Detección de cáncer mediante deep learning

Trabajo de fin de grado

Curso 2019/2020

UNIVERSIDAD COMPLUTENSE MADRID
Grado en Ingeniería Matemática
Facultad de Ciencias Matemáticas

Sofía Vargas Ibarra

Tutor: Daniel Vélez Serrano

Madrid, 9 de julio de 2020

A mi familia y amigos,

Abstract

Image recognition using deep learning through convolutional neural networks (CNNs) has dramatically improved and been increasingly applied to medical fields for diagnostic imaging. This project compares the performance of *feed-forward* neural networks and convolutional networks to identify metastatic tissue in histopathologic scans of lymph node sections. A great improvement is observed in the area under the curve ROC when using CNN.

Resumen

El reconocimiento de imágenes utilizando deep learning mediante las redes neuronales convolucionales (CNN) ha mejorado considerablemente su precisión y está aumentando su uso en el ámbito de la medicina para el diágnostico a través de imágenes. Este proyecto compara el rendimiento de redes neuronales *feed-forward* y redes neuronales convolucionales para identificar tejido metastásico en exploraciones histopatológicas de secciones de ganglios linfáticos. Se observa una gran mejora en el área bajo la curva ROC al utilizar CNN.

Índice

1. Introducción y objetivos

La inteligencia artificial se define como la simulación de la inteligencia humana procesada mediante máquinas. Esto incluye aprender, razonar y la autocorrección. Aunque este concepto no sea algo nuevo, sí que se ha convertido en una realidad que se aplica en el día a día en los últimos años, como para realizar labores repetitivas, el reconocimiento facial o del lenguaje y en diagnósticos médicos.

Las dificultades enfrentadas anteriormente, sugerían la necesidad de que las máquinas tuvieran la habilidad de aprender de forma autónoma. Esta habilidad se conoce como *machine learning*. Las máquinas gracias a datos o ejemplos de la vida real, consiguen extraer patrones, con los cuales toman mejores decisiones por sí mismas. Este aprendizaje depende en gran parte del tratamiento previo de los datos, cada componente o feature de ellos le ayudará a adquirirlo. La tecnología de aprendizaje automático potencia muchos aspectos de la sociedad actual: desde búsquedas en la web hasta filtrado de contenido en redes sociales y recomendaciones en sitios web de comercio electrónico.

En algunos casos, es difícil extraer información de valor de los datos. Por ejemplo, si se quiere detectar en una imagen la presencia de un coche mediante las componentes que la conforman como pueden ser sus ruedas, puede llegar a ser complicado describirla mediante sus píxeles. Las técnicas convencionales del aprendizaje automático están limitadas a la hora de procesar raw data (datos sin procesar). Esto conduce al aprendizaje representativo, donde el algoritmo aprende a procesar dichos datos y a abstraer sus componentes o características. Dentro de este contexto surge el *deep learning*, siendo un subconjunto del aprendizaje automático, en el que gozan de popularidad las redes neuronales convolucionales. Dichas redes son habituales en el tratamiento de información no estructurada (imágenes, texto, etc.) donde gracias a la sucesiva composición de módulos simples pero no lineales se consigue abstraer una representación de los datos más compleja. Para la clasificación en imágenes, las primeras capas tratan de abstraer características más sencillas, como pueden ser presencia de bordes y gracias a las sucesivas capas se puede llegar a la detección de objetos.

Añadir esta complejidad a los algoritmos se ha convertido en algo muy habitual en los últimos años debido a la capacidad de almacenamiento de datos gracias a los dispositivos conectados, al igual que gracias a los avances en velocidad computacional. Tiene aplicaciones en diversos campos de la ciencia e influye en el negocio y en la gestión gubernamental. Además de detección en imágenes y procesamiento del lenguaje, puede llegar a permitir el diseño de medicamentos específicos basandose en la codificación de ADN.

Este estudio trata de identificar a partir de imágenes a color la presencia de cáncer metastásico en exploraciones histopatológicas de secciones de ganglios linfáticos. Los datos vienen de una modificación del conjunto de datos de referencia PatchCamelyon disponible en Kaggle [1]. Esta plataforma permite a los usuarios encontrar y publicar conjuntos de datos, explorar y construir modelos en un entorno científico.

Se introducen los conceptos teóricos de las redes neuronales, incluyendo las convolucionales. Seguidamente, se aplicarán estos dos algoritmos a los datos, previamente preprocesados mediante el paquete Pytorch de Python y se compararán los resultados.

2. Estado del arte

Dado que este TFG utiliza técnicas de *deep learning*, daremos referencias para la clasificación de imágenes.

En las imágenes tomadas de forma natural, no es común que aparezca un único objeto. Esto lleva a pensar que puede no existir una etiqueta concreta para cada imagen, surgiendo así distintos conceptos para la clasificación de imágenes: la segmentación semántica y la de instancias [2]. Mediante la segmentación semántica se trata de etiquetar cada píxel de la imagen, en vez de la totalidad de ella. Se agrupan los píxeles para detectar objetos separados, obteniéndose tantas clases como entidades distintas presentes. Así, en una imagen donde están presentes coches y carretera, cada píxel pertenecerá a una de esas clases. Por otro lado, mediante la segmentación de instancias se identifican el número de instancias que hay en la imagen. Con diferencia al anterior, además de agrupar los píxeles por su tipo de entidad, se identifica cada entidad propia. Para el ejemplo anterior se tendrá una etiqueta distinta para cada coche presente. Una aplicación de esto puede ser contabilizar el número de objetos de la misma clase dentro de la imagen. Estos algoritmos tienen una gran aplicación a los problemas reales, donde se trata de identificar todo lo que está alrededor, como puede ser la conducción autónoma.

Aunque se almacenan datos de forma masiva en casi todos los ámbitos, en alguno de ellos como puede ser el de la salud, conseguir suficientes observaciones puede no ser posible. Una técnica que se puede utilizar en estos casos puede ser GANs (*generative adversial network*) [3], a partir de la cual se puede llegar a aumentar las muestras de patrones, creando nuevas. Se basa en enfrentar dos redes neuronales, una que crea imágenes nuevas y la otra las evalúa. Gracias a ello, se puede aumentar el tamaño de la base de datos y por ello, mejorar el rendimiento del algoritmo que se aplica.

" El *transfer learning* y la adaptación de dominio se refieren a la situación en la que lo aprendido en un entorno se explota para mejorar la generalización en otro entorno " [4]. Ya que mediante las sucesivas capas en las CNN se van detectando cada vez representaciones más complejas, con el *transfer learning* se utilizan las primeras de ellas ya entrenadas para otro problema. Se inicializa el algoritmo con ellas y se cambian las últimas para el proceso en concreto al que se quiere aplicar. Tras ésto, se entrena el algoritmo con las observaciones del problema a abordar, teniéndose así una inicialización mucho más acertada. Existen diversas arquitecturas preentrenadas como AlexNet, ResNet o Inception [5].

3. Conceptos previos

3.1. Conceptos de imágenes

Una imagen digital es una disposición de distintos valores en distintos lugares (píxeles). En el caso de ser en blanco y negro puede interpretarse como una matriz de tamaño $(m \times n)$ donde m indica el número de píxeles en el sentido horizontal, y n en el sentido vertical. Cada píxel indica el valor de la escala de grises al que corresponde, siendo el 0 el valor negro y 255 el blanco.

Para las imágenes a color, el sistema más común es el RGB (*red*, *green*, *blue*) como se observa en la figura 1. La representación de la imagen pasa a ser 3-dimensional, al introducirse el número de canales. Se tienen tres canales, uno por cada color primario, donde el valor de cada píxel en dicho canal indica el valor en su escala, también desde 0 hasta 255.

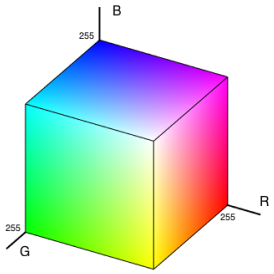

Figura 1: Estructura de imagen (RGB)

3.2. Conceptos matemáticos

3.2.1. Red neuronal *feed-forward*

Las redes neuronales son un tipo de modelos de aprendizaje automático donde se trata de encontrar unos pesos W para aproximar una función $y = f(x; W)$. El modelo más básico es el perceptrón multicapa (MLP) o red neuronal *feed-forward*. El término de *neuronal* se debe a que su estructura está inspirada en el funcionamiento de las redes neuronales biológicas y en este caso la información fluye únicamente hacia delante (*feed-forward*), sin que haya bucles ni interacciones hacia atrás.

Están formadas por una capa con las variables de entrada (input) y otra con las de salida (output), pudiendo existir capas intermedias que permiten combinar las variables de entrada para capturar relaciones de dependencia no lineal con las de salida. Estas capas intermedias se denominan capas ocultas. En el caso de no estar presentes ninguna, se puede interpretar como una regresión lineal (logística para el caso de clasificación).

Las capas son vectores formados por neuronas (denotamos por a_j^L a la neurona j en la capa L). Estas neuronas son resultado de una transformación afín entre unos pesos W_j y una constante b^{L-1}, seguido de una función de activación g que dará lugar a una función no lineal, teniéndose:

$$a_j^L = g(W_j^T a^{L-1} + b^{L-1}) \tag{1}$$

Por notación se suele tomar como $x_j = a_j^0$ y $b^L = a_0^L$

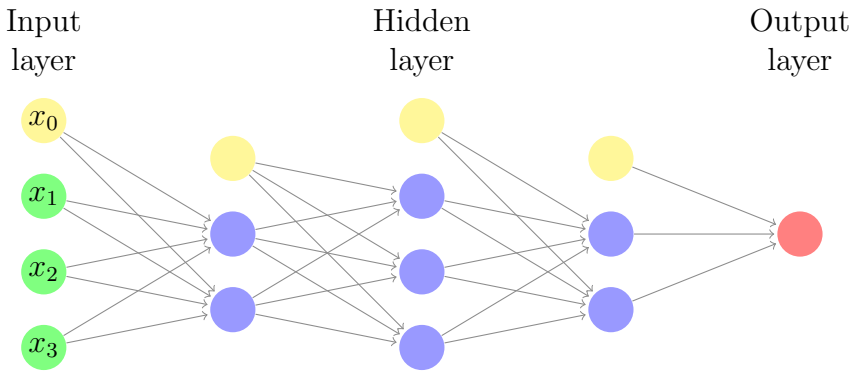

Figura 2: Red neuronal *feed-forward*

El output o salida puede tratarse de un resultado único, o en forma de vector, como por ejemplo para la clasificación multiclase (con más de dos categorías).

Estos modelos pertenecen al aprendizaje supervisado, al conocer de cada input el output correspondiente. Se trata de determinar los pesos que mejor establezcan la relación de dependencia entre el output y los inputs, minimizando el valor de la función error a definir. Se utilizan unos datos de entrenamiento y unos de validación con los cuales se elige la mejor estructura. Por último, se estima de forma independiente el rendimiento del modelo con los datos para el test. En la fase de entrenamiento, los pesos de la red se actualizan cada vez que se procesan un conjunto de observaciones, denominado como tamaño del *batch*. Se va calculando la media del error en cada iteración si dicho tamaño es mayor a uno. Una vez utilizados todos los datos, se habrá completado un *epoch* o iteración completa.

Se llama hiperparámetro a aquel que se elige antes de comenzar el proceso de aprendizaje, como por ejemplo el número de capas ocultas, su tamaño, al igual que el tamaño del *batch* y el número de *epochs*. Para elegirlos de forma óptima se comparan distintos modelos basándose en una métrica evaluada sobre el conjunto de validación, cada uno de ellos con distintos valores.

3.2.1.1. Estandarización y regularización

Dos conceptos importantes a tener en cuenta a la hora de entrenar una red neuronal *feed-forward* son la estandarización y la regularización.

Mediante la estandarización se trata de ajustar las distintas variables observadas de forma que estén en la misma escala. Este proceso, se puede utilizar tanto en el input, como en las sucesivas capas de la red. Una forma de estandarizar es mediante la mín-máx, estandarizando por el rango de la variable:

$$x'_i = \frac{x_i - min(x)}{max(x) - min(x)} \tag{2}$$

Otra forma puede ser restando la media y dividiendo por la desviación típica.

Por otro lado, la regularización se utiliza para prevenir el sobreajuste. Se busca que se tengan buenos resultados para nuevas observaciones, no solo para el conjunto de entrenamiento. Se puede prevenir el sobreajuste aumentando la cantidad de datos, tanto buscando nuevas observaciones como generando nuevos datos a partir de los que ya tenemos, pero no siempre es posible [4].

Otros métodos de regularización pueden ser la penalización de los parámetros y el *dropout* [4].

Para penalizar los parámetros añadimos a la función objetivo (J(W)) un término de penalización $\Omega(W)$. Con esto, la función a minimizar queda:

$$\tilde{J}(W) = J(W) + \alpha\Omega(W) \text{ donde } \alpha \in [0, \infty) \tag{3}$$

Se minimiza tanto el coste o error como una medida del tamaño de los parámetros W. Normalmente, solo se penalizan los parámetros de la transformación afín, dejando la constante sin regularizar. Algunos ejemplos son la regularización Lasso donde el término de penalización se trata de la media del valor absoluto de los coeficientes o la regularización Ridge donde se utiliza la media del cuadrado de ellos.

Por otro lado, la técnica *dropout* añade ruido a las capas ocultas. Se entrenan las subredes neuronales que pueden formarse eliminando algunas neuronas en cada capa de forma aleatoria e independiente en cada caso. Para ello se elige un hiperparámetro que indica la probabilidad de que la neurona esté presente. La ventaja de este método es que no tiene ningún coste computacional.

3.2.1.2. Arquitectura

Se tiene que elegir la arquitectura de la red, eligiéndose el número de capas ocultas y el número de neuronas. Estos valores son hiperparámetros a determinar, ya sea mediante métricas para comparar el rendimiento de cada uno de ellos o mediante algunos métodos. Existen resultados teóricos que demuestran que una red neuronal con sólo dos capas ocultas es suficiente para revolver cualquier problema de clasificación [6], pero debido al coste computacional que puede suponer, al igual que la gran variedad de problemas distintos a abordar, no existe un método generalizado y varía según la muestra de datos.

También se elige la función de activación en cada capa, algunos ejemplos son:

$$\begin{aligned} Sigmoid(x) =& \frac{1}{1+e^{-x}} \\ Relu(x) =& \begin{cases} 0 & \text{si } x \leq 0 \\ x & \text{si } x > 0 \end{cases} \\ LeakyRelu(x) =& \begin{cases} 0{,}01x & \text{si } x \leq 0 \\ x & \text{si } x > 0 \end{cases} \\ Tanh(x) =& \frac{sinh(x)}{cosh(x)} \end{aligned} \tag{4}$$

Si se utiliza la función Sigmoid para clasificaciones binarias en la capa de salida, al dar resultados entre 0 y 1, se puede interpretar como la probabilidad de pertenecer a una de las clases.

Tras esto, se inicializan los pesos. " Utilizar técnicas concretas para inicializar los pesos se puede considerar como una técnica de regularización al llegar a mínimos locales que generalizan mejor " [7]. Una inicialización habitual es la Xavier Uniform:

$$W \sim U(\frac{-\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}) \text{ donde } \mathrm{n}_j \text{ es el tamaño de la capa j.} \tag{5}$$

También se elige la función que se quiere minimizar. Para clasificaciones binarias es común utilizar la entropía cruzada binaria [4]:

$$J = \frac{-1}{n}\sum_{i=1}^{n}[y_i log(p_i) + (1-y_i) log(1-p_i)] \tag{6}$$

donde p_i es el output de nuestra red neuronal, y_i es la etiqueta conocida.

El error en cada neurona se calcula mediante la propagación hacia atrás de él, el error de la capa L con función de activación g es:

$$J^L = J^{L+1} \cdot W^L \cdot \frac{\partial g(x)}{\partial x} \tag{7}$$

Por último, se elige el método de optimización. La idea básica reside en el método del Mini-Batch Descenso del Gradiente, donde se actualizan los parámetros del siguiente modo:

$$w_{k+1} = w_k - \frac{\alpha}{m}\sum_{i=1}^{m}\nabla J_i(x) \tag{8}$$

donde α es la tasa de aprendizaje, $J_i(x)$ es el error en la observación i y m es el tamaño del *batch*. Dos casos concretos de este método son el Gradiente Descendente Estocástico, con m =1 y si m es el tamaño total del número de observaciones se llama Gradiente Descendente de Batch [4].

Es importante la elección de la tasa de aprendizaje, ya que si tiene un valor pequeño, aún siendo más preciso, puede causar un gran coste computacional, o incluso quedarse en un mínimo local. Y por el contrario, si se toma un valor grande, aún siendo el proceso de aprendizaje más rápido, pueden aparecer efectos oscilatorios en torno al mínimo y no encontrarse nunca una buena solución.

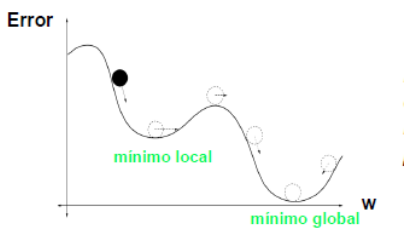

Figura 3: Tasa de aprendizaje

Una forma visual de interpretar este concepto, es mediante la representación del valor de la tasa de aprendizaje como el diámetro de una bola como se representa en la figura 3 . Se quiere elegir el valor suficientemente grande para recorrerlo rápido (sin gran coste computacional) pero sin saltarse recorrido (sin saltarse mínimos). Al ser un parámetro a determinar, se trata de otro hiperparámetro.

Los métodos más utilizados de optimización son los adaptativos, donde varían dicha tasa conforme van aumentando las iteraciones. El más utilizado es el Adam que deriva de adaptative moment estimation, y combina dos métodos AdaGrad y RMSProp [8]. Siendo T el número de iteraciones a realizar, se tiene:

Algorithm 1 Adam

Require: β_1, $\beta_2 \in [0,1]$, ϵ y α

$v_0 = 0, m_0 = 0$

while k $\leq$ T **do**

$g_k = \nabla J(x)$

$m_k = \beta_1 m_{k-1} + (1 - \beta_1) g_k$

$v_k = \beta_2 v_{k-1} + (1 - \beta_2) g_k \circ g_k$

$\widehat{m}_k = \frac{m_k}{1-\beta_1^t}$

$\widehat{v}_k = \frac{v_k}{1-\beta_2^t}$

$w = w - \alpha \frac{\widehat{m}_k}{\sqrt{\widehat{v}_k} + \epsilon}$

k = k + 1

end while

Existen diversas variaciones de estos algoritmos, donde se toman distintas maneras de variar la tasa de aprendizaje (representativa de la velocidad del aprendizaje) o introduciendo gradientes de segundo orden (representativos de un concepto de aceleración en el aprendizaje, término de " momentum ").

3.2.2. Red neuronal de convolución

3.2.2.1. Motivación

Al utilizarse las redes neuronales *feed-forward* para clasificar imágenes, se tienen que disponer todos los píxeles en un único vector, sin tener en cuenta la posición concreta de cada uno de ellos, tanto en el caso de ser en blanco y negro como a color (donde además se tendrán que disponer los distintos canales). Una forma de tenerlo en cuenta es utilizar la operación de convolución en cada capa, donde en vez de tener neuronas se tienen kernels o filtros, los cuales permiten extraer características de la imagen gracias a cada píxel en concreto así como los que le rodean.

La convolución proporciona tres aspectos importantes que ayudan a mejorar los modelos de aprendizaje automático: las interacciones sparse, el hecho de compartir parámetros y la equivarianza [4].

Las redes neuronales convolucionales tienen interacciones sparse al utilizarse kernels de menor tamaño que el input. Por lo tanto, se necesitan almacenar menos parámetros, lo cual reduce las necesidades de memoria y mejora la eficencia.

A diferencia de las redes neuronales *feed-forward* en las que se estima un peso distinto para cada arco de la red, en las de convolución existen pesos compartidos por varios arcos como se representa en la figura 4, donde también se representa la presencia de conexiones sparse.

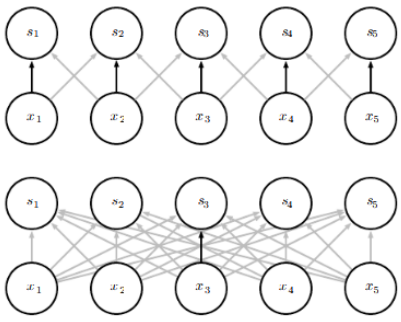

Figura 4: Pesos compartidos CNN

Las líneas resaltadas en negrita representan pesos con el mismo valor. Así, en la arquitectura superior (CNN), se utiliza un mismo valor para todos los inputs. En cambio, en una red neuronal *feed-forward*, representada por la arquitectura inferior, solo se utiliza en uno de ellos. Mediante las redes neuronales convolucionales, en vez de aprender parámetros separados para cada localización, se tienen en cuenta los píxeles próximos también. Además, estos valores se utilizan en sucesivas partes de la imagen, lo cual los generaliza aún más.

Por último, se tiene la equivarianza a la traslación. Esto equivale a que si el input varía, el output varía en el mismo sentido.

$$\text{f es equivariante a g si } f(g(x)) = g(f(x)) \tag{9}$$

En el caso de la convolución, se tiene esta propiedad con la traslación, obteniéndose el mismo resultado si se hace una traslación y después se opera, que si se hace en el orden inverso. La equivarianza no se da generalmente con otras transformaciones, como cambios de escala o la rotación. Es importante no confundir la equivarianza a la traslación con ser invariante. El ser invariante implica que se sigue detectando la clase si aplicamos una tralación al input. Es decir, detectamos un objeto independientemente de su posición en la imagen, lo cual no tiene por qué ocurrir.

3.2.2.2. Conceptos básicos

Dadas dos funciones f y g discretas se define convolución como:

$$(f * g)(t) = \sum_{x}^{\infty} f(x)g(t - x) \tag{10}$$

Llamaremos a la función desplazada kernel y lo denotaremos por K.
En el caso de ser bidimensionales se tiene:

$$S(i, j) = (X * K)(i, j) = \sum_{m=1}^{\infty}\sum_{n=1}^{\infty} X(m, n)K(i - m, j - n) \tag{11}$$

Además, esta operación es conmutativa, por lo que se puede expresar como:

$$S(i, j) = (X * K)(i, j) = \sum_{m=1}^{\infty}\sum_{n=1}^{\infty} X(i - m, j - n)K(m, n) \tag{12}$$

Si suponemos que estas funciones son cero fuera del conjunto de valores que se tienen, se trata de una suma finita. En la práctica se suele utilizar más la expresión (12) al tenerse menos valores posibles para m y n (al ser el kernel de menor tamaño que el input). Se obtiene la conmutatividad por haber " dado la vuelta " al kernel respecto al input. Si m crece, el índice del input crece pero el del kernel decrece. Aún siendo esta propiedad útil para muchas demostraciones teóricas, es más habitual utilizar la correlación cruzada:

$$S(i, j) = (X * K)(i, j) = \sum_{m=1}^{\infty}\sum_{n=1}^{\infty} X(i + m, j + n)K(m, n) \tag{13}$$

La operación es análoga a la anterior solo que no se le " da la vuelta " al kernel. En este contexto, se suele abusar de notación al llamarse a ambas operaciones de la misma manera. En el aprendizaje del algoritmo no influye, ya que aprenderá los pesos en la posición utilizada. Además, la conmutatividad no se mantiene al tenerse sucesivas operaciones tras ésta mencionada [4].

Los kernels (también llamados filtros) son 4-dimensionales, para el caso de imágenes a color, donde cada valor del kernel esta representado por $K_{i,j,o,k}$ siendo la última dimensión el número de filtros. Dados k filtros, de tamaño $(m \times n \times l)$, se tiene que el resultado de hacer la convolución es:

$$Z_{i,j,k} = \sum_{i}^{m} \sum_{j}^{n} \sum_{o}^{l} X_{i+m-1,j+n-1,o} K_{i,j,o,k} \tag{14}$$

3.2.2.3. Arquitectura

En las redes neuronales convolucionales se pueden distinguir tres capas distintas: las capas convolucionales, las capas *pooling* y las capas totalmente conectadas.

- Capa convolucional:

Tanto las capas anteriores como los kernels se suelen tomar cuadrados $(m \times m)$ para simplificar los cálculos. En estas capas se disminuyen las dos primeras dimensiones (longitud y altitud) a costa de aumentar la tercera (profundidad o número de canales). Tanto la capa con la que se va a convolucionar como el kernel deben tener el mismo número de canales, y dicha operación da lugar a un único canal. Los canales de la salida de estas capas vienen dados por el número de filtros que se aplican.

Una forma visual de definir la operación de convolución es superponer el filtro sobre el input, multiplicar elemento a elemento y sumar los resultados. Así, para un input de tamaño (4×4) y un filtro de tamaño (2×2) se tiene:

$$\begin{bmatrix} 3 & 1 & 1 & 3 \\ 1 & 2 & 0 & 4 \\ 3 & 2 & 4 & 0 \\ 1 & 1 & 3 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 4 \end{bmatrix} \qquad (4 = 3 \cdot 1 + 1 \cdot 0 + 1 \cdot 1 + 2 \cdot 0)$$

Para el siguiente paso se necesita definir el *stride*, que indica cómo el filtro convoluciona a lo largo del input, es decir, cómo lo va recorriendo. Así, con *stride* s = 1 el segundo paso sería:

$$\begin{array}{|cccc|}\hline 3 & 1 & 1 & 3 \\ 1 & 2 & 0 & 4 \\ 3 & 2 & 4 & 0 \\ 1 & 1 & 3 & 1 \\ \hline\end{array} \quad * \quad \begin{array}{|cc|}\hline 1 & 0 \\ 1 & 0 \\ \hline\end{array} \quad = \quad \begin{array}{|cc|}\hline 4 & 3 \\ \hline\end{array} \qquad \text{s} = 1.$$

En cambio, para s = 2 se saltan dos celdas, quedando:

$$\begin{array}{|cccc|}\hline 3 & 1 & 1 & 3 \\ 1 & 2 & 0 & 4 \\ 3 & 2 & 4 & 0 \\ 1 & 1 & 3 & 1 \\ \hline\end{array} \quad * \quad \begin{array}{|cc|}\hline 1 & 0 \\ 1 & 0 \\ \hline\end{array} \quad = \quad \begin{array}{|cc|}\hline 4 & 1 \\ \hline\end{array} \qquad \text{s} = 2.$$

Así se obtienen los sucesivos valores fijando un valor para el *stride*. Por último, se necesita definir el *padding*, que es el número de celdas con cero que se añaden al input, lo cual sirve para no perder información de la imagen. Se añaden de forma uniforme en el alto y ancho de la imagen. De esta forma, los dos *padding* más habituales son el *valid padding*, que no añade nada y el *same padding* que añade lo necesario para que el input y el output tengan el mismo tamaño.

Generalizando (14) para un valor de *stride* (s), un *padding* (p), un número de filtros (f) de tamaño $(n \times n \times c)$, la operación de convolución sobre un input de tamaño $(m \times m \times c)$ se puede expresar como:

$$Z_{i,j,k} = \sum_{i}^{m} \sum_{j}^{n} \sum_{o}^{c} X_{i+(m-1)\cdot s, j+(n-1)\cdot s, o} K_{i,j,o,k} \tag{15}$$

Debido a esta operación, es habitual tener un tamaño de salida distinto al de entrada. Denotando con el superíndice L la capa a la cuál pertenece dicha información, el valor de las primeras dos dimensiones del output vendrán determinadas por:

$$m^L = \frac{m^{L-1} - n^L + 2 \cdot p^L}{s^L} + 1 \tag{16}$$

Obteniéndose una salida de tamaño $(m^L \times m^L \times f^L)$.

De forma análoga a las redes neuronales *feed-forward*, tras esta operación se toma una función de activación (4).

Dado el input tomado en los ejemplos anteriores de tamaño (4×6), y un filtro de tamaño (2×2) con s $= 1$ y *valid-padding*, la salida es de tamaño:

$$(\frac{4-2+2\cdot 0}{1}+1) \times (\frac{4-2+2\cdot 0}{1}+1) = 3 \times 3$$

Por último, en el caso de tenerse varios canales, tanto el input como el filtro deben tener el mismo número de ellos, y por cada paso se obtiene un único valor (se suma el valor para cada canal). Así se obtiene como salida un único canal, luego su tamaño vendrá determinado de la misma manera que con la expresión (16). De forma visual, se puede interpretar que se superpone un " cubo " como se representa en la figura 5.

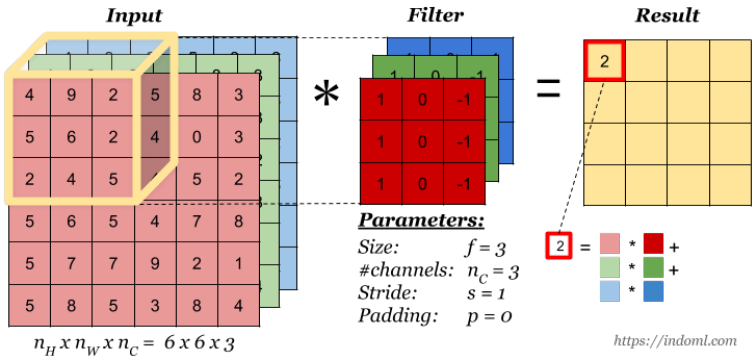

Figura 5: Convolución con 3 canales

- Capa de *pooling*:

Se utiliza después de cada capa de convolución y se reducen las dos primeras dimensiones sin aumentar la tercera (canales). Se define su tamaño, el *stride* y el *padding*. En este caso no se tiene el concepto de número de filtros (se puede interpretar como uno único) ni el número de canales (se aplica la operación a cada canal por separado). La operación es no lineal al calcularse la media en el caso de *average pooling* o el máximo con *max pooling*. Las dos primeras dimensiones se reducen de la misma manera (16) que en la capa convolucional. Así, un ejemplo de *maxpool* tamaño (2×2) con s $= 1$ es :

$$\text{Maxpool} \begin{bmatrix} 1 & 1 & 2 & 5 \\ 2 & 2 & 2 & 3 \\ 1 & 7 & 0 & 1 \\ 1 & 2 & 4 & 1 \end{bmatrix} = \begin{bmatrix} \text{máx}(1,1,2,2)=2 & \text{máx}(1,2,2,2)=2 & \text{máx}(2,5,2,3)=5 \\ \text{máx}(2,2,1,7)=7 & \text{máx}(2,2,7,0)=7 & \text{máx}(2,3,0,1)=3 \\ \text{máx}(1,7,1,2)=7 & \text{máx}(7,0,2,4)=7 & \text{máx}(0,1,4,1)=4 \end{bmatrix}$$

Se obtiene el máximo de los valores en cada canal, quedando el valor del número de canales tras aplicar esta capa igual (en este caso la tercera dimensión se mantiene constante, mientras que con la capa convolucional viene dada por el número de filtros).

- Capa totalmente conectada:

En las últimas capas, se suelen tener capas totalmente conectadas (capa *fully-connected*), que no son más que una red neuronal *feed-forward* donde cada neurona en una capa está conectada con cada una de las de la siguiente. Mediante las capas convolucionales se reducen las dos primeras dimensiones y se aumenta la tercera, y mediante la de *pooling* se reducen aún más las dos primeras dimensiones. Gracias a ello, el output de las sucesivas capas se va asemejando cada vez más a un vector. De esta forma, las capas totalmente conectadas no tendrán un número excesivo de parámetros a estimar.

Es habitual no llegar a reducir las dos primeras dimensiones a 1, luego para esta capa se disponen todos los valores en un vector unidimensional y se opera de forma análoga a una red neuronal *feed-forward* (se tiene un peso por cada neurona y finalmente una función de activación).

En la figura 6 se puede ver un ejemplo de una arquitectura de red neuronal de convolución completa.

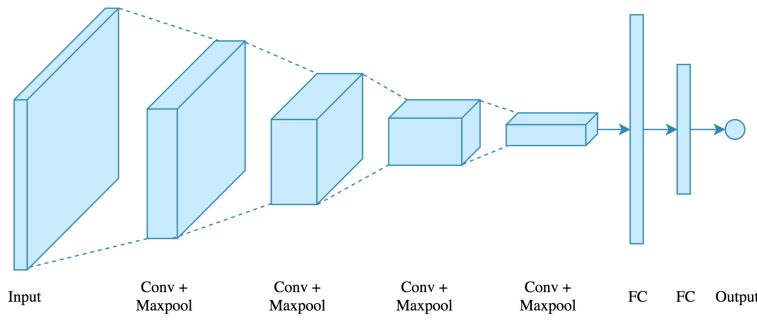

Figura 6: Arquitectura completa de red neuronal convolucional

3.2.3. Métricas de validación

3.2.3.1. Matriz de confusión

El output en un problema de clasificación binario puede tomarse como un valor continuo entre 0 y 1, interpretándose así como la probabilidad de pertenecer a una clase. Para clasificar una observación, se toma un umbral a partir del cual se clasificará como positivo. En caso contrario será negativo. Se pueden dar cuatro casos:

- VP : clasificado como positivo de forma correcta
- VN : clasificado como negativo de forma correcta
- FP : clasificado como positivo de forma incorrecta
- FN : clasificado como negativo de forma incorrecta

Gracias a estos valores, se puede construir la matriz de confusión representada en la figura 7:

	P	N
P	VP	FP
N	FN	VN

Figura 7: Matriz de confusión

Además se indica la especifidad (SPC), que indica el porcentaje de negativos bien clasificados y la sensibilidad (VPR), que indica el porcentaje de positivos bien clasificados:

$$\begin{aligned} SPC &= \frac{VN}{N} \qquad \text{N: total de negativos (VN + FP).} \\ VPR &= \frac{VP}{P} \qquad \text{P: total de positivos (VP + FN).} \end{aligned} \tag{17}$$

Por último, se define como precisión el porcentaje de observaciones bien clasificadas:

$$\text{Precisión} = \frac{VP + VN}{P + N} \tag{18}$$

Estos valores dependen del umbral escogido. Por tanto, se trata de un hiperparámetro a definir para cada modelo, el cual condicionará significativamente la precisión de él. Esto puede traer problemas a la hora de comparar varios modelos, al poderse tener un umbral óptimo distinto para cada uno.

Incluso teniéndose la misma precisión, si se tienen distintos valores de SPC y VPR, debido al número de FN y FP de cada modelo, puede llegar a ser difícil la elección del mejor de ellos. Esto lleva a la pregunta de cuál de los dos valores tiene más importancia para el problema a solucionar [9].

También se puede dar el caso en el que el objetivo no sea obtener la mejor precisión, sino minimizar algún tipo de error. Por ejemplo, en la detección de enfermedades, se suele preferir minimizar los falsos negativos aunque por ello aumenten los falsos positivos e incluso disminuya la precisión. Por tanto, en este caso no se quiere encontrar el modelo con mayor precisión, sino el que minimice los FP y FN, cada uno de ellos con un peso. Es decir, se quiere minimizar:

$$Coste = FP \cdot W_{fp} + FN \cdot W_{fn} \tag{19}$$

Esta formulación tampoco puede llevar a solucionar el problema ya que no es habitual conocer el peso exacto de cada error.

Otro problema que puede surgir con esta métrica es que debido a las observaciones de la base de datos, se puede dar el caso de que el mejor umbral para clasificar sea cerca del valor nulo, en el que todo se clasifica como positivo. Por ejemplo, si el 90 % de las observaciones son positivas, al maximizar la precisión nos podría dar dicha solución, lo cual no parece razonable.

Una forma de evitar estos posibles problemas es mediante las curvas ROC.

3.2.3.2. Curvas ROC

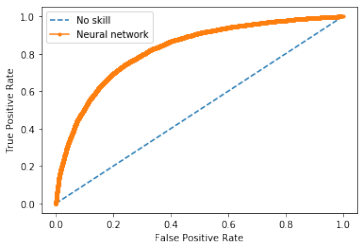

Figura 8: Curva ROC

La curva ROC es una técnica de visualización, organización y selección de clasificadores basados en su rendimiento. Para representar las curvas ROC se calcula la razón de falsos positivos (eje X), que es igual a 1 - SPC y la razón de verdaderos positivos (eje Y), que es VPR como se representa en la figura 8. Se calculan en función de todos los posibles valores del umbral. La recta y = x representa el rendimiento de un clasificador aleatorio.

En este caso, la métrica para comparar modelos es el área bajo la curva (AUC), donde vale 1 en el caso de una clasificación perfecta y 0.5 en el caso de ser aleatoria. Esta métrica nos proporciona una evaluación más general, donde el valor no está condicionado a la elección del umbral y por ello, se puede comparar de forma más clara los modelos.

El área bajo la curva mide la discriminación, es decir, la habilidad del modelo de clasificar correctamente las observaciones positivas como las negativas. Indica la probabilidad de que, escogidos al azar un caso real de tipo " 1 " y un caso real de tipo " 0 ", el modelo de mayor probabilidad al primer caso que al segundo. El modelo perfecto será aquél que deje área 1 bajo la curva ROC (máximo poder predictivo). Esto querría decir que siempre que coja un " 1 " real y un " 0 " real, el modelo da mayor probabilidad al primer caso que al segundo [10].

4. Caso de estudio

4.1. Dataset

Como se dijo en la introducción, se trata de identificar un cáncer metastásico en exploraciones histopatológicas de secciones de ganglios linfáticos. Cada imagen está anotada con una etiqueta binaria que indica la presencia de tejido metastásico.

La muestra de entrenamiento dispone de 220.024 imágenes y la del test de 57.457. Cada una de ellas consta de $(96 \times 96 \times 3)$ píxeles y están etiquetadas siendo positiva si en la región central de tamaño $(32 \times 32 \times 3)$ está presente al menos un píxel del tejido con tumor.

La figura 9 ilustra algunos ejemplos de las imágenes citadas:

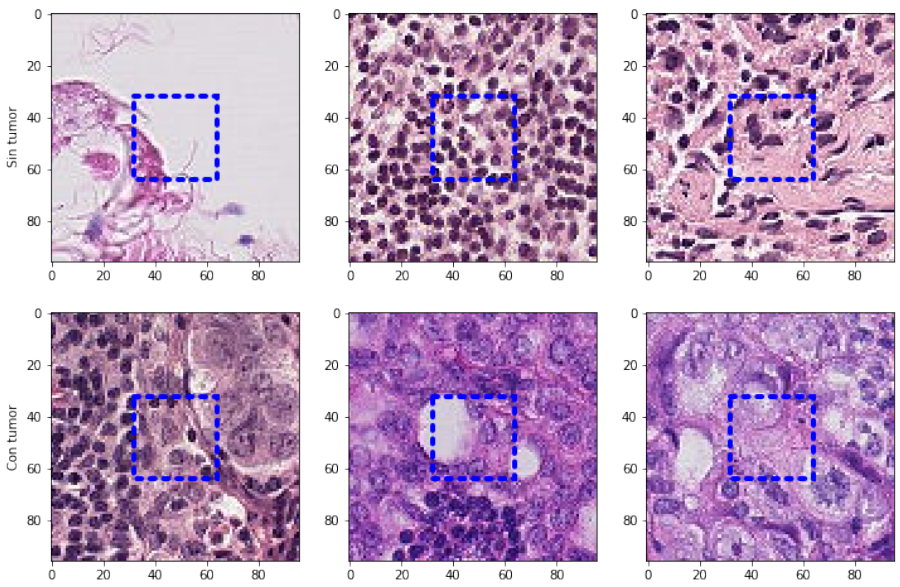

Figura 9: Exploraciones histopatológicas de secciones de ganglios linfáticos

El 40 % de observaciones están clasificadas como “ 1 ” y el 60 % como “ 0 ”.

En la figura 10 se representan los conjuntos tomados:

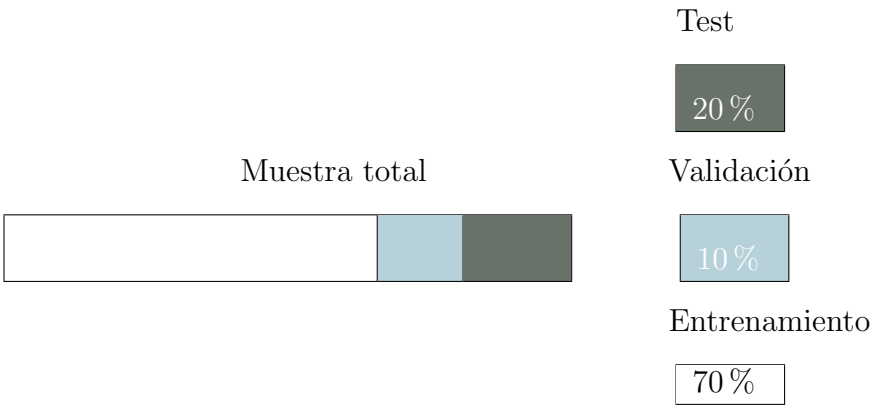

Figura 10: Conjunto de entrenamiento, validación y test

Se compararán los distintos modelos mediante el conjunto de validación y una vez elegido el que tenga mejor rendimiento, se utilizará el conjunto de test para tener una evaluación más real de la clasificación conseguida. El preprocesamiento, como los modelos y los resultados expuestos a continuación se pueden consultar en:

https://github.com/sofiavarib/cancerdetection_kaggle

4.2. Preprocesamiento

Primero se necesita realizar un procesamiento de los datos. Se intentan detectar posibles datos atípicos. Se busca si hay imágenes tanto muy oscuras como muy claras, en cuyo caso no tendrían suficiente información como para clasificarlas.

Se toma la media de los píxeles de los tres canales y gracias a un box-plot se buscan posibles candidatos a ser *outliers*. Los candidatos, en media, se acercan al píxel 0, lo que indica que se han detectado imágenes que son prácticamente negras. En la figura 11 se muestran algunas de ellas:

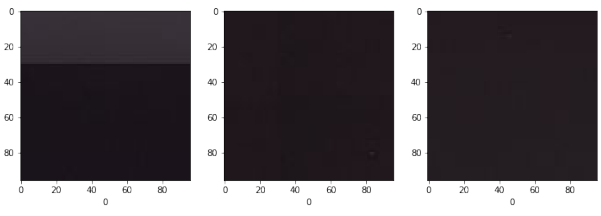

Figura 11: Posibles *outliers*

Aunque representan un porcentaje muy pequeño al ser unas de 200 imágenes (alrededor de un 0.001 % del conjunto de entrenamiento), se decide eliminarlas al no representar nada relacionado al caso estudiado.

4.3. Resultados

Los resultados se separan en dos grupos, el primero donde se realizan modelos mediante redes neuronales *feed-forward*, y el segundo donde se utilizan capas convolucionales. Todos los modelos se entrenan durante un tiempo similar para una comparación más sencilla (apróx. 24h).

- NN1: red neuronal *feed-forward* con una única capa oculta.
- NN2: red neuronal *feed-forward* con dos capas ocultas.
- CNN1: red neuronal de convolución con tres capas convolucionales cada una seguida de una capa de *maxpool* y finalmente una capa totalmente conectada.
- CNN2: red neuronal de convolución con cuatro capas convolucionales seguidas de capas tanto *averagepool* como *maxpool* y finalmente una capa totalmente conectada.

Todos tienen en común las siguientes características:

- Estandarización: mediante el rango de las variables (2), donde al ser el valor mínimo 0 y el valor máximo 255, cosiste en dividir el valor de cada píxel por 255.
- Inicialización: Xavier Uniform (5).
- Error: entropía cruzada binaria (6).
- Función de activación en última capa: Sigmoid (4).

A continuación se expondrán los resultados, así como una explicación con más detalle de ellos.

- **NN1**:

En los modelos NN1 y NN2 se toma únicamente la parte central de la imagen de tamaño 32·32·3 como input, ya que es en la región donde puede estar presente el tejido con cáncer.

Para este modelo se utiliza una única capa oculta con 350 neuronas. Se elige este número de capas de forma que no sea computacionalmente demasiado pesado, pero a su vez de algún resultado razonable. En ella se utiliza como función de activación Leaky Relu (4), además de estandarizar las variables tras esta capa. Se utiliza como algoritmo de optimización el Mini-Batch Descenso del Gradiente (8) con una tasa de aprendizaje de 0.0001 y el momento de 0.8.

A continuación se exponen los resultados obtenidos y la figura 12 representa su curva de aprendizaje:

Número de *epochs*	100
Número de capas ocultas	1
Número de parámetros	(32·32·3+1)·350 + 351
Tamaño del *batch*	174
Número de *batch*	1137
Error de entrenamiento final	0.4916
Error de validación final	0.5094
AUC de validación	0.821

Resultados NN1.

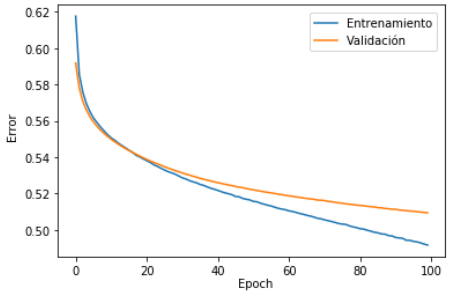

Figura 12: Aprendizaje NN1.

Se obtiene un AUC superior a 0.8 y la curva de aprendizaje muestra como tanto el error de validación como el de entrenamiento han ido bajando al aumentar el número de *epochs*.

- **NN2**:

Como se mencionó anteriormente, se toma únicamente la parte central de la imagen y se introduce otra capa oculta, la cual tiene 40 neuronas. La primera capa es exactamente igual que en el modelo anterior, en la segunda se utiliza como función de activación Leaky Relu (4). De esta forma, se introducen cerca de 14000 parámetros con respecto al modelo anterior. En este caso, el AUC de 0.823 siendo similar al obtenido con NN1. Se vuelve a utilizar el método de optimización Mini-Batch Descenso del Gradiente (8) con una tasa de aprendizaje de 0.0001 y el momento de 0.8 y un tamaño del *batch* de 174 (por lo tanto, siendo 1137 el número de *batch*).

La curva de aprendizaje está representada en la figura 13:

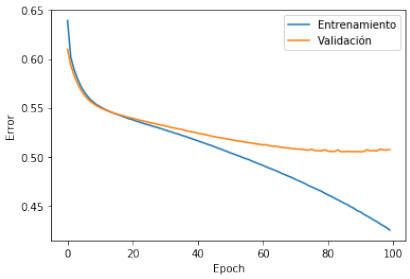

Figura 13: Aprendizaje NN2

Se observa prácticamente el mismo rendimiento que en el caso de NN1. Al añadir más capas, lo único que se reduce el error de entrenamiento (color azul), luego se podría llegar a estar sobreajustando. Se llega a la conclusión de que añadir una capa más no mejora la predicción, lo único que mejora es la eficiencia computacional al conseguir llegar a valores similares para el error en el conjunto de validación con menos *epochs*. Además, destaca que dicho error parece mantenerse constante a partir de cierto valor, luego tener más iteraciones sólo llevaría a sobreajustar los datos y por ello, empeorar el rendimiento real del modelo.

Los siguientes modelos serán mediante redes neuronales convolucionales, las cuales relacionan un conjunto de píxeles cercanos y comparten pesos que se utilizan en toda la imagen para encontrar patrones. Además, gracias a sus conexiones sparse se tiene menor coste computacional (en el sentido de la memoria).

- **CNN1**:

Para este modelo se toma la imagen al completo y consta de tres capas convolucionales, todas ellas seguidas de una función de activación y de una capa de *pool*, todas ellas con *valid padding*. Además, para los modelos de CNN se toma un tamaño del *batch* menor a los anteriores modelos, siendo el valor de éste 29. A continuación se explica con detalle las características de las capas:

Input	Nº de filtros	Tamaño del filtro	Activación	*Pool layer*	Output
(96,96,3)	15	(7,7,3)	Relu	Max(4,4), s = 2	(44,44, 15)

Primera capa.

Input	Nº de filtros	Tamaño del filtro	Activación	*Pool layer*	Output
(44,44,15)	20	(5,5,15)	Relu	Max(4,4), s = 2	(18,18,20)

Segunda capa.

Input	Nº de filtros	Tamaño del filtro	Activación	*Pool layer*	Output
(18,18,20)	30	(3,3,20)	Relu	Max(2,2), s = 2	(8,8,30)

Tercera capa.

Por último, se tiene una capa *fully-connected*.

Input	Nº neuronas	Activación	Output
(8,8,30)	1920	Sigmoid	1

Capa totalmente conectada.

En este caso, se utiliza como algoritmo de optimización el Mini-Batch Descenso del Gradiente (8), con tasa de aprendizaje 0.0001 y momento 0.8.

Al observar la figura 14, destaca una gran mejora al usar capas convolucionales. Con pocos *epochs*, se obtienen errores menores que los modelos anteriores. Se obtiene un AUC considerablemente mayor tras 30 *epochs*, siendo de 0.930. Al ser el error inferior y a su vez el AUC superior, se considera que este modelo tiene mejores resultados que los anteriores. Además, destaca que no se llega a una estabilidad aparente.

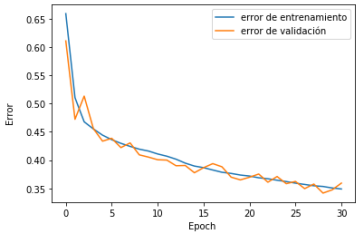

Figura 14: Aprendizaje CNN1

- **CNN2**:

Por último, se busca entrenar un modelo de redes neuronales convolucionales que con el mismo tiempo computacional que el modelo anterior, se puedan realizar más *epochs* y por tanto, se obtenga mejor rendimiento. Para ello se realizan los siguientes cambios:

- Algoritmo de optimización: Adam (1).
- Tasa de aprendizaje de 0.001, β_1= 0.9, β_2= 0.999 y ϵ =10^{-8}.
- *Dropout* con probabilidad 0.25 en las dos primeras capas.
- Se toma *stride* s = 2 en la primera y segunda capa convolucional, así como en la segunda capa de *pooling*.
- *Average pooling* en lugar de *max pooling* en algunas capas.

Además de estos cambios, se introduce una capa convolucional más, todas ellas con *valid padding*, se utiliza un tamaño del *batch* de 29 y también se utiliza la imagen al completo. Las características más concretas de cada capa son:

Input	Nº de filtros	Tamaño del filtro	Activación	*Pool layer*	Output
(96,96,3)	15	(6,6,3), s = 2	Relu	Avg(4,4), s = 1	(43,43, 15)

Primera capa.

Input	Nº de filtros	Tamaño del filtro	Activación	*Pool layer*	Output
(43,43,15)	20	(5,5,15), s = 2	Relu	Avg(4,4), s = 2	(9,9,20)

Segunda capa.

Input	Nº de filtros	Tamaño del filtro	Activación	*Pool layer*	Output
(9,9,20)	30	(3,3,20)	Relu	Max(2,2), s = 1	(5,5,30)

Tercera capa

Input	Nº de filtros	Tamaño del filtro	Activación	*Pool layer*	Output
(5,5,30)	40	(3,3,30)	Relu	Max(2,2), s = 1	(2,2,40)

Cuarta capa

Por último, se tiene una capa *fully-connected*.

Input	Nº neuronas	Activación	Output
(2,2,40)	160	Sigmoid	1

Última capa

Con aproximadamente el mismo tiempo computacional, se realizan 100 *epochs*. Se observa en la figura 15 una mayor inestabilidad en el error de validación (posiblemente debido al *dropout*) pero se obtiene menor error final tanto para el conjunto de validación como el de entrenamiento. Se obtiene un AUC de 0.948 superando a los modelos anteriores. Además se podría reducir aún más el error con sucesivos *epochs*, al no haberse llegado a una estabilidad aparente.

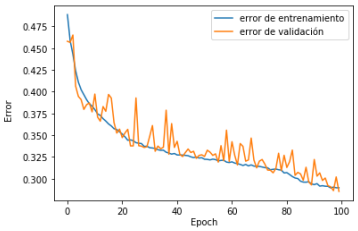

Figura 15: Aprendizaje CNN2

4.3.1. Discusión

Las características principales de los modelos vienen descritas en la siguiente tabla:

Características	**NN1**	**NN2**	**CNN1**	**CNN2**
Número de parámetros	1075901	1089631	17091	25586
Tamaño del *batch*	174	174	29	29
Número de *batch*	1137	1137	6822	6822
Epochs	100	100	30	100
Error de validación final	0.5094	0.5073	0.3592	0.2855
AUC de validación	0.821	0.823	0.930	0.948

Resultados de los modelos.

Se observa la gran diferencia de números de parámetros necesarios para las redes neuronales *feed-forward* y las convolucionales. Además de tener menos parámetros a aprender, se obtiene un error de validación considerablemente menor en las CNN comparado a las NN, al igual que una mayor área sobre la curva. Por lo tanto, las redes neuronales tienen mejor rendimiento con capas convolucionales que sin ellas.

Comparando las curvas de aprendizaje de los modelos, para las redes neuronales *feed-forward* se observa un valor para el error de validación aparentemente estable, donde lo único a mejorar es el error de entrenamiento, lo que podría conllevar a un sobreajuste si se añadieran *epochs*. En cambio, en las CNN no se llega a un punto estable tan claro con las iteraciones realizadas, por lo que se podría llegar a mejores resultados aumentándolas.

Por último, cabe destacar que para los modelos NN se toma solo la parte central de la imagen, donde está presente o no el cáncer. Por lo tanto, se toman menos valores tanto a la hora de entrenar como de evaluar. En cambio, en las CNN se toma la imagen completa. Si en las primeras se tomara completa, aumentaría considerablemente el coste computacional. Además, ya que en las redes neuronales *feed-forward* no se tiene en cuenta la localización de cada píxel, podría simplemente añadir ruido al modelo por lo que no se ha considerado como una posibilidad.

El mejor modelo comparando estos cuatros se trata del último (CNN2), donde se obtiene el mayor valor de AUC para el conjunto de validación. Además, esto se apoya en que se obtiene menor error de validación y entrenamiento. Por lo tanto, se utiliza este modelo para evaluarlo en el conjunto de test y así obtener una evaluación del modelo independiente.

Se obtiene un AUC de 0.9224 sobre el conjunto de test con el modelo CNN2, el cuál es un poco menor al AUC obtenido sobre el de validación. No lo es excesivamente, luego conduce a pensar de que no hay problema de sobreajuste. Se obtiene un modelo con muy buenos resultados a la hora de clasificar las imágenes, al estar considerablemente cerca de la clasificación perfecta, que sería con un AUC de valor 1.

5. Conclusiones

La detección de enfermedades mediante técnicas de *deep learning* puede llevar a grandes avances en el ámbito de la salud. La toma de decisiones apoyándose en resultados gracias a algoritmos, puede suponer una gran mejora de la detección precoz y tratamientos útiles en diversas enfermedades. La rapidez de toma de decisiones y la eficacia de las mismas puede conllevar a una gran mejora de la salud global.

El uso de algoritmos puede permitir a las empresas automatizar sus procesos reduciendo costes y en algunos casos incluso mejorando los resultados. La gran diferencia con el ámbito de salud, además de ser un tema donde el error puede llevar a problemas mucho más graves, es que, por el otro lado, el impacto positivo puede mejorar la situación de muchas personas. Por lo tanto, si se recolectan los datos de forma adecuada y controlada, y se toman decisiones basadas en diversos métodos, puede suponer un gran avance para la sociedad.

Este trabajo presenta cómo las redes neuronales convolucionales pueden ser un complemento a la hora de detectar cáncer en imágenes. Más aún en casos donde al ojo humano le hubiera sido difícil de escalar dichas imágenes, al tratarse de imágenes microscópicas (de secciones de ganglios en este caso). Se presentan las diferencias que supone aplicar redes neuronales *feed-forward* en comparación a redes neuronales convolucionales. Se aprecia una gran mejora con las segundas mencionadas, al obtenerse un gran rendimiento.

No solo se observa cómo se obtienen mejores resultados con las CNN, sino que además se obtienen con incluso menor coste computacional (al tenerse menos parámetros, se necesita menos memoria). Gracias a esta arquitectura se abstraen las características propias de las imágenes, obteniendose así una buena clasificación. Cabe destacar que el número de parámetros no supone una mejora de los resultados, lo cual puede llegar a ser una idea no intuitiva. Destaca la importancia de tomar una arquitectura conveniente para los datos, como son las redes neuronales convolucionales para el caso de imágenes. Teniéndola (CNN), se obtienen mejores resultados tanto en el sentido computacional como con la métrica utilizada.

Referencias

[1] Anthony Goldbloom. *Kaggle*. https://www.kaggle.com/c/histopathologic-cancer-detection. Accedido en 01/11/2019. 2019.

[2] Anna Khoreva y col. "Simple does it: Weakly supervised instance and semantic segmentation". En: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, págs. 876-885.

[3] Ian J. Goodfellow y col. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML].

[4] Ian Goodfellow, Yoshua Bengio y Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[5] Christian Szegedy y col. "Going deeper with convolutions". En: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, págs. 1-9.

[6] Richard Lippmann. "An introduction to computing with neural nets". En: *IEEE Assp magazine* 4.2 (1987), págs. 4-22.

[7] Xavier Glorot y Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". En: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010, págs. 249-256.

[8] Diederik P Kingma y Jimmy Ba. "Adam: A method for stochastic optimization". En: *arXiv preprint arXiv:1412.6980* (2014).

[9] Andrew P Bradley. "The use of the area under the ROC curve in the evaluation of machine learning algorithms". En: *Pattern recognition* 30.7 (1997), págs. 1145-1159.

[10] David G Kleinbaum y col. *Logistic regression*. Springer, 2002.