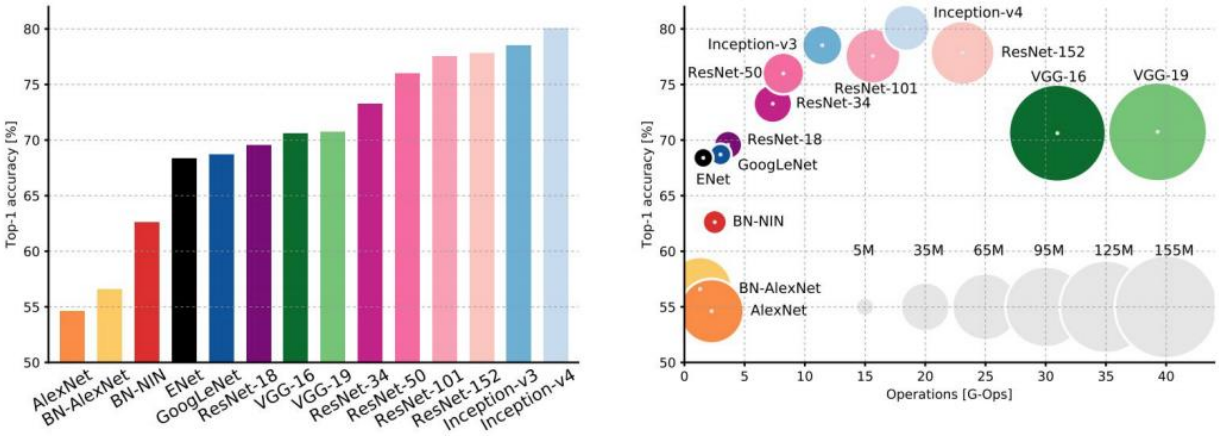


Figura 20 – Gráfico redes neuronales convolucionales profundas [26].

5 TECNOLOGÍAS Y HERRAMIENTAS

"A veces observamos tanto tiempo una puerta que se cierra que vemos demasiado tarde la que está abierta".

-Alexander Graham Bell -

Una vez conocidas las redes neuronales en profundidad, las características principales de las redes neuronales convolucionales y algunas de las más famosas, se describen en este capítulo las distintas tecnologías y herramientas utilizadas para el desarrollo del proyecto.

Dataset

Sin lugar a dudas, dentro del aprendizaje profundo la información que se pretende analizar juega un papel fundamental pues en ella, está basada todo el desarrollo posterior. Para nuestro caso de estudio principal se ha escogido la comunidad de *Kaggle*, enfocada al *deep learning* y que cuenta con gran cantidad de datasets. Además, se han obtenido imágenes de datasets famosos como el de IDC (*Invasive Ductal Carcinoma*) y uno propio para ejemplificar la segmentación de imágenes. Proceso este último, muy común también en el reconocimiento de imágenes.

En concreto, se han seleccionado imágenes correspondientes con el cáncer de mama para entrenar la red neuronal y de huesos y músculos para poder así, verficar su correcto comportamiento. Por otro lado, se han investigado las lesiones de piel y se ha encontrado una fuente de datos correspondiente con imágenes donde dichas lesiones se correspondían con tumores malignos y otras, por el contrario con lesiones benignas.

5.1.1 Generación del dataset de imágenes de hueso y músculo.

Uno de los aspectos más importantes de la relaización de este Trabajo de Fin de Grado ha sido el de la elaboración propia de un dataset para posteriormente poder emplearlo en dos redes neuronales convolucionales distintas.

El proceso ha sido llevado a cabo mediante uno de los lenguajes de programación que se describirán en el siguiente apartado: MATLAB.

En definitiva, el código desarrollado busca que a partir de imágenes originales de 512x512 píxeles correspondientes a hueso y músculo respectivamente, se pudieran segmentar en imágenes más "pequeñas" de 50x50 y corroborar que dichas segmentaciones supusieran hueso o no hueso y músculo o no músculo en la imágenes originales.

Para comprobar si las distitnas secciones de las imágenes principales almacenadas en los archivos se corresponden con hueso o músculo se realizará una comparativa con la *ground truth* o verdad fundamental que se refiere a la información proporcionada por la observación directa en lugar de la información proporcionada por la inferencia.

En el proyecto se analizan hasta diez archivos distintos: DATOS_CASO1.m, DATOS_CASO2.m… DATOS_CASON.m. En ellos, una vez se cargan se dispone de los DATOSX, GT_HUESOX y GT_MUSCULOX, donde la X corresponderá con el número escogido y DATOS con la imagen original. Los archivos GT_HUESO y GT_MUSCULO se corresponderán con la verdad fundamental. La codificación completa se describe en el segundo apartado del Anexo.

Lenguajes de programación

Debemos resaltar que a día de hoy, cualquier lenguaje de programación sirve para la inteligencia artificial, sin embargo se van a presentar los más usados para desarrollar sistemas y algoritmos exitosos, y de entre todos ellos destacaremos el mejor para llevar a cabo el diseño de redes neuronales convolucionales.

Gracias al éxito que la IA ha traído a todos los campos de la ingeniería, se está convirtiendo en algo cada vez más común el desarrollo informático inteligente. Esto se realiza mediante herramientas que permiten crear increíbles sistemas autosuficientes.

Se mostrarán a continuación, los lenguajes de programación más utilizados para crear inteligencia artificial.

5.1.2 LISP

El origen de la programación funcional coincide con el desarrollo del lenguaje LISP [29] por John McCarthy en los primeros años de los 60. Este estilo de programación consiste en la combinación de funciones para producir otras más poderosas.

5.1.3 Prolog

Prolog [30] es un lenguaje de programación simple pero poderoso desarrollado en la Universidad de Aix-Marseille (Marsella, Francia) por los profesores Alain Colmerauer y Philippe Roussel, como una herramienta práctica para programación lógica.

Es un lenguaje de programación seminterpretado. Destaca su facilidad para programar y que el código tiende a ser más corto, sin embargo las librerías disponibles para el manejo de las Redes Neuronales no son extensas.

5.1.4 OPS5

Se trata de un lenguaje de programación basado en reglas [31]. Quizás se trata del menos conocido de los anteriores, no obstante, su importancia en el desarrollo de la inteligencia artificail fue clave dado que se trató del primer lenguaje utilizado con resultado exitoso en un sistema experto.

5.1.5 HASKELL

Al igual que Lisp se trata de un lenguaje de programación funcional [32] por lo que su estructura es similar, es decir, en lugar de realizar acciones en secuencia, evalúan expresiones. Haskell se trata de un lenguaje moderno, de propósito general desarrollado para incorporar el conocimiento colectivo de la comunidad de programación funcional en un lenguaje elegante y potente

5.1.6 MATLAB

Si bien es cierto que MATLAB [33] no es un lenguaje de programación como tal, se trata de un entorno de computación y desarrollo de aplicaciones para desarrollar proyectos en donde se encuentren implicados cálculos matemáticos complejos y la visualización gráfica de los mismos. En nuestro proyecto el uso de MATLAB ha sido principalmente para tratar la segmentación de imágenes. El proceso realizado se explicará con detenimiento en el capítulo siguiente.

5.1.7 Python y R

Por último, se mencionarán conjuntamente los lenguajes de programación más utilizados en el análisis de datos.

R [34] apareció por primera vez en 1996, creado por los profesores de estadística Ross Ihaka y Robert Gentleman de la Universidad de Auckland en Nueva Zelanda.

Python [35] se creó en 1991 por Guido Van Rossum en el Centro para las Matemáticas y la Informática (CWI, Centrum Wiskunde & Informatica) de los Países Bajos.

Si comparamos ambos lenguajes, durante años, R era la elección obvia para aquellos que empezaban en la ciencia de datos. Por otro lado, Python ofrece muchos beneficios, lo que significa que poco a poco se está convirtiendo en el lenguaje más utilizado para el aprendizaje profundo. Algunas de las razones por las que se puede elegir este lenguaje son: la facilidad de uso y construcción de herramientas de análisis, su versatilidad, el crecimiento de la comunidad de usuarios y ser de gran utilidad para el Deep Learning gracias a existir numerosos paquetes

destinados para ello como TensorFlow, Theano o Keras que hacen que sea realmente sencillo crear redes neuronales profundas.

TensorFlow

TensorFlow [36] es un software de computación numérica creado por Google, orientado a problemas de *Deep Learning.*

Posee interesantes integraciones con otras bibliotecas del ecosistema como *Keras*, de la que se hablará más adelante. En el proyecto se explotará la utilidad que presenta para construir y entrenar redes neuronales.

TensorFlow es una librería desarrollada a partir de la combinación de C++ y CUDA El hecho de emplear Python supone un ahorro en cuanto a la declaración del tipo de variables que se obtienen como resultado de las ejecuciones, simplicidad a la hora de utilizar dichas variables, el manejo de vectores y matrices, etc.

La principal estructura de datos ques se maneja en esta librería son los tensors. Con un tensor nos referimos a un conjunto de valores primitivos, por ejemplo números enteros o flotantes, organizados por un array de 1 o N dimensiones, donde el rango del tensor sería el número de dimensiones.

Keras

Keras [37] es una API de redes neuronales a alto nivel, escrita en Python y que puede emplearse haciendo uso de TensorFlow, CNTK o Theano. Fue desarrollada con el enfoque de permitir una experimentación rápida, es decir, pasar de la idea al resultado con el menor retraso posible. Entre sus características destacan:

- Permite una creación de prototipos fácil y rápida (a través de la facilidad de uso, la modularidad y la extensibilidad).
- Admite redes convolucionales y redes recurrentes, así como la combinación de ambas.
- Funciona a la perfección tanto en CPU´s como en GPU´s.

De dicha API [38] se han extraído métodos que posteriormente se mencionan y que en la sección de los Anexos se pueden ver cómo se ha empleado en la elaboración de código. Destacamos:

- *Keras.layers:* donde se describen todos los métodos que poseen las capas keras en común.
- *Keras.layers.convolutional:* aquí se encuentran los métodos utilizados para llevar a cabo las convoluciones en las redes neuronales del proyecto. En nuestra aplicación se han utilizado las convoluciones en dos dimensiones.
- *Keras.layers.core:* de núcleo se obtienen los métodos relacionados con las funciones de activación, el *dropout*, etc.
- *Keras.layers.normalization:* de este paquete se obtienen métodos que permiten la normalización de los *batches* (*Batch Normalization)* y otros métodos que aunque no se utilizan en la elaboración del proyecto son interesante mencionar como: *Bidirectional* o *TimeDistributed.*
- *Keras.models:* en Keras hay dos tipos principales de modelos: *Sequential* (secuencial) y *Model class* (clase modelo). En nuestro proyecto hacemos uso del primero de ellos.
- *Keras.regularizes:* los regularizadores permiten aplicar penalizaciones en los parámetros de capa o en la actividad de capa durante la optimización. Estas penalizaciones se incorporan en la función de pérdida que la red optimiza.

Hardware

Para la realización del proyecto se ha empleado el dispositivo Lenovo-ideapad- 300-15isk cuyas especificaciones se muestran a continuación.

Figura 21 – Portátil Lenovo (I)

Figura 22 – Portátil Lenovo (II)

- Tipo de procesador: Core i7 6500U
- Velocidad del procesador: 2.5 GHz
- Número de procesadores: 2
- Capacidad de la memoria RAM: 12 GB
- Tipo de memoria del ordenador: DDR3 SDRAM
- Capacidad del disco duro: 500 GB
- Tipo de memoria del ordenador: DDR3 SDRAM
- Coprocesador gráfico: AMD
- Descripción de la tarjeta gráfica: AMD Mobility Radeon R5 M330
- Tipo de memoria gráfica: DDR3L-1600 SDRAM
- Plataforma de Hardware: Windows

5.1.8 Procesadores

La CPU [39] y la GPU [40] son los dos principales procesadores existentes en los PC. El primero se encarga de todo tipo de tareas mientras que el segundo está especializado en los gráficos.

Por un lado, la CPU del inglés *Central Processing Unit*, es el procesador de propósito general. Puede llevar a cabo todo tipo de cálculos y está diseñada para el procesamiento en serie de los datos [45].

Por otra parte, se encuentra el procesador gráfico o GPU -*Graphical Processing Unit*- [40] especializado en tareas que requieren un alto grado de paralelismo. Está formada por miles de núcleos en su interior, de tamaño muy reducido y que permiten realizar por tanto, operaciones mucho más reducidas a las de la CPU. Sin embargo, esto hace que una GPU esté optimizada para procesar grandes cantidades de datos y realizar las mismas operaciones específicas constantemente. En realidad, el modelo de computación sobre tarjetas gráficas consiste en usar conjuntamente una CPU y una GPU de manera que formen un modelo de computación heterogéneo.

En los inicios, el problema principal del uso de tarjetas gráficas para el cálculo científico de propósito general: GPGPU – *General Purpose Computing on Graphics Processing Units* - era la necesidad del uso de lenguajes de programación específicos para gráficos.

NVIDIA decidió investigar entonces la forma de modificar la arquitectura de sus GPUs para que fueran programables por aplicaciones científicas además de añadir soporte para lenguajes de alto nivel De este modo, introdujo en sus tarjetas gráficas la arquitectura CUDA -*Compute Unified Device Architecture*-.

CUDA [41] supone una nueva arquitectura para cálculo paralelo de propósito general, con un nuevo modelo de programación paralela, con soporte para lenguajes de alto nivel y constituidas por cientos de núcleos que pueden procesar de manera recurrente miles de hilos de ejecución.

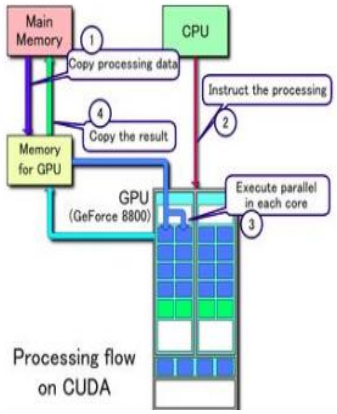


Figura 23 – Arquitectura CUDA

5.1.9 Problemas de instalación

A pesar de las numerosas y destacadas facilidades que la arquitectura CUDA puede proporcionar en el estudio de las redes neuronales, y en general en la Inteligencia Artificial, *Machine Learning* y el *Deep Learning*, el mayor inconveniente que presenta es que se trata de una tecnología exclusiva de NVIDIA y de sus tarjetas, por lo que ante la ausencia de dicha tarjeta en el hardware disponible para la realización del proyecto, no puede hacerse uso de la GPU. Afortunadamente, esto no será algo relevante en el estudio de las Redes Neuronales Convolucionales como tal, sino en la velocidad de procesamiento en dichas redes.

La tarjeta gráfica que ha sido utilizada es la AMD Radeon R5 M330. Es una tarjeta de vídeo dedicada de gama baja para portátiles que se basa en el nuevo derivativo del chip Tonga, y que por tanto, soporta DirectX 12, Vulkan y FreeSync. La velocidad es bastante alta con hasta 1030 MHz, sin embargo, el cuello de botella es la memoria gráfica DDR3 de 64 bits (con velocidad de 1000 MHz, 2000 MHz efectivos).

6 ENTRENAMIENTO DE LAS REDES

"In a forest of a hundred thousand tres, no two leaves are alike. And no two journeys along the same path are alike.".

-Paulo Coelhol -

Para poder utilizar una red neuronal que funcione acorde con sus recomendaciones y de manera óptima, es necesario entrenarla.

Entrenar una red demanda dos aspectos fundamentales: un dataset suficientemente amplio y variado que permita incluir todas las casuísticas de lo que se está analizando y una gran capacidad computacional. En nuestro caso, y al no disponer de una GPU la potencia podría ser suplida por tiempo en caso de ser necesario.

Las dos redes neuronales que se han utilizado en la elaboración del proyecto son ResNet y CancerNet.

La primera de ellas se corresponde con la descrita en el apartado 4.5 mientras que la segunda, aunque no sea una de las oficiales se asemeja al funcionamiento de las VGG.

6.1 ResNet

Como se comentó en el capítulo anterior, la ResNet o *Residual Network* se trata de una red neuronal residual que propone introducir bloques residuales en los que las capas intermedias de un bloque aprendan una función residual con referencia a la entrada del bloque.

Para su desarrollo se han utilizado distintos scripts de Python pues además de implementar la red en sí, se han implementado métodos para:

- Cargar las imágenes de los datasets a la red.
- Generar las imágenes de entrenamiento, validación y pruebas.
- Entrenar la red con dichas imágenes.

Todos estos archivos podrán encontrarse en el Anexo III.

6.1.1 Archivo de configuración

Este archivo permite la configuración de los aspectos esenciales en el entrenamiento de la red neuronal. Se configuran aquí el origen de los datos y el destino de los mismos para poder ser entrenados. Además se establecen también las cantidades de imágenes destinadas al entrenamiento y la validación.

En general se han utilizado el 80% de las imágenes del dataset para el entrenamiento de la red.

6.1.2 Construcción del dataset.

Puesto que el dataset analizado no tiene una división previa para el entrenamiento, la validación y las pruebas, será necesario en primer lugar, llevar a cabo dicha división manualmente. Para crear la división de nuestros datos emplearemos el script *construccion_dataset.py* [Anexo III].

6.1.3 Entrenamiento de la red neuronal convolucional.

A partir del script *modelo_entrenamiento.py* [Anexo III] se consigue entrenar la red y obtener los resultados sobre la misma.

En el código se está haciendo uso de *Keras* para entrenar las imágenes en el modelo de aprendizaje profundo, *sklearn* para imprimir un reportaje de clasificación (*classification_report*), *numpy* para el procesamiento numérico, y *argparse* para el análisis de la línea de comandos.

De todos ellos, el más complejo es *matplotlib*.Debido a que estamos guardando la información en el disco, es necesario utilizar el *backend Agg*. Éste, se utiliza para escribir en un archivo, no para renderizar en una ventana

En general para todos los entrenamientos llevados a cabo durante el proyecto se está trabajando con un número medio de 10 *epochs* y 32 de tamaño de *batch.*

Un tamaño de los lotes de 32 (*batch size*) es adecuado para la mayoría de los sistemas (CPU), como en nuestro caso. Sin embargo, si se hubiera utilizado una GPU podríamos haber incrementado el valor en 64 o incluso más. La tasa de aprendizaje inicial irá decayendo acorde con la función de decaimiento.

Estos valores son necesarios para determinar el número total de pasos por cada época en el proceso de validación/pruebas.

Durante el entrenamiento se aplica un proceso muy útil en el aprendizaje profundo: *data augmentation* (aumento de datos). Se trata de una técnica que consiste en variar la forma, tamaño, posición, etc. de una imagen para que la red neuronal reciba una mayor cantidad de información a pesar de que esta sea la misma pero mostrada de forma diferente. Con ello se consigue que la cantidad de información que la red recibe sea mucho más elevada y que por tanto, la precisión de sus respuestas sea mucho mayor

Los resultados serán obtenidos a partir del método *classification_report* en un formato legible desde la terminal.

6.1.4 ResNet.py

Para la construcción de la red se han empleado multitud de librerías pertenecientes a *Keras* [Anexo III]. El desarrollo de la red sigue el del modelo comentado en el apartado 4 donde se referencian las redes neuronales convolucionales profundas.

Se normalizan las activaciones de la capa anterior en cada lote, es decir, se aplica una transformación que mantiene la activación media cerca de cero y la desviación estándar de activación cerca de uno.

Después se utilizan herramientas para llevar a cabo las convoluciones:

- *Conv2D* crea un núcleo de convolución que está convolucionado con la entrada de la capa para producir un tensor de salidas.
- *AveragePooling2D* para el agrupamiento promedio de los datos espaciales.
- *MaxPooling2D* para la agrupación maxima de los datos espaciales.
- *ZeroPadding2D* se utiliza para la capa de relleno cero en la entrada. Esta capa puede agregar filas y columnas de ceros en la parte superior, inferior, izquierda y derecha de un tensor de imagen.

También se utilizan las capas centrales:

- *Activations* para el desarrollo de las funciones de activación.
- *Dense* para añadir capas ocultas (hidden layers) a la red neuronal.

Y de las capas comunes:

- *Flatten* para convertir los elementos de la matriz de imágenes de entrada en un array plano.
- *Input* para instanciar un tensor en *Keras*.

Hay dos formas de representar los datos de la imagen como una matriz tridimensional. La primera implica tener los canales como la última o tercera dimensión en la matriz *"channels last"*. La segunda implica tener los canales

como la primera dimensión en la matriz, llamados *"channels first"*.

- *channels last:* los datos de imagen se representan en una matriz tridimensional donde el último canal representa los canales de color, por ejemplo [filas] [columnas] [canales].
- *channels first* los datos de imagen se representan en una matriz tridimensional donde el primer canal representa los canales de color, por ejemplo [canales] [filas] [columnas].

Algunas bibliotecas de procesamiento de imágenes y de aprendizaje profundo prefieren los primeros de los canales, y algunas prefieren los últimos. Como tal, es importante estar familiarizado con los dos enfoques para representar imágenes.

CancerNet

6.1.5 Archivo de configuración.

De la misma manera que para la red ResNet antes de llevar a cabo la construcción de la red en si y el entrenamiento de la misma hay que implementar el archivo de configuración, con CancerNet se desarrolla el mismo proceso.

De Nuevo, el origen y destino de los datos dependerá del dataset que se esté empleando para entrenar la red. En el apartado de resultados y conclusiones se observarán los directories utilizados, así como en el manual de usuario adjunto en el anexo de la presente memoria.

También aquí se establece el porcentaje de datos que se quieren utilizar para el entrenamiento –en esta ocasión el 25 % – quedando el resto para la parte de testeo – 75% –. De la sección de entrenamiento de datos se reservan además, algunas imágenes para la validación – en esta ocasión un 10% –.

6.1.6 Construcción del dataset

La construcción del dataset es idéntica a la que se realiza en el apartado 6.1.2 por lo que se pasará directamente a la explicación de cómo se entrena la red.

6.1.7 Entrenamiento de la red neuronal convolucional

En esta ocasión el porcedimiento realizado es muy similar al de ResNet.. Como ocurría con ResNet, el aumento de datos es importante para casi todos los experimentos de aprendizaje profundo con la intención de ayudar con la generalización del modelo.

Esta circunstancia alivia parcialmente la necesidad de recopilar más datos de captación, si bien es cierto, raramente estos dañarán el modelo. En el Anexo III puede observarse el código completo donde además se explica qué aporta cada librería al desarrollo.

6.1.8 CancerNet.py

En la implementación de la CNN se utiliza la librería de aprendizaje profundo *Keras*. La red caracterizada por:

- Utilizar exclusivamente filtros de convolución 3x3, de manera similar a VGGNet.
- Apilar múltiples filtros CONV 3x3 uno encima del otro antes de realizar la agrupación máxima, de nuevo de manera similar a VGGNet.
- Como punto diferenciador a VGGNet utiliza convolución separable en profundidad en lugar de capas convolucionales estándar.

Las convoluciones separables en profundidad se caracterizan porque:

- Son más eficientes.
- Requieren menos memoria.
- Requieren menos computación.
- Su rendimiento es mejor que las estándar en algunas situaciones.

Problemas encontrados

Durante la elaboración del proyecto han sido muchos los problemas encontrados en cuanto a la programación de las redes y su utilización. El principal inconveniente que ya se comentó en el apartado del Hardware ha sido el de la no utilización de una GPU.

De cara a la programación realizada en Python la gran mayoría de librerías utilizadas en la elaboración del proyecto hacían un uso más efectivo de sus servicios si se aplicaban utilizando GPU. Este motivo ha supuesto que para la obtención de cómo hacer uso de redes neuronales convolucionales la investigación haya sido más compleja.

Otra de las dificultades añadidas ha sido la de la utilización del aumento de datos para el entrenamiento de las redes, pues según los parámetros que se utilizaban el aumento crecía de manera exponencial y hacía que la memoria RAM detuviera el proceso por falta de espacio.

Sin embargo, y a pesar de todo esto, considero que al final todo ha quedado en problemas menores en cuanto a comprender el lenguaje Python que era totalmente nuevo para mi. Los foros y consultas a las páginas principales de Python, TensorFlow, Sklearn, etc. han logrado el objetivo de hacer funcionar las redes dando unos resultados bastante próximos a los que en un principio se podían esperar.

7 RESULTADOS Y CONCLUSIONES

"El cambio es siempre el resultado final de todo verdadero aprendizaje.".

-Leo Buscaglia -

Después de realizar la selección e implementación de un modelo obteniendo algunos resultados en forma de probabilidad o clase, el siguiente paso es averiguar la efectividad de este basado en alguna métrica que usa conjuntos de datos de prueba.

En el mundo de la Inteligencia Artificial se utilizan diferentes métricas de rendimiento para evaluar los algoritmos de aprendizaje automático. En consecuencia con el proyecto nos centraremos en los que se utilizan para los problemas de clasificación. Podemos usar métricas de rendimiento de clasificación como *Log-Loss*, Precisión, AUC (Área bajo Curva), etc.

Las métricas que escogidas para evaluar el modelo de aprendizaje automático son muy importantes. La elección de éstas influye en cómo se mide y compara el rendimiento de los algoritmos de aprendizaje automático.

Métricas empleadas

7.1.1 Matriz de confusión.

La matriz de confusión [46] es una de las métricas más intuitivas y sencillas que permite resumir el rendimiento de un algoritmo de clasificación. Antes de indagar en lo que se trata la matriz de confusión y lo que transmite, digamos que estamos resolviendo un problema de clasificación en el que predecimos si una persona tiene cáncer o no, o si una imagen se corresponde con hueso o músculo.

Asignemos una etiqueta a nuestra variable objetivo: 1 cuando una persona tiene cancer y 0 cuando una persona no tiene cáncer. De la misma manera se pueden asignar las etiquetas tal y como se explicó en el capítulo 5 para las imágenes de músculo y hueso. Si dentro de una carpeta de músculo encontramos que la imagen es efectivamente de músculo le corresponderá un 1 mientras que si no lo es será un 0, y al igual con las carpetas de huesos.

Una vez identificado el problema, la matriz de confusión, es una tabla con dos dimensiones ("Actual" y "Predicción") y conjuntos de "clases" en ambas dimensiones. Nuestras clasificaciones actuales son columnas y las pronosticadas son filas.

		Actual	
		Positivos(1)	*Negativos(0)*
Predecido	*Postivos (1)*	TP	FP
	Negativos(0)	FN	TN

Tabla 2 –Matriz de confusión

En realidad, la matriz de confusión en sí misma no es una medida de rendimiento como tal, pero casi todas las métricas de rendimiento están basadas en ella y los números que contiene.

Los términos que tiene asociados son los siguientes:

1. Verdaderos positivos o True Positives (TP): son los casos en los que la clase real del punto de datos fue 1 (verdadero) y la predicción también lo es.
2. Verdaderos negativos o True Negatives (TN): son los casos en los que la clase real del punto de datos fue falsa (0) y el pronóstico también lo es.
3. Falsos positivos o False Positive (FP): son los casos en los que la clase real del punto de datos fue 0 (falso) y el pronóstico es 1 (verdadero). Es falso porque el modelo ha hecho una predicción errónea y positivo porque la clase fue predicha como positiva.
4. Falsos negativos o False Negative (FN): son los casos en los que la clase real del punto de datos fue 1 (verdadero) y la predicción es 0 (falsa). Es falso porque el modelo ha realizado una predicción incorrecta y negativo porque la clase pronosticada fue negativa.

El escenario ideal deseado es que no se den ni falsos positivos ni falsos negativos.

7.1.2 Accuracy.

La precisión en los problemas de clasificación es el número de predicciones correctas realizadas por el modelo sobre todo tipo de predicciones realizadas. De la matriz de confusion:

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{7-1}$$

Se trata de una buena medida cuando las clases de variables de destino en los datos están casi equilibradas, como ocurre en los dataset del proyecto.

7.1.3 Precision.

La precisión es una medida que nos dice qué proporción de pacientes a los que diagnosticamos que tienen cáncer, en realidad tuvieron cáncer. Los positivos pronosticados son las personas que se consideran cancerosas : TP y PF, mientras que las personas que realmente tienen un cáncer son TP. Se puede de nuevo extrapolar a las imágenes de hueso y músculo. Matemáticamente el cálculo viene de:

$$Precision = \frac{TP}{TP + FP} \tag{7-2}$$

7.1.4 Recall.

Es una medida que nos dice qué proporción de pacientes que realmente tuvieron cáncer fue diagnosticado por el algoritmo como si tuviera cancer. Su expresión matemática es:

$$Recall = \frac{TP}{TP + FN} \tag{7-3}$$

El Recall nos da información sobre el desempeño de un clasificador con respecto a falsos negativos (cuántos fallamos), mientras que la precision, descrita en el apartado anterior, nos brinda información sobre su desempeño con respecto a falsos positivos.

En conclusion, si queremos centrarnos más en minimizar los falsos negativos (FN), será interesante que el Recall sea lo más cercano al 100% posible sin que la precisión sea demasiado mala y si queremos centrarnos en

minimizar los falsos positivos (FP), entonces nuestro objetivo debe ser hacer la Precisión lo más cercana al 100% posible.

7.1.5 F1-score.

Representa una puntuación única entre el Recall y la Accuracy anteriores. Una forma de hacerlo es simplemente tomando su media aritmética:

$$F1 = \frac{P + R}{2} \tag{7-4}$$

Pero esto es bastante malo en algunas situaciones. Necesitamos algo más equilibrado que la media aritmética y esa es la media armónica:

$$F1_{armonica} = \frac{2xy}{x} + y \tag{7-5}$$

La media armónica es una especie de promedio cuando x e y son iguales. Pero cuando son diferentes, entonces está más cerca del número más pequeño en comparación con el número más grande.

Por lo tanto, si un número es realmente pequeño entre precisión y recuperación el tipo de puntuación F1 eleva una bandera que está más cerca del número más pequeño que el más grande, lo que le da al modelo una puntuación apropiada en lugar de solo una media aritmética.

Resultados de los distintos datasets

Durante la elaboración del proyecto se ha estado trabajando fundamentalmentecon cuatro conjuntos de datos diferentes.

El primero de ellos consiste en un conjunto de imágenes asociados a cánceres de mama, el segundo con imágenes de otro tipo de cancer: el producido por las lesiones en la piel y conocido comúnmente como melanoma y los dos últimos con imágenes correspondientes a hueso o músculo en función del segmento analizado tal y como se ha ido comentando a lo largo de la memoria del proyecto.

Aunque a priori la idea principal era indagar sobremanera en el análisis de las lesiones de piel, el hecho de no haber encontrado la suficiente cantidad de información para trabajar con dichas lesiones supuso un giro en el análisis final. Es por esto que para la comprobación del correcto funcionamiento de las redes, con los set de datos que más se ha investigado ha sido con los que contenían imágenes de hueso o músculo. Con ellos, además se ha llevado a la práctica el concepto de *cross-validation* que sera explicado en las siguientes líneas.

A continuación para ver la potencia y eficacia de las redes neuronales evaluando los distintos datasets mencionados, se van a rellenar una serie de tablas con los valores devueltos por el entrenamiento de las redes.

Destacar que en las tablas siguientes, además de las estadísticas obtenidas se muestran el número de imágenes que corroboran que se trata de verdadero (1) o falso (0). Esto se obtiene en la columna *Support*.

En los siguientes apartados referentes a los datasets de imágenes de mama y melanoma se ha empleado la red CancerNet y ResNet, respectivamente.

7.1.6 Dataset imágenes cáncer de mama

Las imágenes siguientes provienen de la base de datos *Invasive Ductal Carcinoma* (IDC) y se trata del subtipo de cancer de mama más común. El archivo de datos original consistía en 162 imágenes de biopsias escaneadas a 40x. A partir de ahí se extrajeron 277.524 segmentos de un tamaño de 50x50. Concretamente se tienen 198.738 IDC negativos y 78.786 IDC positivos [47].

Durante el análisis de la red con estas imágenes, los distitntos sets empleados: entrenamiento, validación y pruebas, estaban compuestos de 199.818 imágenes de entrenamiento (el 80% del conjunto total), 22.201 para la validación y 55.505 para las pruebas.

A continuación se muestran algunas de las imágenes analizadas:

- Imágenes correspondientes con ausencia de cáncer (0):

Figura 24 – Imágenes mama (I).

- Imágenes cancerígeneas (1):

Figura 25 – Imágenes mama (II).

La tabla de resultados obtenida es como sigue:

	Precisión	Recall	F1-score	Support
1	0.89	0.90	0.90	39694
0	0.74	0.73	0.74	15811

Tabla 3 – Resultados del dataset con imágenes de mama.

Los resultados han sido obtenidos a partir de cargar en la red CancerNet el set de datos mencionado: IDC. Se han generado directorios de entrenamiento, validación y pruebas, y finalmente se ha entrenado y evaluado la red [Anexo IV]. El resultado de dicha evaluación ha sido registrado en una tabla para poder apreciar exhaustivamente la precisión de la red.

Este resultado final nos demuestra que a pesar de no haber podido emplear un número excesivo de imágenes para entrenar la red, ni haber ejecutado el análisis durante un mayor número de épocas debido a la falta de potencia del hardware utilizado los resultados obtenidos no son del todo negativos. Sin embargo, hay que tener en cuenta que se tartan de decisiones sobre enfermedades humanas, y que la precisión debe ser máxima

7.1.7 Dataset imágenes melanoma

En esta ocasión, las imágenes mostradas anteriormente se corresponden con imágenes reales obtenidas de "*The*

International Skin Imaging Collaboration" (ISIC) [48]. En el archivo pueden encontrarse 19.373 imágenes benignas y 2.286 malignas.

Para el entrenamiento y prueba de la red, los dataset empleados de entrenamiento, validación y pruebas estaban compuestos de 2.376, 265 y 662 imágenes, respectivamente.

En el caso del melanoma también resulta interesante observar las imágenes malignas y benignas:

- Imágenes benignas (0):

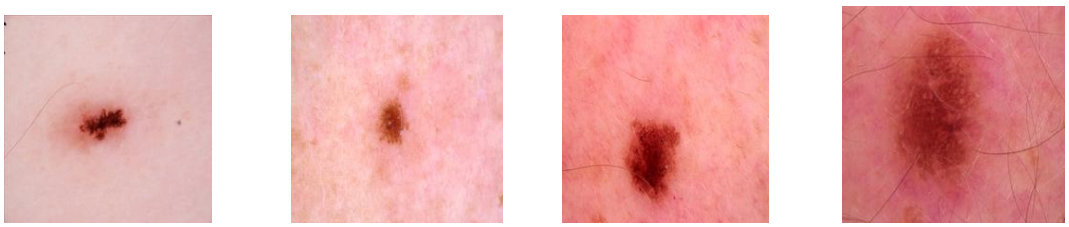


Figura 26 – Imágenes piel (I)

- Imágenes malignas (1):

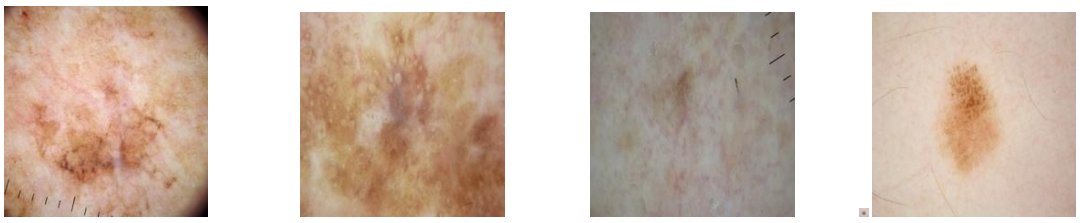


Figura 27 – Imágenes piel (II)

Y los resultados del entrenamiento de la red:

	Precisión	Recall	F1-score	Support
1	0.84	0.83	0.84	346
0	0.82	0.83	0.82	314

Tabla 4 – Resultados del dataset con imágenes de melanoma

Apreciamos ahora que la precisión es similar a la obtenida en el análisis de los cáncer de mama. De nuevo, matemáticamente el resultado no es malo pero a efectos de decisiones médicas es muy mejorable.

7.1.8 Dataset imágenes caso hueso o músculo.

En esta clasificación, como se comentó en el apartado 5 del proyecto, los resultados difieren a los de los dos casos previos pues en esta ocasión, la red va a evaluar si dentro de cada una de las divisiones que se han llevado a cabo, esto es: hueso o músculo, la imagen se corresponde con hueso o no hueso y músculo o no músculo. Al contrario que en los dos análisis anteriores en esta ocasión si se ha trabajado con las dos redes descritas en el proyecto.

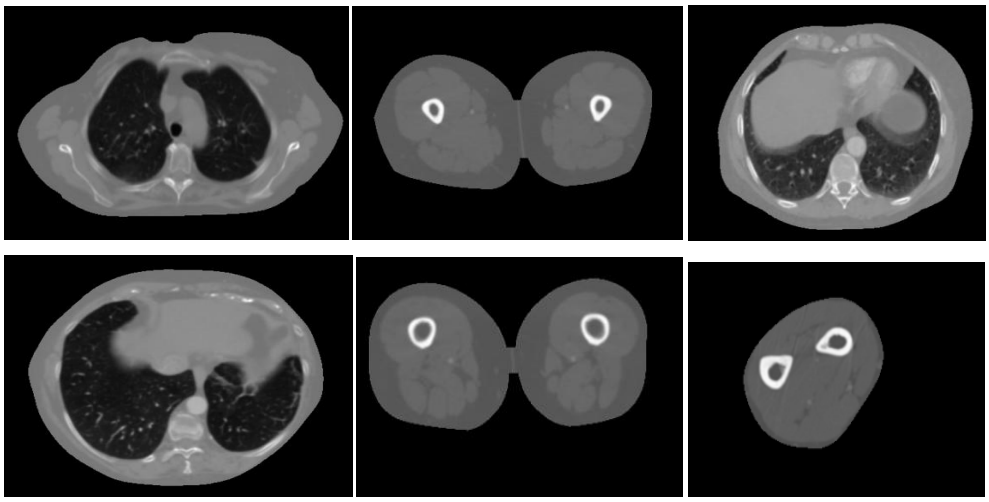


Figura 28. Imágenes de músuculo y hueso

La figura anterior muestra las imágenes originales que haciendo uso de los scripts de MATLAB mencionados en el apartado 5.1.1 de la memoria y posteriormente explicados en el Anexo II, se han utilizado para el análisis de la precisión de las redes CancerNet y ResNet.

Para comprobar la precisión total de nuestros experimentos ha sido muy útil aplicar el método de *Cross-validation* [46] o validación cruzada.

La validación cruzada consiste en una técnica para evaluar los resultados de cierto análisis estadístico y garantizar la independencia de la partición entre datos de entrenamiento y prueba. El método consiste en la obtención de distinas medidas de evaluación sobre diferentes particiones y calcular una media aritmética de los resultados.

En el desarrollo del proyecto, se ha realizado la *k-fold-cross- validation* o validación cruzada de k iteraciones. Concretamente se han llevado a cabo 5. En cada una de ellas se ha entrenado el modelo con el 80% de los casos y se ha evaluado con el 20% restante. En los distitnos casos analizados había una media de 15.000 imágenes de las cuales un porcentaje fue destinado al entrenamiento de la red y otro distinto a la validación y las pruebas [Anexos]

7.1.8.1 Iteraciones en la validación cruzada. Imágenes de Hueso

Los resultados de todas las iteraciones se han ido recogiendo en tablas para finalmente llevar a cabo el cálculo de la media aritmética comentada anteriormente y obtener así la precisión de las resdes entrenadas.

A continuación podemos observar algunas de las imágenes que a través de los archivos de MATLAB se consideraron como hueso, y que posteriormente se analizaron con las redes estudiadas:

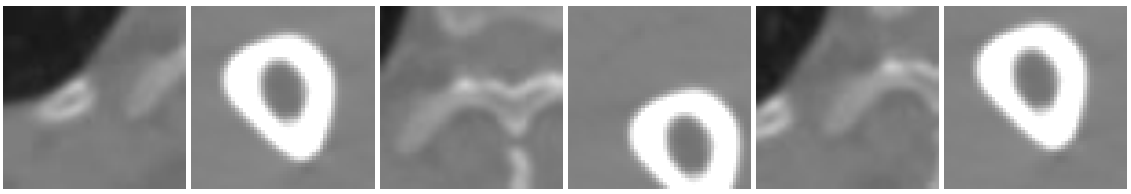


Figura 29. Imágenes hueso.

	Precisión	Recall	F1-score	Support
1	0.79	0.97	0.87	194
0	0.95	0.69	0.80	162

Tabla 5. Iteración 1. ResNet

	Precisión	Recall	F1-score	Support
1	0.81	0.98	0.90	200
0	0.97	0.80	0.89	226

Tabla 6. Iteración 2. ResNet

	Precisión	Recall	F1-score	Support
1	0.77	0.92	0.81	155
0	0.92	0.79	0.83	198

Tabla 7. Iteración 3. ResNet

	Precisión	Recall	F1-score	Support
1	0.79	0.91	0.83	289
0	0.92	0.88	0.85	370

Tabla 8. Iteración 4. ResNet

	Precisión	Recall	F1-score	Support
1	0.77	0.92	0.81	155
0	0.92	0.79	0.83	198

Tabla 9. Iteración 5. ResNet

	Precisión	Recall	F1-score	Support
1	0.786	0.94	0.844	199
0	0.936	0.79	0.84	231

Tabla 10. Promedio con ResNet. Hueso

Concluimos que haciendo uso de la red ResNet las imágenes de la base de datos que no se correspondan con hueso serán obtenidas con aproximadamente un 94% de precisión, mientras que las que se correspondan con hueso lo harán con un porcentaje del 79%. De la misma forma que se ha mostrado anteriormente se han llevado a cabo las predicciones para la red CancerNet. La tabla promedio final resulta:

	Precisión	Recall	F1-score	Support
1	0.914	0.934	0.86	252
0	0.838	0.926	0.858	259

Tabla 11. Promedio con CancerNet. Hueso

Con el uso de CancerNet obtenemos unos porcentajes más compensados en cuanto a si la imagen efectivamente se trata de hueso o no, con un aproximadamente 92% y 84% respectivamente.

7.1.8.2 Iteraciones en la validación cruzada. Imágenes de Músculo.

Los cálculos realizados son idnénticos a los del caso anterior por lo que resulta suficiente con indicar la tabla de resultados de los valores promedio de ambas redes.

También se muestran como ejemplo las imágenes de músculo analizadas:

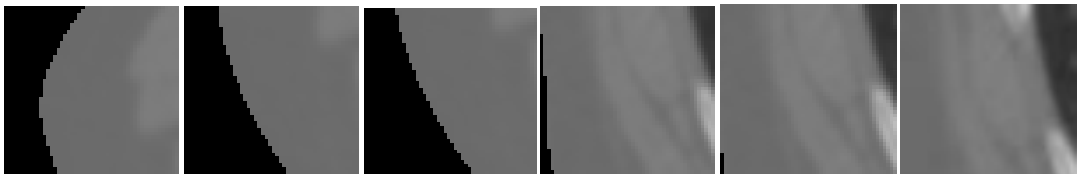


Figura 30. Imágenes músculo.

	Precisión	Recall	F1-score	Support
1	0.898	0.914	0.864	255
0	0.9	0.884	0.854	279

Tabla 12. Promedio con ResNet. Músculo

	Precisión	Recall	F1-score	Support
1	0.872	0.94	0.9	255
0	0.948	0.922	0.934	279

Tabla 13. Promedio con CancerNet. Músculo.

Analizando los resultados promedio de las dos Redes, podemos concluir que para las imágenes de músculo, la precision de nuevo es bastante elevada y además el porcentaje de precisión no difiere en demasía con respecto a las imágenes etiquetadas con 0 (no es músculo) y 1 (músculo).

Conclusiones

Tras haber entrenado y probado las redes en profundidad y en consecuencia de las estadísticas mencionadas y calculadas podemos concluir que:

- Las redes son capaces de calcular correctamente casi la totalidad de las imágenes de hueso o músculo presentes en las bases de datos, pues se ha alcanzado entorno al 92% de precision en todas las situaciones.
- En cuanto a las bases de datos de imágenes sobre tumores pese a que la precisión obtenida ha sido algo menor, podemos afirmar el correcto funcionamiento de las redes y la potencia de estas.
- Un efecto común en las pruebas y causante de imperfecciones ha sido el hecho de que en algunos casos el objetivo a detectar, por estar en los límites de la imagen no pudo analizarse.
- El total de los valores obtenidos corrobora lo mencionado, y es que la precisión de nuestros test ha sido muy elevada en todos los casos.

Por otro lado, cabe destacar que los resultados presentados corresponden con una prueba en particular, y que para cada análisis de las bases de datos se han hecho pequeñas modificaciones en cuanto a la cantidad de imágenes utilizadas para el entrenamiento y para las pruebas, manteniéndose siempre la alta calidad de las estadísticas obtenidas.

Además, a pesar de no contar con las herramientas óptimas para el Deep Learning, se ha demostrado que pueden estudiarse de manera eficiente redes neuronales convolucionales obteniéndose resultados que a priori no se esperaban. El único inconveniente relacionado con esto ha sido el tiempo, pues para la prueba de las redes el modificar alguno de los parámetros suponía que el tiempo de entrenamiento ascendiese incluso a más de un día completo.

El modelado de la red tampoco ha sido algo sencillo pues dar con los valores y parámetros correctos se basa en predicciones, y la imaginación junto a la investigación han jugado un papel fundamental.

Como puede apreciarse este tipo de proyectos no tiene una aplicación práctica directa sino que se trata de una prueba de concepto elaborada. A partir de esta, se pueden asentar las bases para la construcción posterior de modelos robustos de Deep Learning que permitan por ejemplo detectar con una precision óptima si una lesión en la piel es perjudicial o benigna.

En cuanto a las posibles líneas de investigación futuras, considero que la Inteligencia Artificial, el *Machine Learning* y el *Deep Learning* son la base para todos los proyectos venideros independientemente del campo en el que se esté trabajando. Cabe destacar la importancia que está tomando especialmente en el mundo de la medicina pues día a día se evoluciona para que lesiones en la piel, tumores malignos, etc se detecten con seguridad y pueda lucharse contra ellos de una manera mucho más efectiva.

Para finalizar, y desde un punto de vista más personal gracias a la elección de este proyecto que en nada tiene que ver con mis especialización dentro del sector de las Telecomunicaciones (la Telemática) he conseguido aprender nuevas tecnologías interesantes y cuya utilidad para con los demás puede ser muy importante.

REFERENCIAS

[1] Bruno López Takeyas «Introducción a la Inteligencia Artificial. » Instituto Tecnológico de Nuevo Laredo. Web del autor: http://www.itnuevolaredo.edu.mx/takeyas/.

[2] «Qué es Machine Learning» [En línea] Available: https://www.definicionabc.com/tecnologia/machine-learning.php. Último acceso: 03/2019.

[3] «Towards Data Science» [En línea] Available: https://towardsdatascience.com/deep-learning-101-for-dummies-like-me-a53e3caf31b1. Último acceso: 03/2019

[4] Simon Haykin «Neural Networks - A Comprehensive Foundation». Editorial: Pearson. Año de la novena publicación: 2005.

[5] Hilera, G. (1994). «Redes neuronales artificiales. Fundamentos, modelos y aplicaciones». Serie Paradigma, RaMa. Autores: Raquel Flórez López, José Miguel Fernández Fernández.

[6] «Conceptos Básicos sobre Redes Neuronales» [En línea] Available: http://grupo.us.es/gtocoma/pid/pid10/RedesNeuronales.htm#arquitecturaredes. Último acceso: 03/2019.

[7] Pedro Larranaga, Iñaki Inza, Abdelmalik Moujahid ,Departamento de Ciencias de la Computación e Inteligencia Artificial Universidad del Paíss Vasco–Euskal Herriko Unibertsitatea «Tema 8. Redes Neuronales» [En línea] Available: http://www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/t8neuronales.pdf.

[8] «Redes Neuronales Artificiales» [En línea] Available: http://www.oocities.org/motorcity/shop/3186/redesneu.htm. Último acceso: 03/2019.

[9] « Redes Neuronales Artificiales. Un enfoque Práctico. » Editorial: Prentice Hall. Autores: Isasi Viñuela, P., Galván León, I.M. (2004).

[10] «Cleverpy» [En línea] Available: https://cleverpy.com/red-convolucional-pytorch/. Último acceso: 05/2019.

[11] «Edureka!» [En línea] Available: https://www.edureka.co/blog/convolutional-neural-network/. Último acceso: 04/2019

[12] Diego Calvo «Red Neuronal Convolucional CNN» [En línea] Available: http://www.diegocalvo.es/red-neuronal-convolucional/. Último acceso: 04/2019

[13] «Numerentur.org» [En línea] Available: http://numerentur.org/funcion-de-activacion-softmax/. Último acceso: 04/2019

[14] «i-Ciencias»» [En línea] Available: https://www.i-ciencias.com/pregunta/99402/como-funciona-el-relu-funcion-de-trabajo-para-la-z--0. Último acceso: 05/2019

[15] «Medium» [En línea] Available: https://embarc.org/embarc_mli/doc/build/html/MLI_kernels/pooling_max.html?fbclid=IwAR3eoZA_08pk_3eHZOR0Vv5ZTRiQUgu1NmwxVSbKOAXLKRkTTA32WaA94Gs. Último acceso: 04/2019

[16] «Medium» [En línea] Available: https://medium.com/@rajatgupta310198/getting-started-with-neural-network-for-regression-and-tensorflow-58ad3bd75223. Último acceso: 07/2019.

[17] Joanna Jaworek-Korjakowska, Pawel Kleczek, Marek Gorgon «Melanoma Thickness Prediction Based on Convolutional Neural Network with VGG-19 Model Transfer Learning».[En línea]: http://openaccess.thecvf.com/content_CVPRW_2019/papers/ISIC/Jaworek-Korjakowska_Melanoma_Thickness_Prediction_Based_on_Convolutional_Neural_Network_With_VGG-19_CVPRW_2019_paper.pdf. Último acceso: 07/2019.

[18] Mohamed Elleuch, Rania Maalej and Monji Kherallah «A New Design Based-SVM of the CNN Classifier Architecture with Dropout for Offline Arabic Handwritten Recgnition». International Conference on Computational Science 2016, ICCS 2016, 6-8 June 2016, San Diego, California, USA.

[19] «Hackernoon» [En línea] Available: https://hackernoon.com/what-is-one-hot-encoding-why-and-when-do-you-have-to-use-it-e3c6186d008f. Último acceso: 04/2019.

[20] «Skymind» [En línea] Available: https://skymind.ai/wiki/backpropagation. Último acceso: 04/2019.

[21] «Towards data science» Availablehttps://towardsdatascience.com/dropout-on-convolutional-layers-is-weird-5c6ab14f19b2. Último acceso: 04/2019.

[22] «Medium» [En línea] Available: https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76. Último acceso: 04/2019.

[23] «AprendeMachineLearning» [En línea] Availablehttps://www.aprendemachinelearning.com/que-es-overfitting-y-underfitting-y-como-solucionarlo/. Último acceso: 04/2019.

[24] «Medium» [En línea] Available: https://medium.com/@pechyonkin/key-deep-learning-architectures-lenet-5-6fc3c59e6f4. Último acceso: 04/2019.

[25] Mauricio Delbracio, José Lezama, Guillermo Carbajal «Aprendizaje profundo para visión artificial»[En línea] Available: https://iie.fing.edu.uy/~mdelbra/DL2017/slides/c11.pdf. Último acceso: 03/2019.

[26] «Zenva» [En línea] Available: https://pythonmachinelearning.pro/understanding-advanced-convolutional-neural-networks/. Último acceso: 03/2019.

[27] «VGG 16- Convolutional Network for Classification and Detection» [En línea] Available: https://neurohive.io/en/popular-networks/vgg16/?fbclid=IwAR1_8qTXKK2Jo4_WpXdoGnbK46g2NpiUorJeFUf5BxezUVt3AhfF4F-FTuY/. Último acceso: 03/2019.

[28] Joanna Jaworek-Korjakowska, Pawel Kleczek, Marek Gorgon «Melanoma Thickness Prediction Based on Convolutional Neural Network with VGG-19 Model Transfer Learning». Último acceso: 07/2019.

[29] «Medium» [En línea] Available: https://medium.com/ai-society/the-lisp-approach-to-ai-part-1-a48c7385a913. Último acceso: 03/2019.

[30] Jaime Andrés Vargas, Jonathan Alberto Ortiz «Prolog» [En línea] Available: http://ferestrepoca.github.io/paradigmas-de-programacion/proglogica/tutoriales/prolog-gh-pages/Prolog.pdf. Último acceso: 03/2019.

[31] «OSP5» [En línea] Available: https://en.wikipedia.org/wiki/OPS5. Último acceso: 05/2019.

[32] Fernando Sancho Caparrini «Haskell: el Lenguaje Funcional» [En línea] Available: http://www.cs.us.es/~fsancho/?e=110. Último acceso: 04/2019.

[33] «MATLAB» Available: https://www.mathworks.com/. Último acceso: 05/2019.

[34] «Python» [En línea] Available: https://docs.python.org. Último acceso: 05/2019.

[35] «R» [En línea] Available: https://en.wikipedia.org/wiki/R_(programming_language). Último acceso: 05/2019.

[36] «Tutorial de TensorFlow» [En línea] Available: https://www.tutorialspoint.com/tensorflow/tensorflow_tutorial.pdf. Último acceso: 05/2019.

[37] «Keras» Available: https://riptutorial.com/Download/keras-es.pdf. Último acceso: 04/2019.

[38] «Keras» [En línea] Available: https://keras.io. Último acceso: 07/2019.

[39] «CPU» [En línea] Available: https://es.wikipedia.org/wiki/Unidad_central_de_procesamiento. Último acceso: 03/2019.

[40] «GPU» [En línea] Available: https://es.wikipedia.org/wiki/Unidad_de_procesamiento_gr%C3%A1fico. Último acceso: 03/2019.

[41] «CUDA» [En línea] Availablehttps://es.wikipedia.org/wiki/CUDA. Último acceso: 03/2019.

[42] «Artificial Intelligence with Python» Autor: Prateek Joshi. Fuente: https://smtebooks.com/book/4478/artificial-intelligence-python-pdf

[43] Paul Mooney «Breast Histopathology Images» [En línea] Available: https://www.kaggle.com/paultimothymooney/breast-histopathology-images. Último acceso: 06/2019.

[44] «Quora» [En línea] Available: https://www.quora.com/What-is-an-intuitive-explanation-of-cross-validation?fbclid=IwAR05kalRI4WVUy7DWCRUTShEzlyXIhVP9eSJdVE9YENdaCN-Pe5dMLdoYEM. Último acceso: 06/2019.

[45] «GPU vs CPU, diferencias y similitudes» [En línea] Available: https://blog.hostdime.com.co/gpu-cpu-diferencias-similitudes/?fbclid=IwAR1YNjKSo9ce8cSeixItiVnYY7qoPiji1Jj-Ig-sSHNOqYiasg461yXB058. Último acceso 04/2019.

[46] Aurélien Géron «Hands-On Machine Learning with Scikit-Learn & TensorFlow». Editorial: O'REILLY. Publicado: 03/2017.

[47] «Kaggle» [En línea] Available: https://www.kaggle.com/paultimothymooney/predict-idc-in-breast-cancer-histology-images. Último acceso: 07/2019.

[48] «ISIC Archive » [En línea] Available:https://www.isic-archive.com/#!/topWithHeader/wideContentTop/main. Último acceso 07/2019.

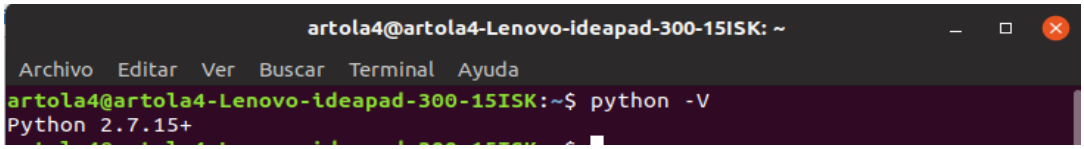
ANEXO

Tutorial de instalación de los componentes del proyecto

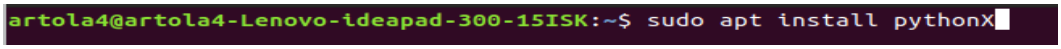
Para la realización del proyecto se ha llevado a cabo una partición de mi ordenador personal para poder utilizar Ubuntu. En concreto, se ha trabajado con la versión de Ubuntu 18.04.

Python

Tal y como se ha comentado en el desarrollo de la memoria, el lenguaje de programación empleado para el diseño, y pruebas de las redes neuronales convolucionales ha sido Python. La versión empleada es la 2.7.15+:



En realidad se puede trabajar con cualquier versión de Python siempre y cuando la versión no esté del todoanticuada. Para su instalación:



Siendo X la versión que se desee. De nuevo, en mi caso la 2.7.15+.

Entornos virtuales

Un entorno virtual de Python es un ambiente creado con el objetivo de aislar recursos como librerías y entorno de ejecución, del sistema principal o de otros entornos virtuales. Lo anterior significa que en el mismo sistema, maquina o computadora, es posible tener instaladas multiples versiones de una misma librería sin crear ningún tipo de conflicto.

Cuando se está desarrollando software con Python, es común utilizar diferentes versiones de un mismo paquete. Para poder utilizar este concepto es necesario instalar una utilidad que permita gestionar la creación y utilización de dichos entornos virtuales llamada **virtualenv**.

Para su instalación, primero es necesario instalar el paquete instalador de Python: **pip** y a seguidamente el entorno virtual mediante: pip install virtualenv.

Directorios

Una vez se tienen instalados Python y el entorno virtual, antes de seguir con la instalación del backend que empleamos (TensorFlow) o las distintas librerías que se utilizan es necesario crear la jerarquía de directorios que se establecerá para el desarrollo del proyecto.

Se puede trabajar en cualquier ubicación de nuestro ordenador personal. En mi caso he escogido la ruta: /home/artola4/Escritorio/TFG

Dentro de ésta se encontrarán los distintos directorios principales que se han utilizado para el entrenamiento de las redes neuronales CancerNet y ResNet. Con ambas redes se lleva a cabo el siguiente procedimiento:

1) Creación del directorio cancerNet_ITX_Hueso/Músculo. Compuesto por :

- busquedaImagenes_py, donde se encuentran los scripts:
  - cancernet.py
  - config.py
  - _init__.py
- modelo_entrenamiento.py
- clasificación
  - entrenamiento
    - 0
    - 1
  - pruebas
    - 0
    - 1
  - Validación
    - 0
    - 1

Para la creación del directorio basta con situarse en la carpeta donde se quieran realizar todas las pruebas y extraer el fichero cancerNet_ITX_Hueso/Musculo.zip

Con todo ello creado hay que situarse en C:/carpetaDeTrabajo/cancerNet_ITX_Hueso/Musculo y crear el entorno virtual que nos permitirá posteriormente trabajar con las distintas librerías de Python.

Desde allí se ejecutarán los siguientes comandos desde la terminal:

- **virtualenv entornoVirtual**
- **source entornoVirtual/bin/activate**
- **pip install numpy opencv-contrib-python**
- **pip install pillow**
- **pip install tensorflow keras**
- **pip install imutils**
- **pip install scikit-learn matplotlib**

Una vez realizado este proceso se construye el dataset mediante el comando:**python construccion_dataset.py**

Dentro de la carpeta clasificación se encuentra el set de imágenes obtenido de la segmentación mediante el código MATLAB. Este set de imágenesse divide a su vez en tres carpetas disitntas que contienen dos carpetas 0 y 1 correspondientes con secciones de no hueso y sí hueso, respectivamente o no músculo y sí musculo, según corresponda. Al ejecutar el comando mencionado:

```
(entornoVirtual) artola4@artola4-Lenovo-ideapad-300-15ISK:~/Escritorio/TFG/Entre
gaFinal/cancerNet$ python construccion_dataset.py
[INFO] construyendo la división 'entrenamiento'
[INFO] construyendo la división de 'clasificacion/entrenamiento'
[INFO] 'creando el directorio clasificacion/entrenamiento/0'
[INFO] 'creando el directorio clasificacion/entrenamiento/1'
[INFO] construyendo la división 'validacion'
[INFO] construyendo la división de 'clasificacion/validacion'
[INFO] 'creando el directorio clasificacion/validacion/0'
[INFO] 'creando el directorio clasificacion/validacion/1'
[INFO] construyendo la división 'pruebas'
[INFO] construyendo la división de 'clasificacion/pruebas'
[INFO] 'creando el directorio clasificacion/pruebas/0'
[INFO] 'creando el directorio clasificacion/pruebas/1'
```

Tras la construcción del set de datos se puede entonces ejecutar el entrenamiento de la red: **python modelo_entrenamiento.py**

```
(entornoVirtual) artola4@artola4-Lenovo-ideapad-300-15ISK:~/Escritorio/TFG/Entre
gaFinal/cancerNet$ python modelo_entrenamiento.py
Using TensorFlow backend.
Found 17336 images belonging to 2 classes.
Found 1926 images belonging to 2 classes.
Found 4816 images belonging to 2 classes.
WARNING:tensorflow:From /home/artola4/Escritorio/TFG/EntregaFinal/cancerNet/ento
rnoVirtual/local/lib/python2.7/site-packages/tensorflow/python/framework/op_def_
library.py:263: colocate_with (from tensorflow.python.framework.ops) is deprecat
ed and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
```

Apareciendo en la terminal el desarrollo del entrenamiento:

```
424/541 [======================>.......] - ETA: 1:04 - loss: 0.0996 - acc: 0.973
425/541 [======================>.......] - ETA: 1:04 - loss: 0.0994 - acc: 0.973
426/541 [======================>.......] - ETA: 1:03 - loss: 0.0992 - acc: 0.973
427/541 [======================>.......] - ETA: 1:02 - loss: 0.0991 - acc: 0.973
428/541 [======================>.......] - ETA: 1:02 - loss: 0.0989 - acc: 0.973
429/541 [======================>.......] - ETA: 1:01 - loss: 0.0992 - acc: 0.973
430/541 [======================>.......] - ETA: 1:01 - loss: 0.0990 - acc: 0.973
431/541 [======================>.......] - ETA: 1:00 - loss: 0.0988 - acc: 0.973
432/541 [======================>.......] - ETA: 1:00 - loss: 0.0986 - acc: 0.973
433/541 [======================>.......] - ETA: 59s - loss: 0.0984 - acc: 0.9734
541/541 [==============================] - 301s 556ms/step - loss: 0.0849 - acc:
 0.9767 - val_loss: 0.0180 - val_acc: 0.9943
Epoch 2/10
  1/541 [..............................] - ETA: 4:33 - loss: 0.0061 - acc: 1.000
  2/541 [..............................] - ETA: 4:46 - loss: 0.0115 - acc: 1.000
  3/541 [..............................] - ETA: 4:47 - loss: 0.0157 - acc: 1.000
  4/541 [..............................] - ETA: 4:50 - loss: 0.0225 - acc: 0.992
  5/541 [..............................] - ETA: 4:47 - loss: 0.0200 - acc: 0.993
  6/541 [..............................] - ETA: 4:56 - loss: 0.0172 - acc: 0.994
  7/541 [..............................] - ETA: 5:07 - loss: 0.0401 - acc: 0.991
  8/541 [..............................] - ETA: 5:00 - loss: 0.0396 - acc: 0.988
  9/541 [..............................] - ETA: 4:55 - loss: 0.0358 - acc: 0.989
 10/541 [..............................] - ETA: 4:50 - loss: 0.0333 - acc: 0.990
```

Una vez finalizado se obtienen los resultados finales y una gráfica que representa la evolución a lo largo de las épocas de la pérdida y precisión de la red.

```
artola4@artola4-Lenovo-ideapad-300-15ISK: ~/Escritorio/TFG/EntregaFinal/cancerNet
 Archivo  Editar  Ver  Buscar  Terminal  Ayuda
488/541 [==========================>...] - ETA: 1:04 - loss: 0.0118 - acc: 0.997
489/541 [==========================>...] - ETA: 1:02 - loss: 0.0118 - acc: 0.997
490/541 [==========================>...] - ETA: 1:01 - loss: 0.0118 - acc: 0.997
491/541 [==========================>...] - ETA: 1:00 - loss: 0.0117 - acc: 0.997
492/541 [==========================>...] - ETA: 59s - loss: 0.0117 - acc: 0.9973
541/541 [==============================] - 671s 1s/step - loss: 0.0114 - acc: 0.
9973 - val_loss: 0.0109 - val_acc: 0.9979
[INFO] evaluando la red...
              precision    recall  f1-score   support

           1       1.00      1.00      1.00      4721
           0       1.00      0.91      0.95        95

   micro avg       1.00      1.00      1.00      4816
   macro avg       1.00      0.95      0.97      4816
weighted avg       1.00      1.00      1.00      4816

[[4721    0]
 [   9   86]]
```

Este proceso se ha repetido para el análisis de las 5 iteraciones de *Cross-validation* comentadas a lo largo del proyecto. Por cada Iteración, ya fuera utilizando ResNet o CancerNet, el set de imágenes se construía de la misma manera. Se hacía uso de un 80% de las imágenes correspondientes a hueso o músculo para el entrenamiento y el 20% restante para la validación.

En la introducción de este apartado para la creación de directorios, solo se mencionan los dos bancos de datos disponibles: Imagenes_DATOS_CASO1 e Imagenes_DATOS_CASO2. Si bien, habrá hasta 10 casos posibles pues son los que se han utilizado para la evaluación de las distintas iteraciones.Los siguientes sets de datos están compuestos por los mismos subdirectorios y por tanto el proceso a realizar es el mismo que para el primer caso. Cobra relevancia la diferenciación para la segmentación de imágenes previa que se realiza en el proyecto.

En primer lugar se introducen los datos que se quieren cargar y se establecen los límites de las búsquedas de las secciones. Estos límites se corresponden con los valores de las casillas de cada matriz en los DATOS. Por experimentación se ha comprobado que los músculos y huesos están entre 500 y 2200, es más, podríamos incluso restringir los valores a 800 a 1600, ya que por encima de 1600 es seguro hueso.

```
%% Inicialización %%
  clear all
  clc
  close all
%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Introducción de datos por parte del usuario %%
  Nombre_Datos='DATOS_CASO2.mat';    % Introducción del nombre de los datos .mat
  Limite_I=800;                      % Introducción del límite inferior
  Limite_S=1600;                     % Introducción del limite superior
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Se inicializarán entonces las variables necesarias y se procederá a la lectura de los datos. En este punto se han incluido unas líneas que deberán ser modificadas según los nombres de las variables que se carguen.Por ejemplo, si carga DATOS_CASO1.mat sus variables asociadas serán DATOS1, GT_HUESO1 y GT_MUSCULO1 si cargamos los datos DATOS_CASO2.mat, deberemos cambiar estas líneas a DATOS2, GT_HUESO2 y GT_MUSCULO2 y así, sucesivamente.

```
%% Datos %%
  fprintf('\nCargados datos: %s',Nombre_Datos);
  load(Nombre_Datos);

  Y=DATOS2; %%%%%%%%%%%%%%%%%%%%%%%%%->MODIFICAR EN CASO DE SER NECESARIO
  GT_Clase_A=GT_HUESO2; %%%%%%%%%%%%%%%%%->MODIFICAR EN CASO DE SER NECESARIO
  GT_Clase_B=GT_MUSCULO2;%%%%%%%%%%%%%%%%%->MODIFICAR EN CASO DE SER NECESARIO

  imshow(Y,[])
  fprintf('\n-> Datos cargados correctamente');
  cont=1;
  cont_H=0;
  cont_M=0;
%%%%%%%%%%%
```

Dependiendo del archivo que se suba se creará una carpeta u otra. Si subimos el archivo DATOS_CASO1 se creará en el directorio actual una carpeta con su mismo nombre donde se irán guardando todas las secciones calculadas. Si la carpeta ya existe con ese nombre se mostrara un mensaje por pantalla advirtiendo de que las imágenes de esa carpeta serán sobrescritas.

La creación de los directorios de las imágenes se realizará de la siguiente forma:

```
%% Creación del directorio de imágenes %%
  DirectorioA=cd();
  DirectorioN=[DirectorioA,'\Imagenes_',Nombre_Datos(1:end-4)];
  DirectorioN1=[DirectorioA,'\Imagenes_',Nombre_Datos(1:end-4),'\Hueso'];    %% MODIFICACION v2
  DirectorioN2=[DirectorioA,'\Imagenes_',Nombre_Datos(1:end-4),'\Musculo'];  %% MODIFICACION v2
  [A,B,C]=mkdir(DirectorioN);
  if isempty(B)==0 && strcmp(B,'Directory already exists.')==1
      fprintf('\n----> Directorio Imagenes%s ya existe, se reemplazaran todas las imagenes',['\',Nombre_Datos(1:
  else
     fprintf('\n-> Directorio Imagenes%s creado correctamente',['\',Nombre_Datos(1:end-4)]);
  end
  mkdir(DirectorioN1);    %% MODIFICACION v2
  mkdir(DirectorioN2);    %% MODIFICACION v2
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Cálculos %%
  % Búsqueda de valores mayores de 500 %
    [filal,columnal,valoresl]=find(Y>Limite_I & Y<Limite_S);
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Y los cálculos descritos en la sección 5.1.1 del proyecto:

```
%% Cálculos %%
  % Búsqueda de valores mayores de 500 %
    [filal,columnal,valoresl]=find(Y>Limite_I & Y<Limite_S);
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  % Recorrido de la imagen %
    L_Filas=length(Y(:,1));
    L_Columnas=length(Y(1,:));
    for i=[1:1:length(valoresl)]
        if filal(i)-24>0 && filal(i)+25<L_Filas && columnal(i)-24>0 && columnal(i)+25<L_Columnas
           Seccion=double(Y(filal(i)-24:1:filal(i)+25,columnal(i)-24:1:columnal(i)+25));
           % Clasificación si hueso o músculo %
             % Clase A (Clasificacion=1) -> HUESO%
               if GT_Clase_A(filal(i),columnal(i))==1 % Sí es hueso (1)
                   Matriz_Secciones{cont,1}=i;
                   Matriz_Secciones{cont,2}=Seccion;
                   Matriz_Secciones{cont,3}=1;
                   cont=cont+1;
                   cont_H=cont_H+1;
               else                                  % No es hueso (0) %% MODIFICACION v2
                   Matriz_Secciones{cont,1}=i;
                   Matriz_Secciones{cont,2}=Seccion;
                   Matriz_Secciones{cont,3}=0;
                   cont=cont+1;
               end
             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
            % Clase B (Clasificacion=2) -> MUSCULO%
              if GT_Clase_B(filal(i),columnal(i))==1  % Sí es músculo (2)
                  Matriz_Secciones{cont,1}=i;
                  Matriz_Secciones{cont,2}=Seccion;
                  Matriz_Secciones{cont,3}=2;
                  cont=cont+1;
                  cont_M=cont_M+1;
              else                                    % No es músculo (3)
                  Matriz_Secciones{cont,1}=i;
                  Matriz_Secciones{cont,2}=Seccion;
                  Matriz_Secciones{cont,3}=3;
                  cont=cont+1;
              end
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        end
    end
%%%%%%%%%%%%%%%%%%%%%%%%%%
cd(DirectorioA)
fprintf('\n\nSeciones totales: %d',cont);
fprintf('\n     Secciones tipo hueso: %d',cont_H);
fprintf('\n     Secciones tipo músculo: %d',cont_M);

```

Finalmente el guardado de las imágenes:

```
%% Guardado de imagenes hueso %%
  fprintf('\n\n->Guardando imágenes de hueso ...');
  DirectorioI=[DirectorioA,'\Imagenes_',Nombre_Datos(1:end-4),'\Hueso'];
  Directorio_0=[DirectorioI,'\0'];
  mkdir(Directorio_0);
  cd(Directorio_0);
  for i=[1:1:length(Matriz_Secciones)]                    % No es hueso (0)
      if Matriz_Secciones{i,3}==0
        Seccion=Matriz_Secciones{i,2};
        imwrite(Seccion./2200,['Imagen_',num2str(i),'.png']);
      end
  end

  Directorio_1=[DirectorioI,'\1'];
  mkdir(Directorio_1);
  cd(Directorio_1);
  for i=[1:1:length(Matriz_Secciones)]                    % Sí es hueso (1)
      if Matriz_Secciones{i,3}==1
        Seccion=Matriz_Secciones{i,2};
        imwrite(Seccion./2200,['Imagen_',num2str(i),'.png']);
      end
  end
  fprintf('\n->Imagenes de hueso guardadas correctamente.');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%% Guardada de imagenes musculo %%
  fprintf('\n\n->Guardando imágenes de músculo ...');
  DirectorioI=[DirectorioA,'\Imagenes_',Nombre_Datos(1:end-4),'\Musculo'];
  Directorio_0=[DirectorioI,'\0'];
  mkdir(Directorio_0);
  cd(Directorio_0);
  for i=[1:1:length(Matriz_Secciones)]                          % No es músculo (3)
      if Matriz_Secciones{i,3}==3
        Seccion=Matriz_Secciones{i,2};
        imwrite(Seccion./2200,['Imagen_',num2str(i),'.png']);
      end
  end

  Directorio_1=[DirectorioI,'\1'];
  mkdir(Directorio_1);
  cd(Directorio_1);
  for i=[1:1:length(Matriz_Secciones)]                          % Sí es músculo (2)
      if Matriz_Secciones{i,3}==2
        Seccion=Matriz_Secciones{i,2};
        imwrite(Seccion./2200,['Imagen_',num2str(i),'.png']);
      end
  end
  fprintf('\n->Imagenes de músculo guardadas correctamente.');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Proceso de clasificación

Dentro de esa primera carpeta principal se crearán posteriormente dos subdirectorios: Hueso y Músculo.

La forma de obtención de las secciones para poder generar dos carpetas (llamadas 1 y 0) de cada clasificación (hueso y músculo) es como sigue:

Clasificación hueso o no hueso:

Así primero se realiza la clasificación de las secciones según sean hueso o no hueso. Usando los datos de GT_HUESO se clasifican las secciones y todas son guardadas en la Matriz_Secciones:

Matriz_Secciones={Nº de la sección, Sección, Clasificación};

Clasificación músculo o no músculo:

Análogamente se realiza la clasificación de las secciones según sean músculo o no músculo. Usando los datos de GT_MUSCULO se clasifican las secciones y todas son guardadas en la Matriz_Secciones:

Matriz_Secciones= {Nº de la sección, Sección, Clasificación};

El término Clasificación podrá tener los siguientes valores:

- 0 si es una sección NO HUESO
- 1 si es una sección SI HUESO
- 2 si es una sección SI MÚSCULO
- 3 si es una sección NO MÚSCULO

El método empleado para la clasificación anterior es el siguiente:

a) Se buscarán las secciones de la imagen que estén entre los limites impuestos al principio.

b) Segmentando la imagen en secciones de 50x50 alrededor de los puntos recogidos en el apartado a)

c) se clasificará esa sección de la imagen comparándola con la imagen original (GT_HUESO y GT_MUSCULO).

d) Se guardará en una matriz (Matriz_Secciones) los datos de todas esas secciones Matriz_Secciones={Nº, Seccion, clasificación de la seccion}

Así en una línea cualquiera de esa matriz podremos encontrar {452, [50x50], 1}, siendo esta la sección número 452, los datos de la sección [50x50] y su clasificación. De nuevo todas estas explicaciones pueden versecodificadas en el segundo apartado del Anexo.

Proceso de guardado

Para un conjunto de datos se creará una carpeta con su nombre y en ella se crearán a su vez dos carpetas, una llamada músculo y otra hueso. En cada una de estas carpetas se crearán otras dos, llamadas 0 y 1. A partir de aquí solo hay que guardar las secciones ya clasificadas, cada una en su carpeta:

- Carpeta hueso:
  - Carpeta 1: Irán las secciones con clasificación 1 (sí es hueso).
  - Carpeta 0: Irán las secciones con clasificación 0 (no es hueso).

- Carpeta músculo:
  - Carpeta 1: Irán las secciones con clasificación 2 (sí es músculo).
  - Carpeta 0: Irán las secciones con clasificación 3 (no es músculo).

Codificación ResNet

Config.py

```
# coding=utf-8
# Trabajo Fin de Grado: Clasificación de lesiones de piel usando redes neuronales convolucionales en Python
# Álvaro Artola Moreno
# Script: config.py
# import de los paquetes necesarios
import os

# Inicialización de la ruta al directorio de entrada original de las imágenes
ORIG_ENTRADA_DATASET = "clasificacion/Imagenes_DATOS_CASO1"

# Inicialización de la ruta base para el nuevo directorio
# que contendrá las imágenes después de llevar a cabo las divisiones
# en entrenamiento y pruebas
RUTA_BASE = "clasificacion"

# División del directorio base en los subdirectorios: entrenamiento,
# validación y pruebas.
RUTA_ENT = os.path.sep.join([RUTA_BASE, "entrenamiento"])
RUTA_VAL = os.path.sep.join([RUTA_BASE, "validacion"])
RUTA_PRUEBA = os.path.sep.join([RUTA_BASE, "pruebas"])

# Definición de la cantidad de datos que se emplearán en el entrenamiento
DIV_ENT = 0.8

# La cantidad de datos de validación será un porcentaje de los datos
# de entrenamiento
DIV_VAL = 0.1
```

En el script:

- Se define la ruta del directorio original de imágenes que se utilizará (línea 9).
- Se establece la ruta base del dataset (línea 14).
- Se establecen las rutas donde se almacenarán las salidas del entrenamiento, la validación y las pruebas en sus respectivos directorios. El archivo construccion_dataset.py será el encargado de desarrollar esta tarea (líneas 18-20).
- Definición de la división entre entrenamiento y pruebas donde el 80% de la información será empleada para el entrenamiento, mientras que el 20% restante se utilizará para las pruebas (línea 23).
- Se establece la división de validación. Del 80 % del porcentaje destinado al entrenamiento, se tomará un 10% para la validación (línea 27).

Construccion_dataset.py

```
1  # coding=utf-8
2  # Trabajo Fin de Grado: Clasificación de lesiones de piel usando redes neuronales convolucionales en Python
3  # Álvaro Artola Moreno
4  # Script: construccion_dataset.py
5
6  # import de los paquetes necesarios
7  from busquedaImagenes_py import config
8  from imutils import paths
9  import random
10 import shutil
11 import os
12
13 #selección de las rutas de acceso a todas las imágenes de entrada
14 #en el directorio de entrada original y mezcla de todas ellas
15 rutasImagenes = list(paths.list_images(config.ORIG_ENTRADA_DATASET))
16 random.seed(42)
17 random.shuffle(rutasImagenes)
```

2. Dividirá las rutas de las imágenes en entrenamiento, validación y pruebas.

3. Creará tres nuevos subdirectorios en el directorio clasificacion/: entrenamiento/, validacion/ y pruebas/.

4. Automáticamente copiará las imágenes en sus correspondientes directorios.

Los paquetes necesarios se importan de las líneas 7 a la 11 y de la 15 a la 17 es donde se realiza la mezcla de imágenes.

A continuación, se dividen los datos:

```
19 # division de las imágenes para entrenamiento y pruebas
20 i = int(len(rutasImagenes) * config.DIV_ENT)
21 rutasEntrenamiento = rutasImagenes[:i]
22 rutasPrueba = rutasImagenes[i:]
23
24 # parte de los datos de entrenamiento serán utilizados para la validación
25 i = int(len(rutasEntrenamiento) * config.DIV_VAL)
26 rutasValidacion = rutasEntrenamiento[:i]
27 rutasEntrenamiento = rutasEntrenamiento[i:]
28
```

Para la división de los datos, primero se calculan los índices de la división entrenamiento/pruebas. Después usando dicho índice y extrayendo un conjunto de elementos de las matrices -empaquetándolos en otras (array slicing)-, dividimos los datos en rutasValidacion y rutasEntrenamiento.

De nuevo se calculan los índices pero en esta ocasión de entrenamiento/validación desde la rutasEntrenamiento. Seguidamente, se dividen las rutas de las imágenes en rutasValidacion y rutasEntrenamiento (líneas 26 y 27). En efecto, las rutasEntrenamiento son reasignadas porque tal y como se mencionó anteriormente, de ese 80% que se

empleará en el entrenamiento un 10% se destinará a la validación.

Una vez organizadas las rutas de las imágenes en sus respectivas divisiones se definen los dataset a construir (líneas 29-34).

```
29  # definición de los dataset a construir
30  datasets = [
31      ("entrenamiento", rutasEntrenamiento, config.RUTA_ENT),
32      ("validacion", rutasValidacion, config.RUTA_VAL),
33      ("pruebas", rutasPrueba, config.RUTA_PRUEBA)
```

Se han creado tres tuplas (datasets) que contienen:

1. El nombre de la división.

2. La ruta para la división.

3. La ruta del directorio de salida.

Con toda esta información, puede empezarse a iterar por cada uno de los datasets:

```
36  # bucle sobre los dataset
37  for (tipo, rutasImagenes, salida) in datasets:
38      # se muestra qué división de los datos estamos creando.
39      print("[INFO] construyendo la división de '{}' ".format(tipo))
40
41      # creamos el directorio de salida en caso de que no exista
42      if not os.path.exists(salida):
43          print("[INFO] 'creando el directorio {}'".format(salida))
44          os.makedirs(salida)
45
46      # bucle sobre las rutas de imagen de la entrada
47      for rutaDeEntrada in rutasImagenes:
48          # extracción del nombre del archivo de la imagen de
49          # entrada junto con su etiqueta correspondiente
50          nombreArchivo = rutaDeEntrada.split(os.path.sep)[-1]
51          etiqueta = rutaDeEntrada.split(os.path.sep)[-2]
52
53          # construcción de la ruta al directorio de etiquetas
54          rutaEtiqueta = os.path.sep.join([salida, etiqueta])
55
56          # creamos el directorio de salida de la etiqueta en caso
57          # de que este no existiera
58          if not os.path.exists(rutaEtiqueta):
59              print("[INFO] 'creando el directorio {}'".format(rutaEtiqueta))
60              os.makedirs(rutaEtiqueta)
61
62          # construcción de la ruta para la imagen destino y
63          # copia de la imagen en si.
64          p = os.path.sep.join([rutaEtiqueta, nombreArchivo])
65          shutil.copy2(rutaDeEntrada, p)
```

En la línea 37 se empieza a iterar por el tipo de dataset, las rutas de las imágenes y el directorio de salida.

Si el directorio de salida no existe, se crea (líneas 42-44).

A partir de ahí, se itera por las rutas en si, empezando en la línea 47. En el bucle:

- Se extrae el nombre del fichero y la etiqueta (líneas 50 y 51).
- Se crea el subdirectorio si es necesario (líneas 54-60)
- Se copia el archivo de imágenes actual en el subdirectorio correspondiente.

Modelo_entrenamiento.py

```
1   # coding=utf-8
2   # Trabajo Fin de Grado: Clasificación de lesiones de piel usando redes neuronales convolucionales en Python
3   # Álvaro Artola Moreno
4   # Script: modeloo_entrenamiento.py
5
6   # Configuración del backend de matplotlib para poder guardar las imágenes
7   import matplotlib
8   matplotlib.use("Agg")
9
10  # import de los paquetes necesarios
11  from keras.preprocessing.image import ImageDataGenerator
12  from keras.callbacks import LearningRateScheduler
13  from keras.optimizers import SGD
14  from busquedaImagenes_py.resnet import ResNet
15  from busquedaImagenes_py import config
16  from sklearn.metrics import classification_report
17  from imutils import paths
18  import matplotlib.pyplot as plt
19  import numpy as np
20  import argparse
21
22  # Construcción del analizador de argumentos y análisis de estos.
23  ap = argparse.ArgumentParser()
24  ap.add_argument("-p", "--plot", type=str, default="plot.png",
25      help="path to output loss/accuracy plot")
26  args = vars(ap.parse_args())
27
```

La línea 14 importa la red neuronal convolucional *ResNet*.

Tenemos además un único argumento por línea de comandos que se analiza en las líneas 23-26, --plot. Con este, de forma predeterminada nuestra gráfica se ubicará en el directorio de trabajo actual y se llamará plot.png.Se detallan también los parámetros de entrenamiento y se define la función de decaimiento:

```
28  # Definición del número de épocas, tasa de aprendizaje inicial y
29  # el tamaño del lote -batch size-.
30  NUM_EPOCHS = 20
31  INIT_TA = 1e-1
32  BS = 32
33
34  def decadencia_polinomica(epoch):
35      # Inicialización del número máximo de épocas, la tasa de aprendizaje base
36      # y la potencia del polinomio.
37      maxEpochs = NUM_EPOCHS
38      baseTA = INIT_TA
39      power = 1.0
40
41      # Cálculo de la nueva tasa de aprendizaje basada en la decadencia
42      # polinómica.
43      alpha = baseTA * (1 - (epoch / float(maxEpochs))) ** power
44
45      # Devolución de la nueva tasa de aprendizaje
46      return alpha
```

En las líneas 30-32 se definen el número de épocas, la tasa inicial de aprendizaje y el tamaño del lote (*batch size*).

La función de decaimiento, definida entre las líneas 34 y 46, nos ayudará a que la tasa de aprendizaje inicial vaya decayendo tras cada época. Se está estableciendo una potencia de 1.0, la cual convierte la decadencia polinómica en una decadencia lineal. El cálculo se lleva a cabo en la línea 43 y posteriormente es devuelto en la 46.

Lo siguiente es extraer el número de rutas de imágenes en los dataset de validación y pruebas:

```
48  # Determinación del número de rutas en entrenamiento, validacion
49  # y directorios de prueba
50  totalEnt = len(list(paths.list_images(config.RUTA_ENT)))
51  totalVal = len(list(paths.list_images(config.RUTA_VAL)))
52  totalPru = len(list(paths.list_images(config.RUTA_PRUEBA)))
53
```

A continuación se aplica *data augmentation* (aumento de datos):

```
54  # Inicialización del objeto de aumento de datos de entrenamiento
55  aumEnt = ImageDataGenerator(
56      rescale=1 / 255.0,
57      rotation_range=20,
58      zoom_range=0.05,
59      width_shift_range=0.05,
60      height_shift_range=0.05,
61      shear_range=0.05,
62      horizontal_flip=True,
63      fill_mode="nearest")
64
65  # Inicialización del objeto de aumento de datos de validación (y prueba)
66  aumVal = ImageDataGenerator(rescale=1 / 255.0)
67
68  # Inicialización del generador del entrenamiento.
69  genEnt = aumEnt.flow_from_directory(
70      config.RUTA_ENT,
71      class_mode="categorical",
72      target_size=(64, 64),
73      color_mode="rgb",
74      shuffle=True,
75      batch_size=BS)
```

En las líneas 55-63 inicializamos el *ImageDataGenerator*, que se utilizará para aplicar el aumento de datos mediante cambios aleatorios, traducciones y volteos de cada muestra de entrenamiento.

El *ImageDataGenerator* de la validación no realizará ningún aumento de datos (línea 66). En su lugar, simplemente cambiará la escala de nuestros valores de píxeles al rango [0,1] como se ha realizado en el generador de entrenamiento. Hay que tener en cuenta que se empleará *aumVal* tanto para la validación como para las pruebas.

Seguidamente, inicializamos los generadores de entrenamiento, validación y pruebas:

```
67
68      # Inicialización del generador del entrenamiento.
69      genEnt = aumEnt.flow_from_directory(
70          config.RUTA_ENT,
71          class_mode="categorical",
72          target_size=(64, 64),
73          color_mode="rgb",
74          shuffle=True,
75          batch_size=BS)
76
77      # Inicialización del generador de la validación.
78      genVal = aumVal.flow_from_directory(
79          config.RUTA_VAL,
80          class_mode="categorical",
81          target_size=(64, 64),
82          color_mode="rgb",
83          shuffle=False,
84          batch_size=BS)
85
86      # Inicialización del generador de pruebas
87      genPru = aumVal.flow_from_directory(
88          config.RUTA_PRUEBA,
89          class_mode="categorical",
90          target_size=(64, 64),
91          color_mode="rgb",
92          shuffle=False,
93          batch_size=BS)
```

En este bloque creamos los generadores *Keras* utilizados para cargar imágenes desde un directorio de entrada.

La función *flow_from_directory* asume:

1. Que existe un directorio de entrada base para la división de los datos.

2. Y que dentro de dicho directorio, hay N subdirectorios, donde cada subdirectorio corresponde con una etiqueta de clase.

Los parámetros que estamos empleando para cada generador son propios de la documentación de preprocesamiento de Keras. En el proyeto se definen:

- *class_mode = categorical*, para asegurar que *Keras* realiza la codificación *one-hot* en las etiquetas.
- *target_size*: redimensionar las imágenes a 64 x 64 píxeles.
- *color_mode*= "*rgb*". Establecer el formato de color a "rgb"
- *shuffle*: mezclar las rutas de las imágenes solo para el generador de entrenamiento.
- Un *batch size* de 32.

Los valores escogidos para los parámetros anteriores dependerán del set de datos que se esté utilizando en cuestión. Así para las imágenes pertenecientes al cáncer de mama y las de melanoma se redimensionarán a 64x64 con un modo de color rgb. Sin embargo, para el caso de los dataset de imágenes de hueso y músculo se trabaja con imágenes de 50x50 y en escala de grises (*grayscale).*

Se puede entonces inicializar las red *ResNet* y compilar el modelo:

```
95      # Inicialización y compilación del modeloo ResNet
96      modelo = ResNet.build(64, 64, 3, 2, (3, 4, 6),
97          (64, 128, 256, 512), reg=0.0005)
98      opt = SGD(lr=INIT_TA, momentum=0.9)
99      modelo.compile(loss="binary_crossentropy", optimizer=opt,
100         metrics=["accuracy"])
```

Destacar que:

- Las imágenes son de 64 x64 x 3 (3 canales RGB)
- Hay un total de 2 clases
- La red realizará un apilamiento de 3,4 y 6 con 64, 128, 256 y 512 capas *CONV*, lo que implica que:
- La primera capa *CONV* en *ResNet*, antes de reducir las dimensiones espaciales, tendrá 64 filtros.
- Después apilará 3 conjutnos de módulos residuales. Las tres capas *CONV* en cada módulo residual aprenderán 32, 32 y 128 filros *CONV* respectivamente. Se reducirán entonces las dimensiones espaciales.
- A continuación, se apilan 4 conjuntos de módulos residuales donde cada una de las tres capas de CONV tendrá 64, 64 y 256 filtros. De nuevo, las dimensiones espaciales serán entonces reducidas.
- Finalmente, se apilan 6 conjuntos de módulos residuales, donde cada capa cONV aprendre 128, 128 y 512 filtors. Las dimensiones espaciales se reducen por última vez antes de realizar la agrupación promedio y se aplica un clasificador Softmax.

Como se comentó anteriormente dependerá del dataset con el que se esté trabajando. De nuevo para las imágenes de mama y de melanoma sí se tendrá 64x64x3 (por los canales RGB) pero en el caso de hueso y músculo será 50x50x1.

En la línea 92 se inicializa el optimizador SGD con la tasa de aprendizaje inicial por defecto de 1e-1 y un momento de 0.9.

En las líneas 93 y 94 se compila el modelo empleado haciendo uso de binary_crossentropy como función de pérdida al estar utilizando una clasificación de dos clases. En caso de que se lleven a cabo más de dos habría que utilizar la categorical_crossentropy.

Una vez establecido lo anterior se puede entrenar el modelo:

```
102  # Definición del conjunto de devoluciones -callbacks- de llamada
103  # y ajuste del modelo
104  callbacks = [LearningRateScheduler(decadencia_polinomica)]
105  H = modelo.fit_generator(
106      genEnt,
107      steps_per_epoch=totalEnt // BS,
108      validation_data=genVal,
109      validation_steps=totalVal // BS,
110      epochs=NUM_EPOCHS,
111      callbacks=callbacks)
```

En la línea 104 creamos el conjunto de *callbacks*. Éstos son ejecutados al final de cada época. En el proyecto, estamos aplicando la función de decaimiento a través del *LearningRateScheduler* de *Keras* que recibe como parámetro el *schedule*: función que toma el índice de la época como entrada (entero, indexado desde 0) y la tasa de aprendizaje actual, devolviendo una nueva tasa de aprendizaje (float).

La llamada a *modelo.fit_generator* en las líneas 98-104 le indica al script que inicie el proceso de captación.El generador *genEnt* automáticamente cargará las imágenes del disco y analizará las etiquetas de clase de la ruta de la imagen.Análogamente, *genVal* realizará el mismo proceso pero solo con los datos de validación.Se pueden evaluar los resultados de dataset de pruebas:

```
113  # Reinicio del generador de pruebas y posterior uso del modelo entrenado
114  # para hacer predicciones sobre los datos.
115  print("[INFO] evaluando la red...")
116  genPru.reset()
117  predIdxs = modelo.predict_generator(genPru,
118      steps=(totalPru // BS) + 1)
119
120  # Para cada imagen en el conjunto de pruebas, hay que encontrar el índice
121  # de la etiqueta con la maypor probabilidad pronosticada correspondiente
122  predIdxs = np.argmax(predIdxs, axis=1)
123
124  # Muestra de un informe de clasificación bien formateado
125  print(classification_report(genPru.classes, predIdxs,
126      target_names=genPru.class_indices.keys()))
127
```

Técnicamente la línea 116 puede ser eliminada, sin embargo, es una buena praxis el restablecer los generadores de *Keras* antes de la evaluación.

Para evaluar el modelo se llevarán a cabo predicciones sobre los datos de prueba y posteriormente se hallará la etiqueta con la mayor probabilidad para cada imagen en el conjunto de prueba (líneas 117-122)

Después, se imprimirá la *classification_report* (líneas 125 y 126).

Por último, se trazarán los datos de entrenamiento:

```
128  # Trazado de la pérdida en el entrenamiento y la precisión
129  N = NUM_EPOCHS
130  plt.style.use("ggplot")
131  plt.figure()
132  plt.plot(np.arange(0, N), H.history["loss"], label="perdida_entrenamiento")
133  plt.plot(np.arange(0, N), H.history["val_loss"], label="perdida_validacion")
134  plt.plot(np.arange(0, N), H.history["acc"], label="precision_entrenamiento")
135  plt.plot(np.arange(0, N), H.history["val_acc"], label="precision_validacion")
136  plt.title("Pérdida y precisión en el análisis del Dataset")
137  plt.xlabel("Epoch #")
138  plt.ylabel("Perdida/Precision")
139  plt.legend(loc="lower left")
140  plt.savefig(args[u"plot"])
```

ResNet.py

```
# imports de los paquetes necesarios
from keras.layers.normalization import BatchNormalization
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import AveragePooling2D
from keras.layers.convolutional import MaxPooling2D
from keras.layers.convolutional import ZeroPadding2D
from keras.layers.core import Activation
from keras.layers.core import Dense
from keras.layers import Flatten
from keras.layers import Input
from keras.models import Model
from keras.layers import add
from keras.regularizers import l2
from keras import backend as K
```

Una vez se han establecido los imports necesarios se define el método estático de la red neuronal donde se describen los distintos bloques que se van creando y las convoluciones que se llevan a cabo en cada uno de ellos.

```
class ResNet:
    @staticmethod
    def residual_module(data, K, stride, chanDim, red=False,
        reg=0.0001, bnEps=2e-5, bnMom=0.9):
        # la rama de acceso directo del módulo ResNet
        # se inicializa con los datos de entrada
        shortcut = data

        # el primer bloque del módulo son las CONVs 1x1
        bn1 = BatchNormalization(axis=chanDim, epsilon=bnEps,
            momentum=bnMom)(data)
        act1 = Activation("relu")(bn1)
        conv1 = Conv2D(int(K * 0.25), (1, 1), use_bias=False,
            kernel_regularizer=l2(reg))(act1)

        # el segundo bloque del módulo son las CONVs 3x3
        bn2 = BatchNormalization(axis=chanDim, epsilon=bnEps,
            momentum=bnMom)(conv1)
        act2 = Activation("relu")(bn2)
        conv2 = Conv2D(int(K * 0.25), (3, 3), strides=stride,
            padding="same", use_bias=False,
            kernel_regularizer=l2(reg))(act2)

        # el tercero es de nuevo un set de CONVs 3x3
        bn3 = BatchNormalization(axis=chanDim, epsilon=bnEps,
            momentum=bnMom)(conv2)
        act3 = Activation("relu")(bn3)
        conv3 = Conv2D(K, (1, 1), use_bias=False,
            kernel_regularizer=l2(reg))(act3)

        # en caso de querer reducir el tamaño espacial se aplica
        # una capa de CONV al acceso directo
        if red:
            shortcut = Conv2D(K, (1, 1), strides=stride,
                use_bias=False, kernel_regularizer=l2(reg))(act1)

        # se suman la rama y la CONV final
        x = add([conv3, shortcut])

        # devuelve la suma como la salida del módulo de ResNet
        return x
```

```
    @staticmethod
    def build(width, height, depth, classes, stages, filters,
        reg=0.0001, bnEps=2e-5, bnMom=0.9):
        # inicializa la forma de entrada para ser "channel last"
        # y la propia dimensión de los canales
        inputShape = (height, width, depth)
        chanDim = -1

        # si es utiliza "channels first" se actualiza la forma de entrada
        # y la dimensión de los canales
        if K.image_data_format() == "channels_first":
            inputShape = (depth, height, width)
            chanDim = 1

        # se establece la entrada y se aplica el BN
        inputs = Input(shape=inputShape)
        x = BatchNormalization(axis=chanDim, epsilon=bnEps,
            momentum=bnMom)(inputs)

        # se aplica CONV => BN => ACT => POOL para reducir el tamaño espacial
        x = Conv2D(filters[0], (5, 5), use_bias=False,
            padding="same", kernel_regularizer=l2(reg))(x)
        x = BatchNormalization(axis=chanDim, epsilon=bnEps,
            momentum=bnMom)(x)
        x = Activation("relu")(x)
        x = ZeroPadding2D((1, 1))(x)
        x = MaxPooling2D((3, 3), strides=(2, 2))(x)

        # bucle sobre el número de etapas
        for i in range(0, len(stages)):
            # inicialización del paso y aplicación de un módulo residual
            # usado para reducir la dimensión espacial del volumen de entrada
            stride = (1, 1) if i == 0 else (2, 2)
            x = ResNet.residual_module(x, filters[i + 1], stride,
                chanDim, red=True, bnEps=bnEps, bnMom=bnMom)

            # bucle sobre el número de capas en el escenario
            for j in range(0, stages[i] - 1):
                # aplicación de un módulo ResNet
                x = ResNet.residual_module(x, filters[i + 1],
                    (1, 1), chanDim, bnEps=bnEps, bnMom=bnMom)

        # aplicación de  BN => ACT => POOL
        x = BatchNormalization(axis=chanDim, epsilon=bnEps,
            momentum=bnMom)(x)
        x = Activation("relu")(x)
        x = AveragePooling2D((8, 8))(x)

        # clasificador softmax
        x = Flatten()(x)
        x = Dense(classes, kernel_regularizer=l2(reg))(x)
        x = Activation("softmax")(x)

        # creación del modelo
        model = Model(inputs, x, name="resnet")

        # devolución de la arquitectura de red construida
        return model
```

Codificación CancerNet

Config.py

```
# coding=utf-8
# Trabajo Fin de Grado: Clasificación de lesiones
# de piel usando redes neuronales convolucionales en Python
# Álvaro Artola Moreno
# Script: config.py
import os

# Inicialización de la ruta al directorio de entrada original de las imágenes
ORIG_ENTRADA_DATASET = "clasificacion/Hueso"

# Inicialización de la ruta base para el nuevo directorio destino
# que contendrá las imágenes después de llevar a cabo las divisiones
# en entrenamiento y pruebas
RUTA_BASE = "clasificacion"

# Inicialización de la ruta base para el nuevo directorio
# que contendrá las imágenes después de llevar a cabo las divisiones
# en entrenamiento y pruebas
RUTA_ENT = os.path.sep.join([RUTA_BASE, "entrenamiento"])
RUTA_VAL = os.path.sep.join([RUTA_BASE, "validacion"])
RUTA_PRUEBA = os.path.sep.join([RUTA_BASE, "pruebas"])

# Definición de la cantidad de datos que se emplearán en el entrenamiento
DIV_ENT = 0.25

# La cantidad de datos de validación será un porcentaje de los datos
# de entrenamiento
DIV_VAL = 0.1
```

Modelo_entrenamiento.py

```
# coding=utf-8
# Trabajo Fin de Grado: Clasificación de lesiones
# de piel usando redes neuronales convolucionales en Python
# Álvaro Artola Moreno
# Script: modelo_entrenamiento.py

# Configuración del backend de matplotlib para poder guardar las imágenes
import matplotlib
matplotlib.use("Agg")

# import de los paquetes necesarios
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import LearningRateScheduler
from keras.optimizers import Adagrad
from keras.utils import np_utils
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from pyimagesearch.cancernet import CancerNet
from pyimagesearch import config
from imutils import paths
import matplotlib.pyplot as plt
import numpy as np
import argparse
import os

# Construcción del analizador de argumentos y análisis de estos.
ap = argparse.ArgumentParser()
ap.add_argument("-p", "--plot", type=str, default="plot.png",
    help="path to output loss/accuracy plot")
args = vars(ap.parse_args())
```

1. *Matplotlib*: paquete de trazado científico que es el estándar de facto para Python. Se configura para usar el backend "Agg" de modo que puedan guardarse los gráficos de entrenamiento en el disco. (líneas 8 y 9)
2. *Keras*: se utiliza la ImageDataGenerator, LearningRateScheduler, el oprimizador Adagrad y np_utils.
3. *Sklearn*: de scikit-learn es necesario implementar classification_report y confusion_matrix.
4. *BusquedaImagenes_py*: se pone en funcionamiento cancerNet (entrenamiento y evaluación). Además, también es necesario obtener el archivo de configuración para capturar las rutas a las tres divisiones de datos.
5. *Imutils*: se usarán los módulos paths capturar rutas a cada una de las imágenes.
6. *Numpy*: herramienta utilizada para el procesamiento numérico con Python.
7. *Python*: tanto arparse como os están integrados en las instalaciones de Python. Se utiliza argparse para analizar un argumento por línea de comandos.

Una vez se han importado las librerías necesarias y se han parseado los argumentos por línea de comandos, se definen los parámetros de entrenamiento, incluyendo las rutas de las imágenes de entrenamiento teniendo en cuenta el desequilibrio de clase.

```
31
32  # Definición del número de épocas, tasa de aprendizaje inicial y
33  # el tamaño del lote -batch size-.
34  NUM_EPOCHS = 10
35  INIT_LR = 1e-2
36  BS = 32
37
38  # Determinación del número de rutas en entrenamiento, validacion
39  # y directorios de prueba
40  rutasEntrenamiento = list(paths.list_images(config.RUTA_ENT))
41  totalEnt = len(rutasEntrenamiento)
42  totalVal = len(list(paths.list_images(config.RUTA_VAL)))
43  totalPru = len(list(paths.list_images(config.RUTA_PRUEBA)))
44
45  # sesgo de los datos etiquetados
46  etiquetasEntrenamiento = [int(p.split(os.path.sep)[-2]) for p in rutasEntrenamiento]
47  etiquetasEntrenamiento = np_utils.to_categorical(etiquetasEntrenamiento)
48  clasesTotales = etiquetasEntrenamiento.sum(axis=0)
49  pesoDeClases = clasesTotales.max() / clasesTotales
50
```

Se definen el número de épocas, la tasa de aprendizaje inicial y el tamaño de los lotes (líneas 34-36). A partir de ahí, tomamos las rutas de imágenes de entrenamiento y se determina el número total de imágenes en cada una de las divisiones (líneas 40-43).

Continuamos y se calcula el valor de cada clase – *pesoDeClases* – para los datos de entrenamiento teniendo en cuenta el desequilibrio/sesgo de la clase.

A continuación se inicializa el objeto para el aumento de datos:

```
51  # Inicialización del objeto de aumento de datos de entrenamiento
52  aumEnt = ImageDataGenerator(
53      rescale=1 / 255.0,
54      rotation_range=20,
55      zoom_range=0.05,
56      width_shift_range=0.1,
57      height_shift_range=0.1,
58      shear_range=0.05,
59      horizontal_flip=True,
60      vertical_flip=True,
61      fill_mode="nearest")
62
63  # Inicialización del objeto de aumento de datos de validación (y prueba)
64  aumVal = ImageDataGenerator(rescale=1 / 255.0)
65
```

El objeto de aumento de datos es el *aumEnt* (líneas 52-61). Las rotaciones aleatorias, los cambios, las cizallas y los giros se aplicarán a los datos a medida que se generen. El escalamiento de las intensidades de píxeles de imagen en el rango [0,1] se realiza mediante el generador *aumEnt* y el *aumVal* (línea 64).

Se pueden entonces inicializar cada generador:

```
66    # Inicialización del generador del entrenamiento.
67    trainGen = aumEnt.flow_from_directory(
68        config.RUTA_ENT,
69        class_mode="categorical",
70        target_size=(50, 50),
71        color_mode="grayscale",
72        shuffle=True,
73        batch_size=BS)
74
75    # Inicialización del generador de la validación.
76    valGen = aumVal.flow_from_directory(
77        config.RUTA_VAL,
78        class_mode="categorical",
79        target_size=(50, 50),
80        color_mode="grayscale",
81        shuffle=False,
82        batch_size=BS)
83
84    # Inicialización del generador de pruebas
85    genPrueba = aumVal.flow_from_directory(
86        config.RUTA_PRUEBA,
87        class_mode="categorical",
88        target_size=(50, 50),
89        color_mode="grayscale",
90        shuffle=False,
91        batch_size=BS)
92
```

Cada generador proporcionará lotes de imágenes bajo demanda, como se indica en el parámetro *batch_size*.

```
93    # Inicialización y compilación del modeloo CancerNet
94    model = CancerNet.build(width=50, height=50, depth=1,
95        classes=2)
96    opt = Adagrad(lr=INIT_LR, decay=INIT_LR / NUM_EPOCHS)
97    model.compile(loss="binary_crossentropy", optimizer=opt,
98        metrics=["accuracy"])
99
100   # Ajuste del modelo
101   H = model.fit_generator(
102       trainGen,
103       steps_per_epoch=totalEnt // BS,
104       validation_data=valGen,
105       validation_steps=totalVal // BS,
106       class_weight=pesoDeClases,
107       epochs=NUM_EPOCHS)
```

El modelo se inicializa con el optimizador *Adagrad* (líneas 94-98).

Mediante la llamada al método de Keras *fit_generator*, el proceso de entrenamiento se inicializa. Hacer uso de este método permite que los datos de las imágenes residan en el disco en lotes, en lugar de tener el conjunto de datos completo en la memoria RAM durante el entrenamiento.

Después de haberse completado el entrenamiento, se evalúa el modelo en los datos de prueba:

```
109  # Reinicio del generador de pruebas y posterior uso del modelo entrenado
110  # para hacer predicciones sobre los datos.
111  print("[INFO] evaluando la red...")
112  genPrueba.reset()
113  predIdxs = model.predict_generator(genPrueba,
114      steps=(totalPru // BS) + 1)
115
116  # Para cada imagen en el conjunto de pruebas, hay que encontrar el índice
117  # de la etiqueta con la maypor probabilidad pronosticada correspondiente
118  predIdxs = np.argmax(predIdxs, axis=1)
119
120  # Muestra de un informe de clasificación bien formateado
121  print(classification_report(genPrueba.classes, predIdxs,
122      target_names=genPrueba.class_indices.keys()))
123
```

Las líneas 113 y 114 realizan predicciones de todos los datos de entrenamiento (de nuevo utilizando el objeto generador).

Los índices más altos son tomados en cada muestra (línea 118) y después se imprime un informe de clasificación – classification_report – (líneas 121 y 122).

Una vez realizado todo lo anterior, se calcula también la matriz de confusión – *confusion_matrix* –. Y por último, se genera y se guardan las gráficas del entrenamiento:

```
124  # Cálculo de la matriz de confusion y uso de ella
125  cm = confusion_matrix(genPrueba.classes, predIdxs)
126
127  # Impresion de la matriz de confusion
128  print(cm)
129
130  # Trazado de la perdida en el entrenamiento y la precision
131  N = NUM_EPOCHS
132  plt.style.use("ggplot")
133  plt.figure()
134  plt.plot(np.arange(0, N), H.history["loss"], label="entrenamiento_perdida")
135  plt.plot(np.arange(0, N), H.history["val_loss"], label="validacion_perdida")
136  plt.plot(np.arange(0, N), H.history["acc"], label="precision_entrenamiento")
137  plt.plot(np.arange(0, N), H.history["val_acc"], label="precision_validacion")
138  plt.title("Perdida en el entrenamiento y precision en el Dataset")
139  plt.xlabel("Epoch #")
140  plt.ylabel("Perdida/Precision")
141  plt.legend(loc="lower left")
142  plt.savefig(args["plot"])
```

CancerNet.py

Las importaciones de Keras se enumeran en las líneas 2-10. Se utilizará la API *secuential* de Keras para crear CancerNet:

```
1  # imports de los paquetes necesarios
2  from keras.models import Sequential
3  from keras.layers.normalization import BatchNormalization
4  from keras.layers.convolutional import SeparableConv2D
5  from keras.layers.convolutional import MaxPooling2D
6  from keras.layers.core import Activation
7  from keras.layers.core import Flatten
8  from keras.layers.core import Dropout
9  from keras.layers.core import Dense
10 from keras import backend as K
11
12 class CancerNet:
13     @staticmethod
14     def build(width, height, depth, classes):
15         # inicializacion del modelo junto con el formato de entrada
16         # para ser channels last y la dimension de los canales en si
17         model = Sequential()
18         inputShape = (height, width, depth)
19         chanDim = -1
20
```

El uso de *SeparableConv2D* permite las circunvoluciones en profundidad.

En la red, el método de construcción requiere de cuatro parámetros:

- *Width* (anchura), *height* (altura) y *depth* (profundidad): aquí se especifica la forma del volumen de la imagen de entrada a la red, donde la profundidad se refiere al número de canales de colo que contiene cada imagen.
- *Classes* (clases): el número de clases que la red predecirá, son dos

Una vez realizado se inicializa el modelo (línea 17) y subsecuentemente se especifica el *inputShape* (forma de entrada) en la línea 18. Por estar haciendo uso de TensorFlow como *backend*, podemos en ese momento añadir capas. Otros *backend* que especifican "*channels_first*" requieren que *depth* (la profundidad) esté al frente de la *inputShape*

Se definen entonces las capas DEPTHWISE_CONV → RELU → POOL:

```
27         # CONV => RELU => POOL
28         model.add(SeparableConv2D(32, (3, 3), padding="same",
29             input_shape=inputShape))
30         model.add(Activation("relu"))
31         model.add(BatchNormalization(axis=chanDim))
32         model.add(MaxPooling2D(pool_size=(2, 2)))
33         model.add(Dropout(0.25))
34
35         # (CONV => RELU => POOL) * 2
36         model.add(SeparableConv2D(64, (3, 3), padding="same"))
37         model.add(Activation("relu"))
38         model.add(BatchNormalization(axis=chanDim))
39         model.add(SeparableConv2D(64, (3, 3), padding="same"))
40         model.add(Activation("relu"))
41         model.add(BatchNormalization(axis=chanDim))
42         model.add(MaxPooling2D(pool_size=(2, 2)))
43         model.add(Dropout(0.25))
44
45         # (CONV => RELU => POOL) * 3
46         model.add(SeparableConv2D(128, (3, 3), padding="same"))
47         model.add(Activation("relu"))
48         model.add(BatchNormalization(axis=chanDim))
49         model.add(SeparableConv2D(128, (3, 3), padding="same"))
50         model.add(Activation("relu"))
51         model.add(BatchNormalization(axis=chanDim))
52         model.add(SeparableConv2D(128, (3, 3), padding="same"))
53         model.add(Activation("relu"))
54         model.add(BatchNormalization(axis=chanDim))
55         model.add(MaxPooling2D(pool_size=(2, 2)))
56         model.add(Dropout(0.25))
```

Se desarrollan tres bloques con un aumento de apilamiento y número de filtros. Además se realiza normalización de lotes – *BatchNormalization* – y *dropout.*

```
58      # primer y único conjunto de capas FC => RELU
59      model.add(Flatten())
60      model.add(Dense(256))
61      model.add(Activation("relu"))
62      model.add(BatchNormalization())
63      model.add(Dropout(0.5))
64
65      # clasificador softmax
66      model.add(Dense(classes))
67      model.add(Activation("softmax"))
68
69      # devolución de la arquitectura de red construida
70      return model
```

Las capas FC → RELU y el clasificador softmax son el eje de la red. La salida del clasificador softmax serán las predicciones de los porcentajes para cada clase que el modelo predirá y finalmente el modelo es devuelto al srcipt de entrenamiento.