ETSIIT
Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Técnicas de Deep Learning para el diagnóstico de cáncer de mama

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:
Alejandro Jerónimo Fuentes
Tutor/es:
Rosa Rodríguez Sánchez

UNIVERSIDAD DE GRANADA

Noviembre 2020

Técnicas de Deep Learning para el diagnóstico de cáncer de mama

Autor
Alejandro Jerónimo Fuentes

Tutor/es
Rosa Rodríguez Sánchez
Ciencias de la computación e Inteligencia Artificial

Grado en Ingeniería Informática

UNIVERSIDAD
DE GRANADA

GRANADA, Noviembre 2020

Técnicas de *deep learning* para diagnóstico de cáncer de mama

Alejandro Jerónimo Fuentes

Palabras clave: *deep learning*, cáncer de mama, redes neuronales convolucionales, clasificación, segmentación de imágenes.

Resumen

En los últimos años, las técnicas de *deep learning* son cada vez más usadas para resolver problemas como el diagnóstico de cáncer. Uno de los tipos más frecuentes es el cáncer de mama, por lo que es necesario contar con métodos de diagnóstico temprano para evitar la propagación de la enfermedad. Anteriormente, el uso de técnicas clásicas como la extracción manual de características requerían de más tiempo y esfuerzo para obtener buenos resultados de clasificación. Sin embargo, gracias al *deep learning*, no es necesaria la extracción manual de características, consiguiendo resultados aun mejores. En este trabajo se realiza una introducción al *deep learning* y a la segmentación de imágenes, explicando las principales técnicas y arquitecturas de redes neuronales convolucionales. El objetivo del proyecto es el desarrollo de varios modelos de clasificación de imágenes histológicas y de segmentación de células cancerosas, aplicando las arquitecturas y mejorando el diagnóstico con técnicas clásicas. Con el uso de estos modelos se puede detectar rápidamente si existe cáncer en una imagen y localizar células cancerosas para un estudio posterior.

Deep learning techniques for breast cancer diagnosis

Alejandro Jerónimo Fuentes

Keywords: deep learning, breast cancer, convolutional neural networks, classification, image segmentation

Abstract

In recent years, deep learning techniques are becoming more used to resolve problems like cancer diagnosis. Breast cancer is one of the most commons forms of this disease, so we need a early diagnosis method to prevent propagations. In the past, old methods like hand-crafted features required more time and effort to get good classification results. However, thanks to deep learning, it is not neccesary to use hand-crafted features and results can be even better. In this project, we carry out an introduction to deep learning and image segmentation by looking at the main methods and convolutional neural networks architectures. The goal of this project is to develop different classification and segmentation models in a whole slide images and cancerous cell nuclei dataset. To do this, we need to use the architectures explained above, implementing them and improving hand-crafted features results. With these models, we can detect if a image contains cancer quickly and cancerous cells. The results can be used then for medical analysis.

Índice general

Índice de figuras

Índice de tablas

1 Introducción

En los últimos tiempos, las técnicas de *deep learning* para la resolución de problemas han ido cobrando cada vez más importancia. Hasta hace poco tiempo, problemas que resultaban casi imposibles de tratar por una máquina han conseguido resultados bastante aceptables gracias al *deep learning*. Algunos ejemplos de estos problemas son el reconocimiento de voz, traductores automáticos y el diagnóstico de enfermedades.

En el campo del diagnóstico de enfermedades, la inteligencia artificial es capaz de diagnosticar enfermedades tan bien como los médicos. Gracias al uso de los ordenadores, es posible decir si una persona tiene una determinada enfermedad o no en cuestión de segundos. Para ello, es necesario contar con bastantes datos de pacientes que hayan tenido la enfermedad. Los datos pueden ser imágenes médicas o texto. En el caso del cáncer, no solo se pretende que los resultados se obtengan en un tiempo concreto y de manera correcta sino también la detección de zonas del tejido que son difíciles de ver para el ojo humano.

El objetivo de las nuevas tecnologías consiste en mejorar las condiciones de nuestra vida cotidiana, haciendo posible ayudar a los demás. Este es el principal motivo por el que decidí hacer este trabajo, que consiste en aplicar las distintas técnicas de *deep learning* para el diagnóstico de cáncer de mama.

1.1 Estructura del proyecto

Antes de empezar a desarrollar el contenido del trabajo en profundidad, es conveniente explicar la estructura de capítulos que sigue:

Introducción: En el primer capítulo se hace una breve introducción al cáncer de mama en la actualidad y los tipos de cáncer que aparecen en los problemas a resolver. Se detallan los objetivos y la planificación del proyecto.

Estado del arte: Se realiza una introducción a la redes neuronales y redes neuronales convolucionales. Se verán los tipos de capas de una red neuronal convolucional, funciones de activación y las principales arquitecturas.

Experimentación: Se hará un análisis exploratorio y preprocesamiento de los datos que se van a utilizar. A continuación, se desarrollará un modelo de clasificación de IDC (carcinoma ductal invasivo) probando distintas arquitecturas.

Segmentación: En este capítulo se hace una breve introducción a la segmentación de imágenes, describiendo los tipos y las arquitecturas del estado del arte. A continuación, se desarrollará un modelo de segmentación semántica de células cancerosas de TNBC

(cáncer de mama triple negativo), realizando también el correspondiente análisis de los resultados obtenidos.

Conclusiones: En el último capítulo se extraen las conclusiones sobre el desarrollo del proyecto y se proponen trabajos futuros.

1.2 El cáncer de mama en la actualidad

El cáncer de mama es a día de hoy, uno de los tipos de cáncer más frecuentes del mundo. Si consultamos los datos de la Asociación española contra el cáncer (AECC) [1], vemos que en 2019 se diagnosticaron en España 33.307 casos nuevos. Además, el número de casos ha aumentado un 7,5 % entre 2012 y 2019.

Aunque en los últimos años la mortalidad del cáncer de mama ha bajado considerablemente, gracias al estudio de la enfermedad, los tratamientos existentes siguen siendo bastante agresivos y difíciles de afrontar para la persona y su entorno. Un diagnóstico de cáncer de mama supone para la persona un impacto en las relaciones personales, laborales y en su economía.

En las relaciones personales la enfermedad puede llevar a situaciones de estrés y preocupación tanto por parte de la persona como amigos y familiares. La persona puede tener temor ante la incertidumbre y los constantes cambios que produce la enfermedad. En cuanto a la situación laboral, es de esperar que baje el rendimiento de la persona y que se produzcan bajas temporales o permanentes, lo que supone un impacto en la economía familiar. Por último, a nivel económico, la enfermedad supone un incremento del gasto en la economía familiar. Esto se produce por los costes del tratamiento, desplazamientos hacia centros sanitarios o la alimentación específica de la persona. Estas consecuencias se agravan aún más si la enfermedad se diagnostica en fases avanzadas.

Por todo lo anterior, es necesario realizar un diagnóstico temprano para evitar tratamientos más agresivos y obtener mayores probabilidades de supervivencia. Por tanto, con el desarrollo de un modelo que muestre si una persona tiene cáncer o no, se produce una bajada importante del impacto que supone la enfermedad. Además, permite que el personal sanitario trabaje de una forma más eficiente y precisa, ya que el modelo puede mostrar aquellas regiones afectadas del tejido humano que son más difíciles de observar por un humano y que pueden pasar desapercibidas.

1.2.1 Carcinoma ductal invasivo (IDC)

El carcinoma ductal invasivo (IDC) es el tipo de cáncer de mama más frecuente. Se estima que alrededor del 80 % de los casos de cáncer de mama son IDC. Por tanto, sería necesario obtener un buen modelo de clasificación que ayude a diagnosticar este tipo de cáncer.

El IDC consiste en la propagación del cáncer desde los conductos lácteos hacia toda la mama. Es decir, es *invasivo* porque invade los conductos que producen leche, y es ductal,

debido a que se produce primero en estas zonas. Con el paso del tiempo, y si no se controla la enfermedad, el cáncer puede propagarse hacia otras partes del cuerpo, aumentando la probabilidad de mortalidad.

En cuanto a los métodos de diagnóstico más típicos, encontramos las mamografías, resonancias magnéticas o biopsias. Sin embargo, existen métodos aún más precisos como la microscopía virtual (*Whole slide imaging*).

La microscopía virtual realiza un escaneo completo de láminas histológicas, que contienen el tejido de la zona del cuerpo humano escaneada. Una vez digitalizadas las láminas, se pueden ver en un monitor de ordenador en alta resolución. De esta forma, se pueden ver aquellas zonas del tejido que se encuentran en mal estado de forma más precisa. Esta tecnología se utiliza principalmente para propósitos docentes, investigación o diagnóstico de enfermedades.

Entre las ventajas podemos destacar la alta precisión frente a métodos que requieren el uso de un microscopio óptico o la exploración física de los pacientes. En el caso del cáncer de mama, se puede determinar aquellas regiones del tejido más afectadas y establecer un tratamiento u otro según el grado de la enfermedad. Por otro lado, esta tecnología presenta un alto coste económico, debido al uso de los microscopios virtuales. Además, tenemos que tener en cuenta el peso de las imágenes en el almacenamiento del ordenador. Una imagen histológica completa puede llegar a ocupar decenas de gigas debido a la alta resolución. Por tanto, es necesario disponer de bases de datos suficientemente grandes para gestionar imágenes de gran tamaño. Otra opción sería ajustar el tamaño de las imágenes en el ordenador, pero se perdería información relevante.

1.2.2 Cáncer de mama triple negativo

El cáncer de mama triple negativo (TNBC) es uno de los tipos más agresivos y representa alrededor del 10 % al 15 % de todos los cánceres de mama. Se produce cuando no existen en las células receptores de estrógeno ni de progesterona; tampoco contiene la proteína HER2 (factor de crecimiento epidérmico humano). Cuando se dan estas tres condiciones, los médicos no pueden usar determinadas terapias para destruir las células, quedando solo la quimioterapia y la extracción del tumor (tumorectomía) o de la mama (mastectomía). A diferencia de otros tipos de cáncer de mama, el TNBC se propaga mucho más rápido y tiene peores pronósticos. También tiene más probabilidades de reaparecer después del tratamiento.

1.2.3 Calcificaciones

Otro de los métodos más usados para diagnosticar cáncer de mama son las mamografías. Actualmente, es el método más usado por su eficacia y su coste. Las calcificaciones son acumulaciones de calcio que se hallan en las mamografías. La mayoría de las calcificaciones tienen un origen benigno, aunque también existen casos de calcificaciones que, dependiendo de su agrupación, pueden tener un origen maligno. Existen dos tipos de calcificaciones: macrocalcificaciones y microcalcificaciones. En general, las calcificaciones más grandes suelen ser benignas (macrocalcificaciones); pero hay que prestar mucha atención a las microcalcificaciones, ya que suelen tener más probabilidad de tener origen maligno. De hecho, las microcalcificaciones son

la principal manifestación del carcinoma ductal in situ (CDIS), otro tipo de cáncer de mama. La mayoría de las microcalcificaciones no pueden ser detectadas a simple vista; por tanto, es muy importante detectarlas para poder ofrecer un tratamiento temprano.

1.2.4 Segmentación de células

La segmentación de células es una de las tareas más importantes en el análisis de imágenes biomédicas. Con esta tarea se pretende detectar la localización de las células en imágenes tomadas por un microscopio para su posterior análisis por expertos. A la hora de realizar la segmentación, además de detectar la localización, se podría detectar el tipo de célula y contar el número de células de cada tipo.

En el contexto del cáncer de mama triple negativo, el diagnóstico se lleva a cabo mediante el análisis de imágenes de una biopsia. En este caso, se puede utilizar la segmentación y el aprendizaje profundo para aprender a detectar células que presentan las tres condiciones necesarias. De esta forma, también se podría contar el número de células para comprobar la propagación del cáncer y aplicar el tratamiento adecuado con más urgencia.

1.3 Problemas a resolver

En este trabajo se desarrollará un modelo de clasificación de IDC a partir de un *dataset* de imágenes histológicas. El objetivo es encontrar el mejor modelo posible para hacer la clasificación, probando las arquitecturas de redes neuronales más usadas en la actualidad. A continuación, desarrollaremos un modelo de segmentación de células cancerosas de TNBC, utilizando una de las arquitecturas más eficaces para segmentación semántica de imágenes biomédicas.

Como resultado de la aplicación de los modelos de *deep learning*, podremos comprobar si una imagen histológica presenta cáncer y localizar las células cancerosas.

1.4 Objetivos

El objetivo general del proyecto consiste en el desarrollo de varios modelos de clasificación y segmentación de imágenes médicas, utilizando técnicas actuales de *deep learning*, para mejorar el diagnóstico de cáncer con técnicas clásicas.

Dicho objetivo se puede dividir en los siguientes objetivos específicos:

- Aprender los fundamentos de redes neuronales convolucionales, arquitecturas e hiperparámetros.
- Aprender las herramientas de *deep learning* y de procesamiento de datos más utilizadas en el mundo laboral.
- Realizar un análisis exploratorio y preprocesamiento de varios conjuntos de datos de imágenes médicas.

- Estudiar y analizar arquitecturas de redes neuronales convolucionales para la resolución del problema.

- Aplicar transferencia de aprendizaje en las arquitecturas para obtener mejores resultados.

- Aplicar segmentación semántica de células cancerosas en un conjunto de imágenes histológicas de cáncer de mama triple negativo, utilizando la arquitectura U-Net.

- Aplicar segmentación semántica en un conjunto de datos de mamografías, localizando calcificaciones, con el uso de la arquitectura *U-Net*.

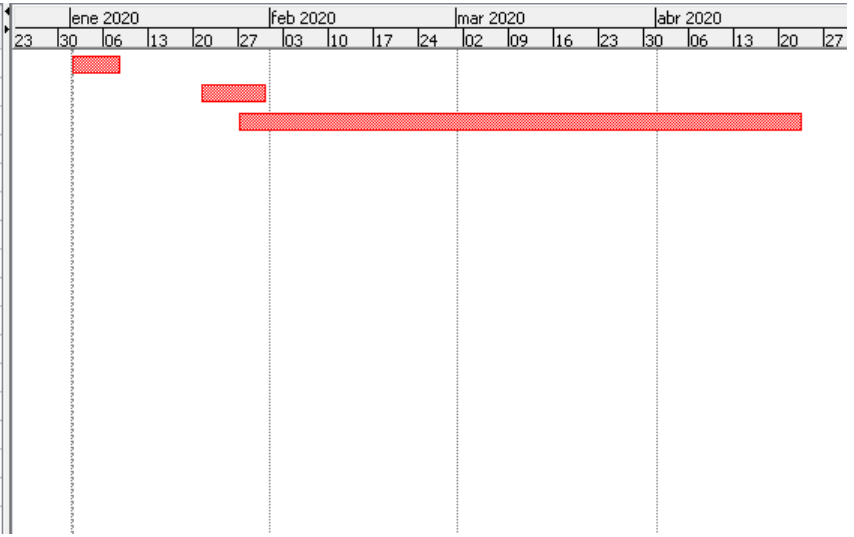
1.5 Planificación

En la **Figura 1.1** se detalla la planificación realizada del proyecto. En primer lugar, se realiza la fase de documentación en los primeros meses. Esta fase se dedica a realizar un estudio de los conceptos de visión por computador y de *deep learning* necesarios para el desarrollo del proyecto. La fase de documentación está presente durante todo el desarrollo, ya que de vez en cuando es necesario consultar la literatura. A continuación, se producen las fases de análisis y diseño; donde se realiza el diseño y la organización del software, realizando también el preprocesamiento de los datos y los *scripts* necesarios para la fase de implementación. En la fase de implementación, situada en los meses de agosto y septiembre, se estudian las librerías de desarrollo y se implementa el código para entrenar cada modelo. Al mismo tiempo se realiza la fase de verificación, corrigiendo errores de programación y ajustando los hiperparámetros para mejorar los resultados del modelo. Se produce, a continuación, el análisis de los resultados y su documentación en la memoria. Finalmente, en el mes de octubre, se realiza segmentación semántica en el *dataset* de mamografías y se extraen las conclusiones y posibles trabajos futuros del proyecto. En cuanto a la elaboración de esta memoria, se ha ido escribiendo a la vez que se realizaban las fases anteriormente expuestas.

1.6 Presupuesto

Con respecto a los costes de personal, el número medio de horas diarias estimadas es 5. El tiempo de desarrollo ha sido de 5 meses (20 días al mes). Si fijamos el precio por hora a 10 €, el coste total de personal serían 5000 €, aproximadamente. En la **Tabla 1.1** podemos ver los costes de *hardware* necesarios para realizar el entrenamiento de los modelos en un equipo local. Por tanto, los costes totales del proyecto ascenderían a 5835,88 €.

	Nombre	Duracion	Inicio
1	Planteamiento de objetivos	6 days?	1/01/20 8:00
2	Estudio de conceptos básicos de visión por computador	8,25 days?	21/01/20 8:00
3	Estudio de NN y CNN	64 days?	27/01/20 8:00
4	Memoria: Introducción	4 days?	15/07/20 7:00
5	Análisis exploratorio de los datos	2 days?	29/07/20 7:00
6	Memoria: Estado del arte	7 days?	18/07/20 7:00
7	Preprocesamiento de los datos	2 days?	30/07/20 7:00
8	Memoria: Análisis exploratorio de los datos	2 days?	30/07/20 7:00
9	Experimentación de arquitecturas	30 days?	3/08/20 7:00
10	Aprendizaje de la librería Keras	5 days?	3/08/20 7:00
11	Scripts para preprocesamiento y carga de datos	2 days?	30/07/20 7:00
12	Memoria: Análisis de los resultados	13 days?	14/09/20 7:00
13	Segmentación semántica U-Net	17 days?	1/10/20 7:00
14	Memoria: segmentación semántica U-Net	17 days?	1/10/20 7:00
15	Procesamiento del dataset	3 days?	1/10/20 7:00
16	Implementación U-Net	15 days?	5/10/20 7:00
17	Revisión y conclusiones	9 days?	25/10/20 7:00



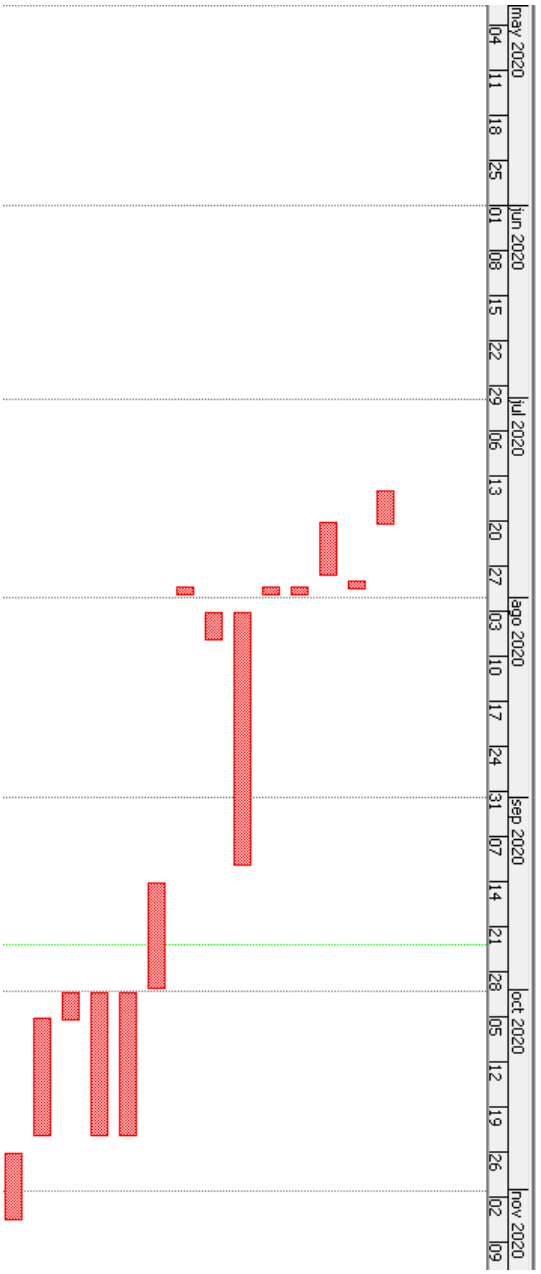


Figura 1.1: Diagrama de Gantt de la planificación del proyecto

Componente	**Modelo**	**Precio (€)**
CPU	Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz	395 €
GPU	NVIDIA GeForce GTX 1650 OC 4GB	149,89 €
RAM	Crucial DDR4 2400 PC4-19200 16GB 2x8GB CL17	152,99 €
SSD	Samsung 860 EVO Basic SSD 1TB SATA3	138 €
Total		835,88 €

Tabla 1.1: Costes de *hardware*

2 Estado del arte

2.1 Introducción al aprendizaje profundo

El *deep learning* o aprendizaje profundo, está formado por un subconjunto de técnicas de aprendizaje automático. Las técnicas de aprendizaje automático, a su vez, son otro subconjunto de la inteligencia artificial.

Durante los primeros años de la IA, se desarrollaron los primeros sistemas expertos que intentaban simular el conocimiento de un experto sobre un problema concreto. Los sistemas expertos necesitaban una aportación manual del conocimiento del experto para realizar los razonamientos. Más tarde, las técnicas de aprendizaje automático utilizaban conjuntos de datos para crear un modelo del problema, donde el experto seleccionaba las características de los datos más significativas para mejorar el aprendizaje. A continuación, se desarrolló el aprendizaje de representaciones. En el aprendizaje de representaciones, las características se descubren automáticamente a partir de los datos. Por último, se desarrollaron la técnicas de *deep learning*. Las técnicas de *deep learning* son capaces de aprender las características directamente de los datos, y además, pueden aprender nuevas características más complejas en un nivel más alto de abstracción. Al aprender características más complejas, podremos crear mejores modelos para resolver el problema propuesto.

La base del *deep learning* está formada por las redes de neuronas artificiales. Las redes neuronales surgen en la década de los 40 sin mucho éxito. Actualmente, las redes neuronales viven una época dorada gracias a las grandes cantidades de datos que generamos (*big data*) y el desarrollo de un hardware cada vez más potente. Con el uso de las redes neuronales, podremos resolver problemas que son muy fáciles para los seres humanos, pero muy complejos para un ordenador. En las siguientes secciones, veremos los conceptos básicos de las redes neuronales y se explicarán las principales arquitecturas de redes convolucionales.

2.2 Redes neuronales

2.2.1 Definición

Una red neuronal artificial es un modelo matemático que nos permite resolver problemas de clasificación con bastante éxito. Para aprender a clasificar, las redes neuronales están inspiradas en la biología del cerebro humano.

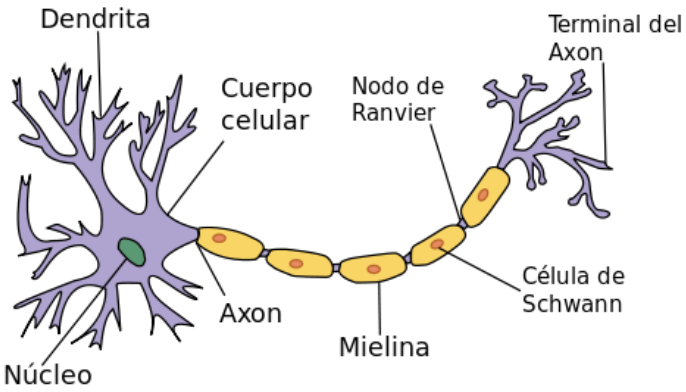


Figura 2.1: Partes de una neurona. Fuente: Wikipedia Commons.

En el cerebro humano, cada neurona recibe señales eléctricas a través de las dendritas. Si la señal es lo suficientemente fuerte, la neurona se activa y transmite la señal por el axón, llegando a las dendritas de otras neuronas conectadas.

Si tomamos como referencia el cerebro humano, podemos crear un modelo abstracto para que pueda ser entendido por una máquina.

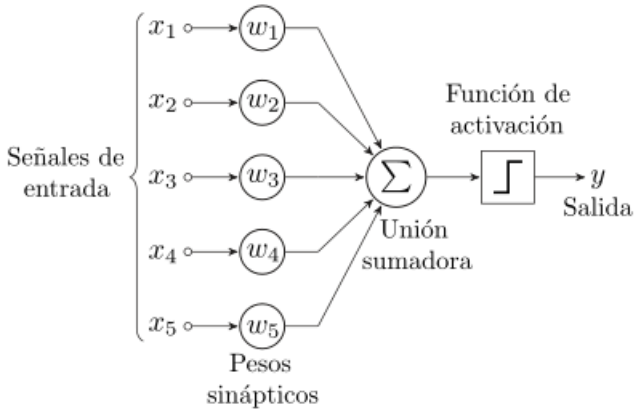


Figura 2.2: Perceptrón con cinco señales de entrada. Fuente: Wikipedia.

El modelo más simple es el perceptrón. El perceptrón está formado por una sola neurona en una capa. En la imagen anterior, tenemos cinco entradas $x_1, x_2, ..., x_5$ con sus respectivos pesos sinápticos $w_1, w_2, ..., w5$. En el contexto de este trabajo, las entradas serán las intensidades de los píxeles de la imagen. En muchas ocasiones, es frecuente utilizar un sesgo o *bias* w_0, que consiste en marcar una entrada con un valor fijo, en general $x_0 = 1$. El sesgo determina el umbral de activación de la neurona. Si tenemos un sesgo alto, entonces se supera el umbral y la neurona se activa.

A continuación, se realiza la suma ponderada de dichos pesos con las entradas $\sum_{i=1}^{n} w_i x_i$ y el resultado se pasa como entrada a la función de activación $f(\sum_{i=1}^{n} w_i x_i)$. La función de activación será la responsable de la salida de la neurona, ya que aplica una transformación

no lineal a la suma ponderada. Si aplicamos únicamente elementos lineales, la red aprenderá una función lineal a partir de las entradas. La mayor parte de los problemas reales, y también en el caso del diagnóstico, son no lineales. Por tanto, necesitamos aplicar este tipo de transformaciones para representar la no linealidad de las imágenes de entrada y realizar la clasificación correcta.

Finalmente, la función de activación devuelve una salida discreta, si el problema es de clasificación, o continua, si es de regresión. En el caso del diagnóstico de enfermedades, la salida será discreta y binaria, indicando si la persona tiene cáncer o no.

Para que la red aprenda, es necesario ajustar los pesos hasta que la red sea capaz mostrar la salida deseada. Para controlar esto, se utiliza una función de pérdida que calcula el error de la salida, midiendo las diferencias entre el valor de la salida y el de la entrada. El objetivo consiste en minimizar la función para encontrar los pesos que hagan que el error sea mínimo. Para ello, se utiliza el algoritmo de propagación hacia atrás (*backpropagation*), que se encarga de optimizar la función de pérdida.

2.2.2 Funciones de activación

Anteriormente hemos visto la necesidad de utilizar una función de activación para representar la no linealidad de los datos de entrada. Existen muchas funciones de activación con resultados distintos en la práctica. Las más relevantes para este trabajo son las siguientes:

Función sigmoide

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

La función sigmoide es una de las funciones más importantes y usadas en *deep learning*. La función escala los valores introducidos a valores entre 0 y 1. La función no está centrada en cero y está acotada entre 0 y 1. Su principal problema es la saturación del gradiente y su lenta convergencia. Debido a estos problemas, el uso de la función sigmoide es cada vez menor, prefiriendo el uso de alternativas como la función ReLU.

Tangente hiperbólica

$$tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

El rango de valores de la función tangente se encuentra entre -1 y 1. Está centrada en cero y la función usada por Cruz-Roa et al. [2] en su propuesta. Uno de los principales problemas que tiene esta función es la saturación del gradiente, haciendo el proceso de aprendizaje bastante lento.

Función de activación lineal rectificada (ReLU)

$$f_{relu} = max(0, z)$$

La función ReLU es una de las funciones más utilizadas en redes neuronales convolucionales debido a sus buenos resultados en la práctica. ReLU solo tiene en cuenta los valores positivos. Si tenemos valores negativos la neurona no se activa, siendo este su principal problema. Al no saturar las neuronas, el aprendizaje de la red es muy rápido y es adecuado para el tratamiento de imágenes. En el capítulo 3, utilizaremos esta función para tratar de mejorar los resultados de la propuesta de Cruz-Roa et al. [2].

Leaky ReLU

$$f(z) = \begin{cases} z & \text{si} \quad z \geq 0 \\ \alpha \times x & \text{en otro caso} \end{cases}$$

Variante de la función ReLU que intenta resolver el problema de la muerte de neuronas, multiplicando la entrada de la función por un coeficiente α en los casos negativos.

Función de unidades lineales exponenciales (ELU)

$$f(z) = \begin{cases} z & \text{si} \quad z \geq 0 \\ \alpha \times (exp(z) - 1) & \text{en otro caso} \end{cases}$$

ELU es una de las funciones de activación más recientes en el campo de *deep learning*. El parámetro α es constante, a diferencia de otras variantes de ReLU, donde se aprende. Según Cruz-Roa et al. [2], no solo realiza el aprendizaje más rápido, sino que mejora los resultados de clasificación.

Función softmax

$$y_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

donde y_j será la probabilidad de pertenecer a la clase j. z_j es la entrada neta de la neurona *softmax*, esto es, la suma ponderada de pesos que vimos anteriormente $\sum_i w_{ij} x_i$. K es el número de clases que tiene nuestro problema, también será el número de neuronas de la capa de salida.

La función *softmax* devuelve la probabilidad de pertenecer a una determinada clase y la suma de las probabilidades es 1. Por tanto, será especialmente útil para trabajar en problemas de clasificación. Tendremos tantas salidas como clases haya en el problema . La predicción de la red será la clase que tenga la probabilidad más alta. *Softmax* se usa en la capa de salida de la red. En el diagnóstico de cáncer, al ser un problema binario, usaremos dos neuronas de tipo *softmax*.

2.2.3 Función de error: Entropía Cruzada

En todos los experimentos de este trabajo se utiliza la función de entropía cruzada como función de error. Dicha función se utiliza junto a la función *softmax*. La función de error sirve para ajustar los pesos de la red durante el entrenamiento. Dependiendo de su valor, el ajuste será mayor o menor. Podemos definir la función de entropía cruzada como:

$$H(p,q) = -\sum_{x} p(x) \log q(x)$$

donde $p(x)$ es la probabilidad deseada de clasificación y $q(x)$ la probabilidad obtenida por la función *softmax*. La función penaliza bastante las probabilidades muy alejadas de la probabilidad deseada. Por ejemplo, si en la instancia actual de entrenamiento la imagen tiene cáncer, $p(x) = 1$ en la neurona *softmax* que muestra la probabilidad de cáncer, y $p(x) = 0$ en el caso contrario. Por tanto, si $p(x) = 1$ y $q(x)$ tiene un valor muy bajo, el error será muy alto, por lo que habrá que realizar un ajuste muy grande durante el entrenamiento.

2.3 Redes neuronales convolucionales

2.3.1 Definición y ventajas

En *deep learning* se utilizan redes neuronales profundas, es decir, redes con dos o más capas de neuronas. Las capas de la red están conectadas entre sí, donde cada neurona de la capa c estará conectada a todas las neuronas de la siguiente capa $c + 1$. Este tipo de redes se denominan redes neuronales *feed-forward*.

El problema de utilizar redes neuronales *feed-forward* con imágenes reside en el número de pesos que hay que ajustar. Si tenemos imágenes muy grandes, tendríamos que ajustar redes con cientos de millones de pesos, intratable para un ordenador. Por este motivo, utilizaremos las redes neuronales convolucionales (CNN).

CNN es una de las técnicas más populares en *deep learning* debido a su buen rendimiento con imágenes. Gracias a las CNN, la red aprende los patrones presentes en la imagen sin necesidad de una extracción manual de características. En las primeras capas, la red aprenderá a detectar los bordes de la imagen, y a lo largo de las capas, objetos y escenas. Con los patrones de la imagen aprendidos, la red podrá decidir cuando una imagen tiene IDC o no.

Dentro de la CNN, existen varios tipos de capas con hiperparámetros que, dependiendo de su valor, el modelo ofrecerá mejores o peores resultados.

2.3.2 Tipos de capas

2.3.2.1 Capa convolucional

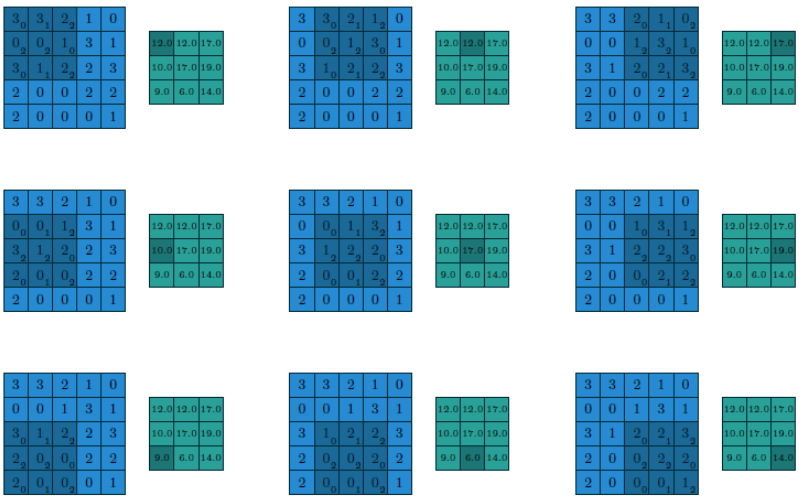


Figura 2.3: Convolución de una matriz 5×5 con un Kernel 3×3. Fuente: Dumoulin and Visin [3]

La capa convolucional es el componente clave de una red convolucional. La entrada de la capa será una matriz bidimensional, correspondientes a las intensidades de los píxeles de la imagen. A continuación, se realiza la operación de convolución con una máscara llamada *kernel*. El *kernel* representa el filtro de características que queremos aprender, es decir, los pesos que vamos a ajustar.

En una capa convolucional, las neuronas estarán conectadas a distintas subregiones de la imagen de entrada. De esta forma, aprendemos las características locales de esas subregiones. Otra característica fundamental es el uso de pesos compartidos por varias neuronas; de esta forma, cada neurona se beneficia de las características encontradas anteriormente y se reduce el número de pesos a entrenar. También se suelen usar varios filtros para aprender distintas características. La salida de la capa convolucional será un conjunto de imágenes (mapas de rasgos), resultado de las convoluciones con los filtros. Si aplicamos K filtros distintos, tendremos K imágenes distintas.

En la **Figura 2.3** vemos un ejemplo sencillo de convolución. Para realizar la convolución, multiplicamos la región sombreada de la matriz azul con el kernel, multiplicando cada elemento con el elemento correspondiente en el kernel. Si sumamos todos los resultados de las multiplicaciones obtendremos el resultado de la convolución, en el primer elemento de la matriz de color verde, que sería 12.

Los hiperparámetros que usaremos en las capas convolucionales son los siguientes:

Profundidad: El primer hiperparámetro a considerar es la profundidad. La profundidad se define como el número de filtros que vamos a entrenar. Si vamos a aprender K filtros, tendremos un mapa de rasgos con profundidad K.

Conectividad Local: Para reducir el número de pesos, las neuronas se conectan a subregiones de la imagen de entrada. El tamaño de cada subregión es otro hiperparámetro F, igual al tamaño del filtro.

Paso: El paso S indica el número de convoluciones que se van a aplicar en la imagen. Si tenemos $S = 1$, realizamos todas las convoluciones posibles mientras que si tenemos $S = 2$, nos saltamos una de cada dos muestras. El paso se utiliza principalmente para reducir el coste de la operación de convolución y para obtener mapas de rasgos más pequeños.

Rellenado con ceros: Cuando se realiza una convolución, el tamaño de la imagen de salida es inferior al de la imagen de entrada. Para evitar esto, se utiliza el rellenado con ceros (*zero-padding*), rellenando los bordes de la imagen vertical y horizontalmente.

2.3.2.2 Capa de agregación

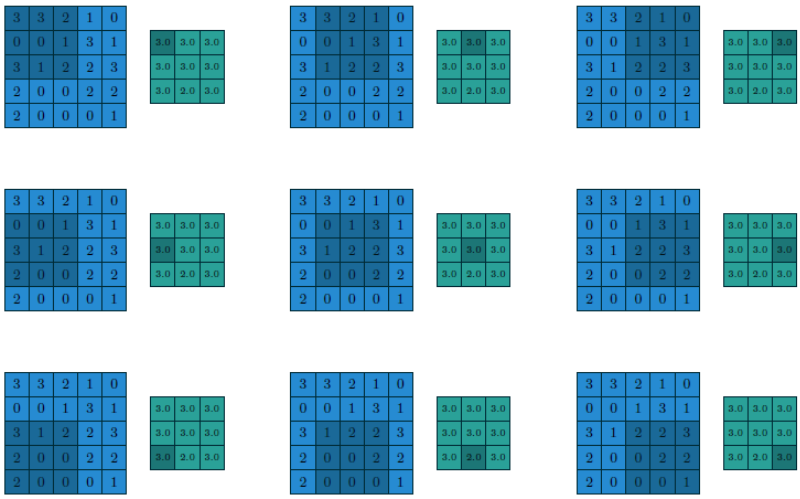


Figura 2.4: *Max Pooling* de una matriz 5×5 con un Kernel 3×3. Fuente: Dumoulin and Visin [3]

En una capa de agregación o *pooling* se reduce aún más el tamaño de la imagen de entrada. Las capas de agregación se sitúan después de las capas convolucionales, por lo que la entrada es el conjunto de mapas de rasgos. Este tipo de capas se utiliza para reducir el tamaño de los datos de las imágenes y para obtener invarianza frente a cambios en las imágenes. Existen dos técnicas para hacer la agregación:

Max-pooling: Para realizar el max-pooling usamos un nuevo filtro de tamaño F sobre la imagen de entrada. A continuación, se aplica el filtro sobre una subregión de la imagen de entrada y se selecciona el valor máximo de la subregión definida por el filtro.

Average-pooling: La técnica es similar a la anterior. En este caso se realiza el promedio de los valores de la subregión definida por el filtro.

Existen dos hiperparámetros destacables en las capas de agregación:

- **Ventana:** Es el tamaño del filtro F que se usa para obtener la salida de la capa.
- **Paso:** Igual que en las capas convolucionales, indica el número de agregaciones que se van a realizar.

Es importante controlar correctamente estos hiperparámetros, ya que podríamos perder demasiada información del problema con la reducción de las imágenes.

2.3.2.3 Normalización por lotes

Para acelerar el proceso de entrenamiento, usaremos la normalización por lotes. En esta capa se normalizan las imágenes de entrada sobre un minilote. Por cada minilote, se calcula primero la media μ y la varianza σ^2:

$$\mu = \frac{1}{m}\sum_{i=1}^{m} h_i$$

$$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(h_i - \mu)^2$$

A continuación, se realiza la normalización. Si consideramos x como los niveles de activación del minilote, podemos calcular la normalización x_{bn} como:

$$x_{bn} = \frac{h - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

donde ϵ es un número muy pequeño que se utiliza para evitar las divisiones por cero.

En la práctica, se suele ubicar esta capa antes de la capa de la función de activación no lineal.

2.3.2.4 Capa completamente conectada

Finalmente, se suele aplicar una capa completamente conectada antes de la capa de salida *softmax*. En una capa completamente conectada las neuronas están conectadas a todas las demás de sus capas previas y se suele usar para resolver el problema de la red, una vez extraídas las características más útiles. Después de la capa *softmax*, se utiliza la función de entropía cruzada, que calcula el error de clasificación de la red.

2.3.2.5 Capa *Dropout*

Se suele usar este tipo de capa para reducir el sobreajuste. Al entrenar una red, es bastante fácil que el modelo se adapte muy bien a los datos de entrenamiento, pero no sería capaz de generalizar para otros datos de entrada nuevos. Una capa *dropout* elimina las conexiones entre neuronas en las capas totalmente conectadas con una probabilidad p en cada conexión. En general, se suelen introducir entre capas totalmente conectadas y con $p = 0.5$. Si usamos $p = 0.5$, eliminamos el 50 % de conexiones entre una capa u otra.

2.3.3 Arquitecturas de redes neuronales convolucionales

Si combinamos las capas anteriores de una u otra forma, definimos la arquitectura de la red. El éxito del modelo dependerá de su arquitectura. Actualmente existen muchas arquitecturas diferentes para el tratamiento de imágenes. A continuación, se describen las más importantes para este trabajo, realizando un recorrido por el estado del arte.

2.3.3.1 Propuesta original

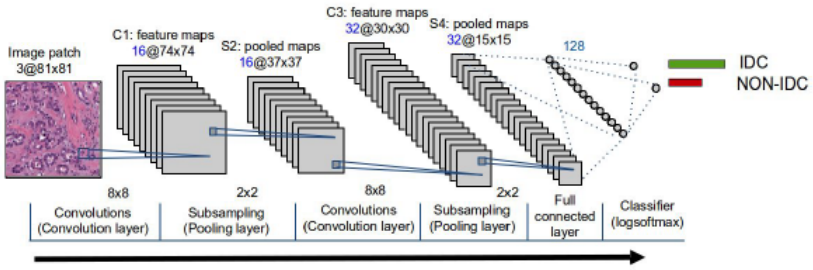


Figura 2.5: Arquitectura de la propuesta original. Fuente: Cruz-Roa et al. [2]

Cruz-Roa et al. [2] propone una arquitectura convolucional de 3 capas (2 capas convolucionales y 1 totalmente conectada). Se han utilizado 16 y 32 neuronas para las capas convolucionales, y 128 para la capa totalmente conectada. En todos los experimentos se ha utilizado un kernel de tamaño 8×8 para las capas convolucionales y 2×2 para las capas de *pooling*. A continuación, se describe el proceso para el desarrollo del modelo de clasificación:

Preprocesamiento de los datos: Cada imagen se divide en pequeños trozos llamados *patches*. Cada *patch* se convierte del espacio de color RGB a YUV y se normaliza con media cero y varianza uno. La razón de este preprocesamiento es acelerar el entrenamiento y aumentar las diferencias de las características de las imágenes de entrada.

Capa convolucional: Se utilizan dos capas convolucionales. La función de activación es la tangente hiperbólica. A continuación, se aplica una normalización por contraste para reducir el sobreajuste y mejorar el rendimiento general.

Capa de agregación: Después de cada capa convolucional, se añade otra de agregación. En este caso se aplica *L2 pooling*, en vez de seleccionar el máximo se realiza la raíz cuadrada de la suma de los cuadrados de los valores definidos por la ventana.

A continuación, se añade la capa totalmente conectada para resolver el problema.

Capa de clasificación: La capa de salida posee dos neuronas, una por cada clase. La salida viene dada por la función *softmax*. El optimizador de la red es el gradiente descendente estocástico (SGD) y la función de error es $L(x) = -\sum_i \log[\frac{e^{x_i}}{\sum_j e^{x_j}}]$, donde x_i son las salidas de cada neurona de la capa *softmax*. La salidas de la red son valores comprendidos entre 0 y 1. Dichos valores son probabilidades de tener cáncer o no. Al juntar todos los *patches* en la imagen original, podemos crear un mapa de probabilidad identificando las zonas más afectadas.

Con respecto a los resultados, se han utilizado varias medidas de rendimiento para evaluar el modelo. Las medidas utilizadas son: Precisión (Pr), Sensibilidad (Rc), Especificidad (Spc), F1-Score y *Balanced Accuracy* (BAC):

	Pr	Rc/Sen	Spc	F1	BAC
CNN	0.6540	0.7960	0.8886	**0.7180**	**0.8423**

Tabla 2.1: Resultados del modelo con varias medidas de rendimiento. Fuente: Cruz-Roa et al. [2]

Los autores han realizado una comparación del modelo basado en CNN con métodos de extracción manual de características, y un modelo de aprendizaje automático basado en bosques aleatorios (*random forest*). Los métodos de extracción manual de características se basan en las características de la imagen, como el color, texturas y bordes. Los mejores resultados los ha obtenido el modelo basado en CNN, siendo más eficaces las técnicas de *deep learning*.

2.3.3.2 AlexNet

AlexNet es una arquitectura propuesta por Krizhevsky et al. [4] para la competición *ImageNet Large-Scale Visual Recognition Challenge* (ILSVRC) del año 2012. El dataset de ImageNet está formado por 15 millones de imágenes, etiquetadas en 22.000 categorías diferentes. Para la competición se ha utilizado un subconjunto de ImageNet, formado por 1 millón de imágenes de entrenamiento, 50.000 imágenes de validación y 150.000 de prueba. Todas las imágenes están etiquetadas en 1000 categorías distintas. La propuesta consiguió reducir el error top-5 de un 26 % a un 15.3 % en el conjunto de prueba.

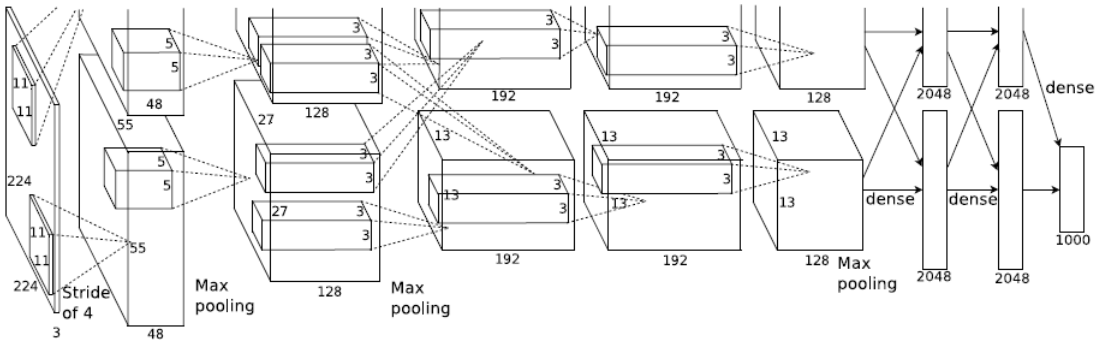


Figura 2.6: Arquitectura AlexNet. Fuente: Krizhevsky et al. [4]

La arquitectura está formada por 5 capas convolucionales y 3 capas totalmente conectadas. La red recibe como entrada una imagen a color $224 \times 224 \times 3$ y se caracteriza por usar filtros de tamaño 11×11, 5×5 y 3×3 en las capas convolucionales. También se añaden varias capas de agregación *max-pooling* en las dos primeras capas convolucionales y en la última. Se utiliza ReLU como función de activación en la salida de cada neurona. La salida de la red está formada por una capa *softmax* con 1000 valores correspondientes a las etiquetas. El optimizador de la red es el gradiente descendente estocástico (SGD) con *momentum*.

Para evitar el sobreajuste, se realiza aumento de datos (*data augmentation*). Las imágenes de entrada originales tienen una resolución de 256×256. A continuación se recortan aleatoriamente haciendo traslaciones horizontales y verticales, obteniendo *patches* de tamaño 224×224. Además, se añade una capa *dropout* con probabilidad 0.5 en las dos primeras capas totalmente conectadas. También se resta la media RGB a cada una de las imágenes de entrenamiento como preprocesamiento.

Existen varios autores que utilizan esta arquitectura para resolver el problema de clasificación en imágenes histológicas. Algunos ejemplos son Spanhol et al. [5] y Wei et al. [6]. Spanhol et al. [5] consigue superar el 80 % en la medida de rendimiento *patient score*. *Patient score* mide la relación existente entre el número imágenes con cáncer clasificadas correctamente y el número de imágenes totales del paciente. Por otro lado, Wei et al. [6] evalúa diferentes arquitecturas y consigue hasta un 90 % de *patient score* en AlexNet. Ambos autores utilizan el *dataset* BreaKHis.

2.3.3.3 VGGNet

VGG (*Visual Geometry Group*) fue propuesta por Simonyan and Zisserman [7] para la competición ILSVRC-2014, quedando en segunda posición. Esta arquitectura es similar a AlexNet y se caracteriza por usar filtros de tamaño 3×3 en las capas convolucionales. Existen varias configuraciones de esta arquitectura. En el capítulo 4, utilizaremos las configuraciones de 16 y 19 capas. En la variante de 16 capas existen 13 capas convolucionales, seguidas por una capa de *max-pooling*. Existen dos bloques formados por dos capas convolucionales y tres bloques con tres capas convolucionales. En cada bloque el número de filtros aumenta. Finalmente se añaden 3 capas totalmente conectadas, seguidas por una capa *softmax*. La variante de 19 capas consiste en añadir otra capa más en los 3 últimos bloques de capas convolucionales.

La principal desventaja de esta arquitectura consiste el número de parámetros que hay que entrenar debido al tamaño de filtro, llegando a la cifra de 128 millones de parámetros.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224×224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Tabla 2.2: Configuraciones de VGGNet. Fuente: Simonyan and Zisserman [7]

Esta arquitectura se utiliza bastante para comparar con otras propuestas y para transferencia de aprendizaje (*transfer learning*), como podemos comprobar en Khan et al. [8], [9] y Deniz et al. [10]. En general, los resultados obtenidos con VGGNet son bastante buenos. Khan et al. [8] consigue superar el 90 % de *accuracy* mientras que Deniz et al. [10] concatena las arquitecturas AlexNet y VGGNet16, logrando alcanzar un 93.02 % de precisión.

2.3.3.4 GoogLeNet

GoogLeNet, también conocida por *Inception V1*, fue la arquitectura ganadora de ILSVRC-2014, propuesta por Szegedy et al. [11]. Esta arquitectura consiguió reducir el error top-5 de ImageNet a un 6,67 %, muy cercano al rendimiento del ser humano y mejor que la tasa de error de VGGNet. Se caracteriza principalmente por reducir el alto número de parámetros de las arquitecturas vistas anteriormente, llegando a obtener 4 millones de parámetros. Por otro lado, la profundidad de la red aumenta a 22 capas.

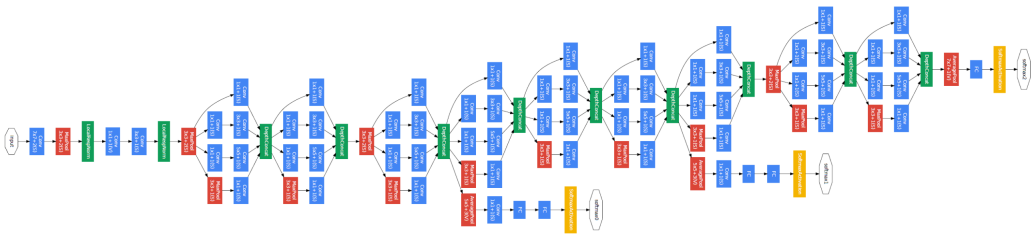
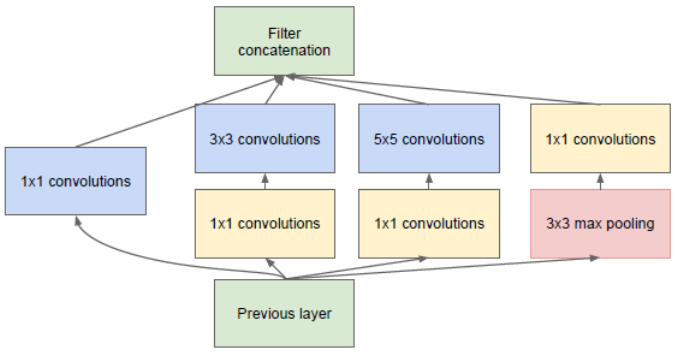


Figura 2.7: Arquitectura GoogLeNet/Inception V1. Fuente: Szegedy et al. [11]

Para reducir el número de parámetros, los autores realizaron las siguientes estrategias:

- **Uso de convoluciones 1×1:** Al utilizar un tamaño de filtro muy pequeño, disminuye bastante la cantidad de parámetros a aprender.
- **Global Average Pooling:** Se usa antes de las capas totalmente conectadas para reducir aún más el número de parámetros a entrenar. Dado que en las capas totalmente conectadas reside el mayor número de parámetros a entrenar de toda la red, es conveniente reducir su número para lograr una mayor eficiencia. Consiste en reducir los mapas de características de tamaño 7×7 a 1×1, reduciendo bastante los parámetros de las capas totalmente conectadas.
- **Módulo Inception:** En las arquitecturas descritas anteriormente, se ajusta el tamaño de filtro en cada una de las capas. En el caso de GoogLeNet, se realizan convoluciones 1×1, 3×3, 5×5 y max-pooling 3×3 en paralelo. A continuación, se concatenan todos los filtros y se pasa como entrada a la siguiente capa. La arquitectura añade una convolución 1×1 antes debido al alto coste de la operación. Usando convoluciones 1×1, se obtiene una mayor eficiencia sin perder precisión en los resultados. El objetivo del módulo consiste en aprender distintos tipos de características de la imagen usando tamaños distintos de filtros.



(b) Inception module with dimension reductions

Figura 2.8: Módulo Inception usado por GoogLeNet. Fuente: Szegedy et al. [11]

GoogLeNet es una arquitectura muy usada para todo tipo de proyectos. En el campo del diagnóstico de cáncer de mama, Khan et al. [8] y Chang et al. [12] la utilizan para extraer características, aplicando transferencia de aprendizaje. En el caso de Chang et al. [12], llega a obtener 0.93 de AUC (*Area Under the Curve*), mientras que Khan et al. [8] consigue 93.5 % de *accuracy*.

Para realizar los experimentos de este trabajo, se ha utilizado la variante *InceptionV3*, implementada por Keras. A diferencia de *InceptionV1*, *InceptionV3* utiliza convoluciones factorizadas, que ayudan a reducir el número de parámetros a entrenar, mejorando la eficiencia. También utiliza convoluciones más pequeñas de 3×3, en lugar de 5×5. Por otro lado, utiliza un clasificador auxiliar entre las capas de la red. El objetivo de este clasificador es actuar de regularizador de la red para evitar el sobreajuste. *InceptionV3* consigue mejores resultados en comparación con *InceptionV1* en ImageNet [13].

2.3.3.5 ResNet

ResNet (*Residual Neural Network*) fue propuesta en 2015 para ILSVRC-2015 por He et al. [14]. Está formada por 152 capas y obtuvo un error top-5 de 3.57 % en el *dataset* de ImageNet, superando el rendimiento humano. La idea principal se basa en el aumento de capas a través de una conexión residual en cada una de ellas. También se caracteriza por el uso de la normalización por lotes. Estas características permiten mejorar notablemente el aprendizaje, llegando obtener los mejores resultados en los últimos años.

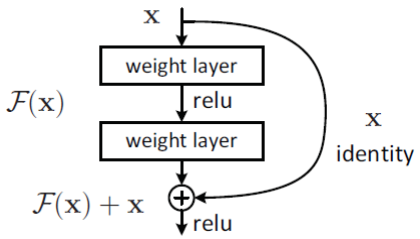


Figura 2.9: Módulo residual de *ResNet*. Fuente: He et al. [14]

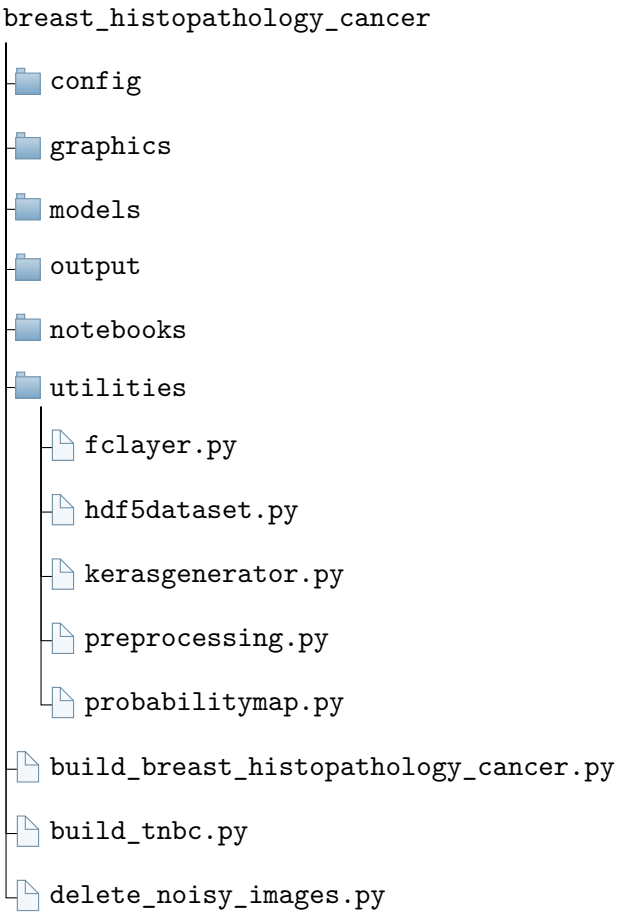
Al principio del módulo se añade la entrada, procedente de la capa anterior de la red. Los datos de la entrada se guardan conectándose con la salida del módulo. A continuación, se aplican las convoluciones, funciones de activación y normalización por lotes a la entrada. Finalmente, en la salida del módulo, añadimos los datos que guardamos anteriormente. De esta forma, añadimos los datos residuales del principio del módulo en la salida. El uso de módulos residuales permite un aprendizaje más rápido de los datos.

El uso de ResNet en la literatura es también muy frecuente. Khan et al. [8] la usa para comparar los resultados con su *framework* desarrollado, llegando a obtener un 94,89 % de *accuracy* en varios *datasets* de imágenes histológicas.

3 Experimentación

En este capítulo se desarrollan varios modelos de clasificación de imágenes con IDC. Además, se realizan las descripciones del software, la base de datos, y se analizan los resultados obtenidos.

3.1 Estructura del software



El código [15] está organizado en una estructura de directorios cuya raíz es *breast_histopathology_cancer*. A continuación, se describen cada uno de los directorios y archivos:

- `config`: En este directorio se encuentran los archivos de configuración con todas las rutas. Las rutas pueden ser de entrada (*dataset*) o de salida (modelos, gráficos, construcción del *dataset* etc.).

- En `graphics` se guardan todos los gráficos que se generan en los experimentos y las matrices de confusión.

- `models`: En esta carpeta se guardan todos los *scripts* utilizados para entrenar las arquitecturas y el *script* de evaluación.

- `output`: Directorio donde se almacenan los archivos de los modelos ya entrenados en formato *h5*. Estos archivos se pueden cargar en Keras para evaluar los modelos y para introducir nuevos datos de entada. Además, también se guarda un archivo `json`, que contiene la media de las componentes RGB en todo el *dataset* usado, y un *dataframe* con los valores de las coordenadas y rutas de cada imagen. Este *dataframe* sirve para reconstruir la imagen histológica completa de la paciente.

- `notebooks`: Directorio que incluye los documentos de Jupyter realizados para hacer el análisis exploratorio y la experimentación del capítulo 4 en Google Colab.

- En `utilities` se guardan archivos que facilitan el desarrollo de los distintos modelos:

  - `fclayer.py`: En este archivo se encuentran las implementaciones de las capas totalmente conectadas utilizadas para *fine-tuning*.

  - `hdf5dataset.py` se encarga de la creación de la base de datos HDF5.

  - `kerasgenerator.py` el objetivo de este *script* consiste en cargar, a medida que se van necesitando, las imágenes del disco en memoria para hacer el entrenamiento o para evaluar el modelo. De esta forma, se pueden cargar grandes cantidades de datos en memoria. También se encarga de aplicar el preprocesamiento a las imágenes en tiempo de entrenamiento.

  - `preprocessing.py` contiene las estrategias de entrenamiento usadas en el proyecto. En este caso, el reescalado de imágenes, la resta de la media RGB, recortes aleatorios de la imagen y la normalización entre 0 y 1.

  - `probabilitymap.py` *script* que se encarga de reconstruir la imagen histológica de la paciente y marcar las zonas de la imagen con cáncer. Un ejemplo del resultado de este script lo podemos ver en la **Figura 3.13**.

- `build_breast_histopathology_cancer.py`: Este archivo se encarga de construir el *dataset* almacenado en un archivo HDF5 y el *dataframe*. Con este *script* se pueden especificar los ejemplos de cada clase para balancear el *dataset* y también se calcula la media RGB de todo el *dataset* construido.

- `build_tnbc.py`: Se encarga de leer del disco las imágenes y las máscaras del *dataset* TNBC —usado en el capítulo 4—, y almacenarlas en dos archivos de datos NumPy respectivamente.

- `delete_noisy_images.py`: Borra todas las imágenes con resolución distinta a 50×50. Este tipo de imágenes pertenecen a los bordes de la imagen histológica y no contienen información relevante del problema.

3.2 Descripción del dataset

Uno de los *dataset* que se va a utilizar durante el desarrollo de este trabajo consiste en un conjunto de imágenes histológicas tomadas a 279 mujeres en el hospital de la Universidad de Pensilvania (EEUU) y el Instituto del cáncer de Nueva Jersey. El *dataset* ha sido obtenido en la plataforma Kaggle [16]. Las imágenes fueron digitalizadas con una magnitud de 40x. Esto supone una resolución gigantesca del orden 10^{10} píxeles. Por este motivo Cruz-Roa et al. [2] decidieron reescalar por un factor de 16:1, bajando la resolución.

Una vez reescaladas las imágenes, no solo se pretende detectar cáncer, sino las zonas donde se localiza. Por tanto, no sería recomendable pasar como entrada de la red neuronal la imagen completa. Por este motivo, en el *dataset* se dividen las imágenes histológicas en imágenes más pequeñas (*patches*) y la red neuronal consigue aprender mejor el tejido afectado por la enfermedad.

Las regiones de IDC de las imágenes en el *dataset* han sido señaladas manualmente por un experto. Estas anotaciones fueron realizadas mediante un programa informático. Con el uso de técnicas de *deep learning* podemos marcar las zonas de IDC de forma más precisa que el experto, ya que la red habrá aprendido como se comportan los tejidos que poseen IDC durante el entrenamiento. También ahorraremos tiempo y esfuerzo al experto a la hora de diagnosticar la enfermedad.

Por último, veremos como se estructuran los datos en el disco. En la raíz veremos un listado de carpetas cuyos nombres representan los identificadores de las pacientes en formato numérico. Dentro de cada carpeta encontramos otras dos subcarpetas con las etiquetas 0 y 1. En la carpeta 0 encontraremos aquellas regiones de la imagen histológica de la paciente donde no está presente el IDC. Análogamente, la carpeta 1 tendrá aquellas regiones que tengan cáncer.

El nombre de los archivos sigue el siguiente formato:

```
{id_paciente}_idx5_{coordenada_x}_y{coordenada_y}_class{numero_clase}.png
```

Las cooordenadas x e y representan las coordenadas de la imagen histológica.

3.3 Herramientas utilizadas

Por último, es conveniente señalar las herramientas que se han utilizado para llevar a cabo este trabajo:

Python: Actualmente es uno de los mejores lenguajes de programación para trabajar en el ámbito de la ciencia de datos debido al gran número de librerías existentes, su sintaxis sencilla y la facilidad de aprendizaje.

OpenCV: Librería *open source* para el procesamiento de imágenes y que es compatible con la librería NumPy. Usaremos esta librería para realizar tareas como el reescalado de imágenes, conversión de espacios de color, etc.

NumPy: Es una librería para el cálculo eficiente de operaciones con vectores y matrices.

Matplotlib: Usaremos Matplotlib para la visualización de los datos.

Keras: Es una biblioteca de algoritmos de *deep learning*. Con Keras ajustaremos los hiperparámetros de las distintas capas, utilizaremos varias funciones de activación, optimizadores y evaluaremos el rendimiento del modelo.

Todos los experimentos se han llevado a cabo en un ordenador con las siguientes prestaciones:

CPU	Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz 6 procesadores principales, 12 procesadores lógicos
GPU	NVIDIA Geforce GTX 1650 4GB
RAM	16GB

3.4 Análisis exploratorio

El primer paso a la hora de trabajar con un conjunto de datos consiste en realizar una exploración para ver la distribución de los datos y si son correctos. En esta sección realizaremos dicha exploración y veremos si hay que transformar los datos antes del entrenamiento.

Lo primero que deberíamos de hacer es ver de cuántos datos disponemos realmente y la dimensión de las imágenes. Utilizando la librería del sistema de Python podemos ver que tenemos 279 carpetas, correspondiente al número de pacientes. En total tenemos 277.524 imágenes que podemos utilizar para entrenar y validar el modelo.

Otro aspecto a destacar es la resolución de las imágenes. En este caso, tenemos imágenes con una resolución de 50×50 píxeles y 3 canales correspondientes al espacio de color RGB. El tamaño es suficiente para entrenar la red, ya que una resolución muy alta dificulta el entrenamiento de la red, haciendo necesario un hardware más potente.

Uno de los aspectos más importantes del análisis exploratorio es comprobar si existe desbalanceo de clases. En el conjunto de datos tenemos 198.738 imágenes que no tienen IDC y 78.786 imágenes que sí lo tienen. A simple vista podemos decir que las clases están desbalanceadas. Si calculamos la frecuencia relativa vemos que el 71,6 % de las imágenes no tienen cáncer, frente al 28,3 % que sí tienen.

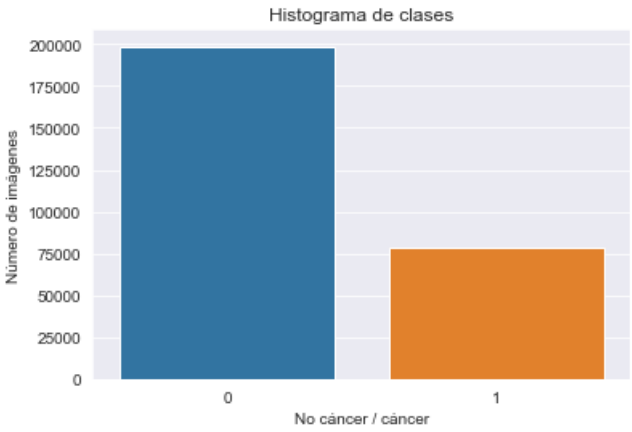


Figura 3.1: Histograma de clases

Observando las gráficas de las distribuciones podemos ver que no son uniformes. La mayoría de pacientes tienen entre 800 y 1200 imágenes mientras que existen muy pocos que tengan más de 2000. Por otro lado, la mayoría de pacientes tienen pocas imágenes de cáncer; otra señal más del desbalanceo.

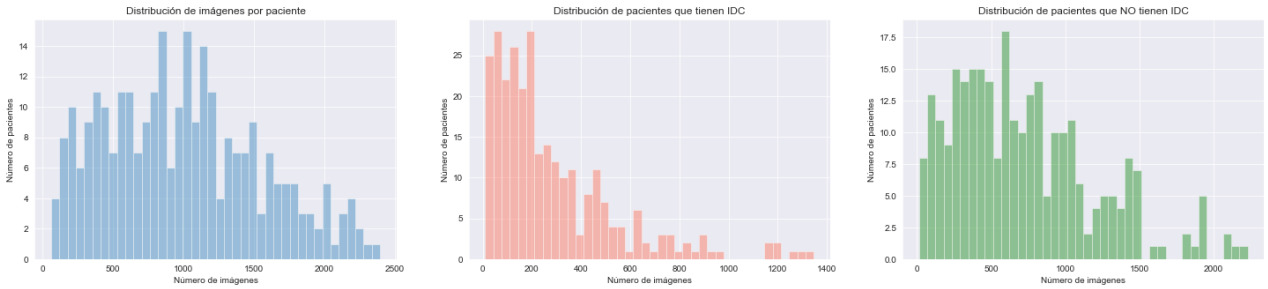


Figura 3.2: Distribuciones de imágenes

A continuación, vamos a estudiar si existen diferencias significativas entre las distintas clases de imágenes:

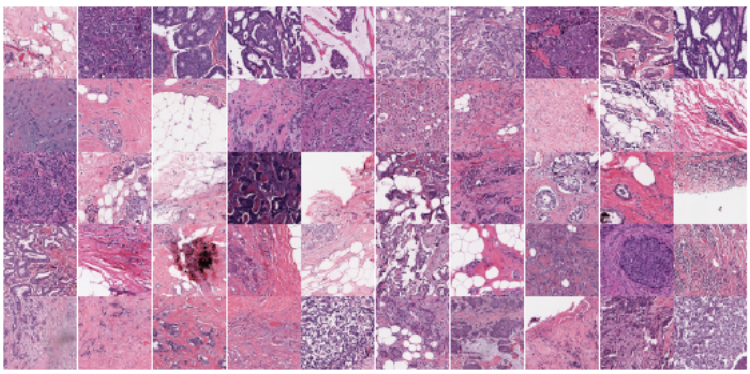


Figura 3.3: Imágenes con IDC

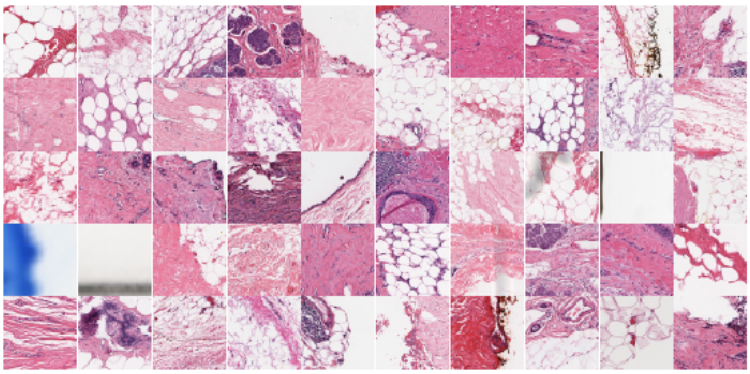


Figura 3.4: Imágenes sin IDC

Podemos apreciar que las imágenes con IDC tienen un color más ocuro. Esto puede ser relevante para el algoritmo de aprendizaje porque puede clasificar como cáncer a las imágenes con píxeles más ocuros.

Conclusiones

- Los datos están desbalanceados. Existen varias técnicas para solucionar este problema común:
  - **Submuestreo**: Se trata de reducir la muestra de la clase mayoritaria para equilibrar las clases. La desventaja es que se pierde información que puede ser relevante.
  - **Sobremuestreo**: Aumentamos la clase minoritaria con ejemplos parecidos. No se pierde información, pero al tener tantos datos parecidos puede llevar a un sobreajuste.
  - **Penalización en el error**: Algunos algoritmos penalizan más el error cuando se clasifica incorrectamente la clase minoritaria.

- No existe el mismo número de imágenes por paciente. Esto puede deberse a la resolución de las imagenes histológicas originales, ya que todas no son del mismo tamaño.
- Existe un problema de memoria al cargar todo el *dataset*, debido a la gran cantidad de imágenes. Por tanto, es importante procesar el *dataset* de una forma adecuada para una máquina con recursos limitados. Para solucionarlo, se pueden usar las siguientes estrategias:
  - **Submuestreo**: Al reducir la muestra de la clase mayoritaria se reduce también el número de imágenes en la RAM, pero se pierde información.
  - **Entrenamiento por lotes**: Consiste en el cargar poco a poco el *dataset* en memoria, a medida que se vaya necesitando. Para la librería *Keras* se puede realizar un generador que vaya entrenando la red neuronal poco a poco.

3.5 Preprocesamiento

En primer lugar, se han eliminado las imágenes con una resolución distinta a 50×50. Estas imágenes pertenecen a los bordes de la imagen histológica y no contienen información relevante para el problema.

Debido a la limitación de recursos de la máquina usada para los experimentos, es necesario reducir el *dataset* para una mayor eficiencia. Con la reducción del número de imágenes, liberamos carga tanto para la CPU como la GPU. En este caso, se ha reducido el *dataset* a **20.000** imágenes en total. A continuación, se han dividido las imágenes en conjuntos de entrenamiento, validación y prueba. El conjunto de entrenamiento está formado por 14.470 imágenes (75 %); el de validación, por 2.550 imágenes (15 %), y el de prueba, por 3.000 imágenes (15 %).

A la hora de hacer la reducción de datos, se ha tenido en cuenta el número de imágenes de cada clase para evitar el desbalanceo. Se ha utilizado el paquete *imblearn* para balancear los datos. De esta forma, existe el mismo número de imágenes de cada clase en todo el *dataset*. Además, los subconjuntos de entrenamiento, validación y prueba están estratificados, es decir, tienen también el mismo número de imágenes de cada clase.

Para almacenar la imágenes en el disco, se ha usado el formato HDF5. El formato HDF5 es muy usado para gestionar conjuntos de datos grandes que no caben en memoria. Al abrir la base de datos HDF5, no se carga el dataset completo en la memoria. Se puede acceder a los datos a medida que los vayamos necesitando. Un archivo HDF5 está formado por bloques, donde se pueden almacenar distintos *datasets* en cada uno de ellos. Para este problema se han usado dos datasets: uno para guardar los valores de los píxeles de la imagen y otro para las etiquetas de dichas imágenes.

Por otro lado, se ha creado un generador para Keras. El objetivo del generador es cargar, a medida que se necesiten, los datos del disco sin saturar la memoria RAM del equipo. También se aplicarán las opciones de preprocesamiento en tiempo de entrenamiento.

Por último es necesario normalizar los datos de las imágenes. Una red neuronal trabaja mejor con números pequeños, comprendidos entre 0 y 1. Si usamos números grandes, a la red le costará más trabajo realizar los cálculos para el aprendizaje. En este caso, se ha restado la media de las componentes RGB de todo el *dataset* a cada una de las imágenes. A raíz de los experimentos realizados, este tipo de normalización ofrece mejores resultados que la normalización 0 y 1, que consiste en dividir cada una de las componentes RGB por 255.

3.6 Estrategia de entrenamiento

Para realizar el entrenamiento, se han usado estas dos estrategias importantes:

3.6.1 Transferencia de Aprendizaje (*Transfer Learning*)

La transferencia de aprendizaje es una técnica bastante usada en la literatura de *deep learning* por los grandes resultados que obtiene. La idea consiste en aprovechar las características aprendidas en una red entrenada previamente con un *dataset* igual o distinto del problema a resolver. De esta forma, no es necesario entrenar una arquitectura desde cero, ahorrando recursos computacionales y de tiempo. En general, entrenar una red neuronal desde cero suele tardar varios días o incluso semanas. Por ello, utilizaremos arquitecturas preentrenadas con los pesos de ImageNet. Podemos usar transferencia de aprendizaje de dos maneras distintas:

3.6.1.1 Extracción de características

Para extraer las características, se eliminan las capas totalmente conectadas para aprovechar solamente las características de las capas convoluciones. A continuación, se introducen todas las imágenes del *dataset* en la red. No se realiza entrenamiento, sino que se evalúan cada uno de los datos. Dichas imágenes de salida, dependiendo de la arquitectura, tendrán una resolución distinta de la imagen original. Por último, se realiza un entrenamiento de las características extraídas con un clasificador tradicional (p. ej. Regresión logística o SVM). La extracción de características funciona bien cuando se tienen datos parecidos a los datos que se utilizaron para entrenar la red y un *dataset* pequeño. En el contexto de este trabajo, esta técnica queda descartada debido a la gran cantidad de imágenes y la variabilidad de los datos con respecto a ImageNet, ya que tenemos imágenes biomédicas. Por tanto, necesitamos una técnica más adecuada como *fine-tuning*.

3.6.1.2 *Fine-Tuning*

Al contrario que la extracción de características, *fine-tuning* nos permite aprovechar las ventajas de la transferencia de aprendizaje en *datasets* grandes y distintos a los *datasets* originales. Para aplicar *fine-tuning*, en primer lugar, eliminamos las capas totalmente conectadas de la arquitectura y las sustituimos por una capa totalmente conectada más simple. A continuación se aplican las dos fases siguientes:

- Se «congelan» las capas anteriores a la capa totalmente conectada. De esta manera, extraemos las características de las capas convolucionales y entrenamos solo los pesos de la última capa. Esto se hace para aprovechar las características aprendidas en la última capa, que se utiliza para resolver el problema.

- Una vez que tengamos suficiente precisión (*accuracy*), podemos «descongelar» las últimas capas convolucionales o la arquitectura completa para aprender características más específicas del problema. Así pues, sería recomendable usar una tasa de aprendizaje muy baja para no modificar radicalmente las características aprendidas en la fase anterior. Se realizará el entrenamiento durante un número determinado de épocas hasta que la precisión del modelo aumente.

3.6.2 *Data Augmentation*

Al recorrer el conjunto de datos de entrenamiento varias veces, corremos el riesgo de sobreajustar la red. El sobreajuste se produce cuando el modelo se apega a los datos de entrenamiento y no consigue generalizar bien ante nuevos datos de entrada, produciendo un error de entrenamiento muy bajo y un error de validación muy alto. Para evitar el sobreajuste, se suelen utilizar técnicas como *data augmentation*. Al usar este tipo de técnicas aumentamos el error en el conjunto de entrenamiento, pero en cambio reducimos el error en los conjuntos de entrenamiento y prueba

Data augmentation (aumento de datos en español) es una técnica que consiste en añadir nuevos datos en cada época de entrenamiento de la red, introduciendo modificaciones en las imágenes. Dichas modificaciones pueden ser el escalado, recorte, rotación o hacer zoom de la imagen. De esta forma, en cada época la red recibe datos distintos a los anteriores, reduciendo el sobreajuste.

3.7 Análisis de resultados

En este apartado se muestran los resultados obtenidos y su análisis correspondiente. Dichos resultados se obtienen mediante la evaluación de cada uno de los modelos en el conjunto de prueba, obtenido en la etapa de preprocesamiento.

A la hora de evaluar un modelo, es conveniente usar una medida de rendimiento adecuada para medir la calidad del mismo. Las medidas más importantes en el contexto de este trabajo son las siguientes:

Precisión (accuracy)

$$accuracy = \frac{VP + VN}{VP + FP + VN + FN}$$

La precisión relaciona los datos etiquetados correctamente por el modelo, positivos (VP) y negativos (VN) con todos los datos del modelo, tanto falsos positivos (FP) como falsos negativos (FN). La precisión es una métrica que funciona bien cuando los datos están balanceados. Si tuviéramos un dataset donde el 98 % de las imágenes tienen tejido canceroso, al usar un modelo que siempre etiqueta positivamente, obtendríamos un 98 % de precisión. Una medida errónea. Dado que el *dataset* preprocesado está balanceado, esta será la medida principal para evaluar la calidad del modelo.

Exactitud (precision)

$$precision = \frac{VP}{VP + FP}$$

La exactitud relaciona el número de casos positivos etiquetados correctamente por el modelo con todos los casos positivos detectados por el modelo.

Sensibilidad (Recall)

$$recall = \frac{VP}{VP + FN}$$

La sensibilidad relaciona el número de casos positivos etiquetados correctamente por el modelo con todos los casos positivos reales. La sensibilidad es una medida crucial para el diagnóstico de enfermedades, ya que se podrían etiquetar casos negativos cuando en realidad son positivos, y por lo tanto, tienen cáncer. Por este motivo, es importante tener en cuenta esta métrica, que penaliza los falsos negativos frente a los falsos positivos.

Para medir estas métricas, se ha utilizado la librería Scikit-learn para calcular los resultados y se ha creado una matriz de confusión para visualizar mejor los resultados.

3.7.1 VGGNet

3.7.1.1 Estrategia de entrenamiento para VGG16

Para implementar la arquitectura VGGNet de 16 capas, se han utilizado los modelos preentrenados existentes en la librería Keras con los pesos aprendidos en el entrenamiento de ImageNet. Keras dispone de las configuraciones de 16 y 19 capas. En cuanto al preprocesamiento, la imágenes se han redimensionado a 224×224 píxeles para evitar conflictos en la entrada de la arquitectura. A continuación, se han aplicado las técnicas de *data augmentation*, volteando las imágenes horizontal y verticalmente, rotando 90° aleatoriamente y haciendo zoom del 50 %. El tamaño de *batch* para *data augmentation* es 1, es decir, se aplican las transformaciones en cada una de las imágenes manteniendo el tamaño original del conjunto de entrenamiento. Para el entrenamiento, se ha usado un tamaño de batch de 32 y se ha establecido el número de épocas a 20. El entrenamiento se ha llevado a cabo en dos fases. En la primera fase, se congelan todas las capas de la arquitectura y se sustituyen las capas totalmente conectadas por una más simple. En la segunda fase se entrenan los pesos de las 3 últimas capas convolucionales y los de la capa totalmente conectada. En ambas fases se utiliza el mismo número de épocas y tamaño de *batch*. En la primera fase se usa como tasa de aprendizaje $\alpha = 0.01$, mientras que en la segunda fase se usa $\alpha = 0.001$ para no cambiar radicalmente los pesos aprendidos.

La capa totalmente conectada está compuesta por 256 neuronas, seguida de una capa *dropout* con probabilidad 0,5 para reducir el sobreajuste. Finalmente se añade la capa *softmax*, que muestra las salidas de la red.

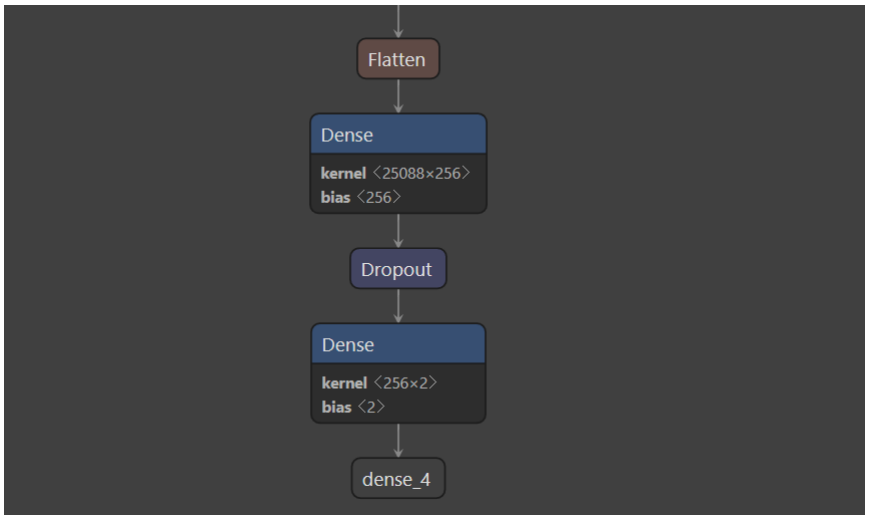


Figura 3.5: Arquitectura de la capa totalmente conectada en VGG16 y VGG19

3.7.1.2 Estrategia de entrenamiento para VGG19

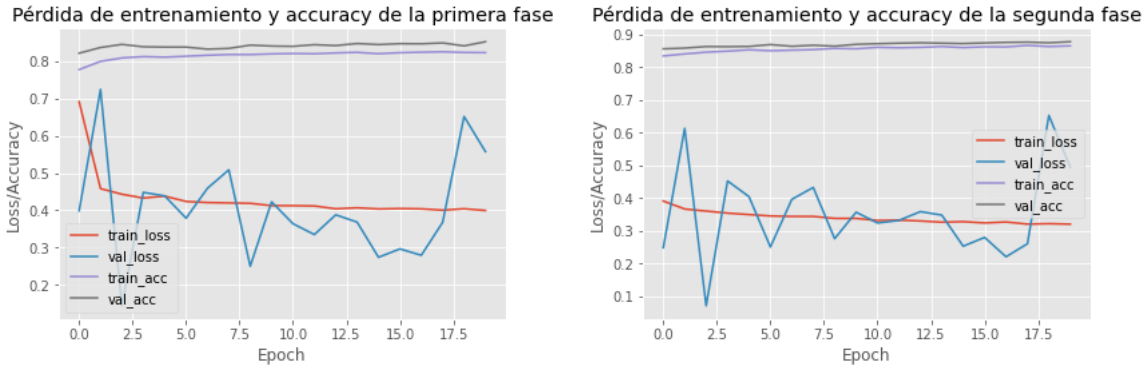
Los hiperparámetros son similares a la arquitectura anterior. En este caso, se ha utilizado el modelo preentrenado que ofrece Keras. Los mejores resultados se han obtenido con un tamaño de *batch* de 64 y el número de épocas a 15. Se ha bajado el número de épocas porque la precisión no aumenta con un número mayor. Al igual que en la arquitectura anterior, se redimensionan las imágenes a 224×224 y se utilizan la misma configuración de *data augmentation*. El entrenamiento también se ha llevado a cabo en dos fases, en otras 15 épocas. En la primera fase se congela toda la red, excepto la capa totalmente conectada. Se ha usado también la misma capa conectada que VGG16. Finalmente, en la segunda fase se descongelan las 4 últimas capas convoluciones para aprender más características. El optimizador usado es SGD con $\alpha = 0.01$ en las dos fases.

	precision	recall	f1-score	support
non-cancer	0.90	0.84	0.87	1500
cancer	0.85	0.91	0.88	1500
accuracy			0.87	3000
macro avg	0.88	0.87	0.87	3000
weighted avg	0.88	0.87	0.87	3000

Tabla 3.1: Resultados en diferentes métricas del modelo VGG16

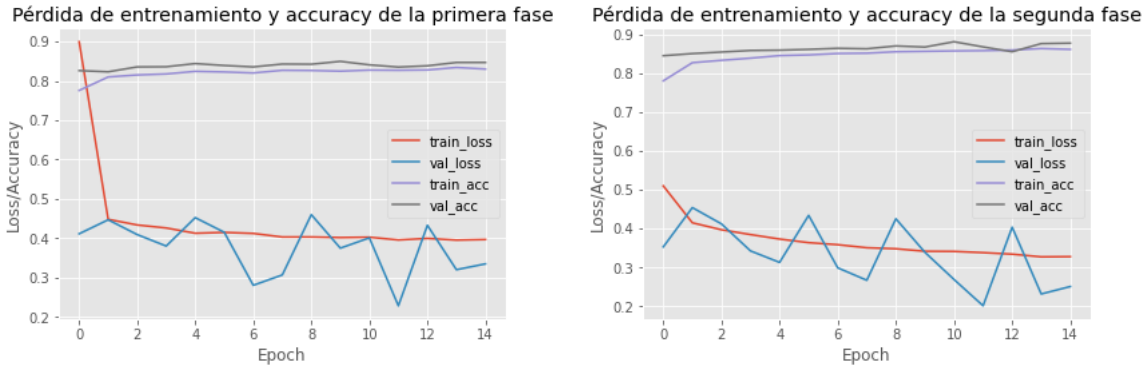
	precision	recall	f1-score	support
non-cancer	0.86	0.88	0.87	1500
cancer	0.88	0.85	0.86	1500
accuracy			0.87	3000
macro avg	0.87	0.87	0.87	3000
weighted avg	0.87	0.87	0.87	3000

Tabla 3.2: Resultados en diferentes métricas del modelo VGG19



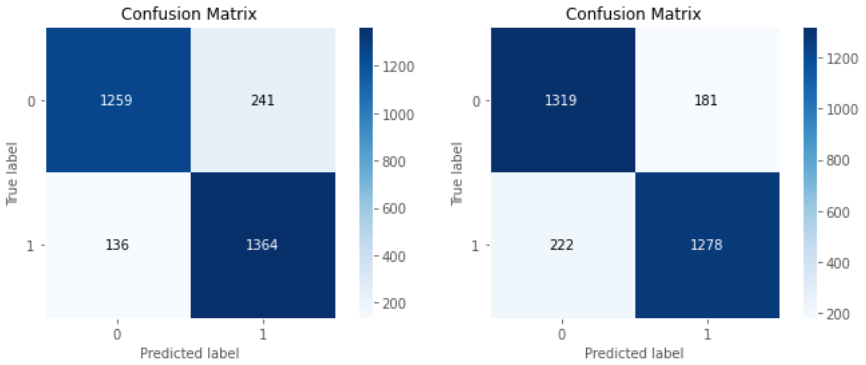
(a) Primera fase **(b)** Segunda fase

Figura 3.6: Evolución de las funciones de pérdida y precisión en VGG16 durante el entrenamiento



(a) Primera fase **(b)** Segunda fase

Figura 3.7: Evolución de las funciones de pérdida y precisión en VGG19 durante el entrenamiento



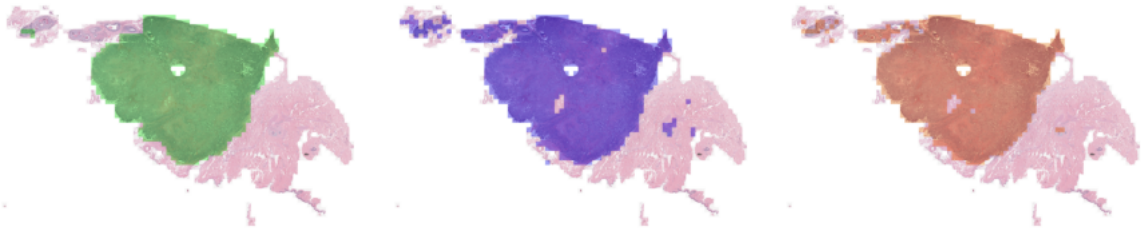
(a) VGG16 **(b)** VGG19

Figura 3.8: Matrices de confusión

Podemos observar que los resultados son muy similares en ambas arquitecturas. VGG16 y VGG19 obtienen una precisión (*accuracy*) de 87 %. El resultado es bastante bueno y al usar un conjunto de entrenamiento balanceado, esta medida es bastante significativa. Si nos fijamos en el *recall* de la etiqueta cáncer, el resultado es mejor aún en el caso de VGG16, llegando al 91 %. Esto significa que existen muy pocos falsos negativos y el modelo ha conseguido aprender las características presentes en un tejido con cáncer. En el caso del modelo VGG19, el número de falsos negativos crece, disminuyendo el *recall*. Aun así, el resultado sigue siendo bastante positivo, ya que disminuye el número de falsos positivos con respecto a VGG16 y el número de falsos negativos sigue siendo bajo. Al obtener un *recall* más bajo, la exactitud (*precision*) aumenta.

Podemos ver los resultados con más claridad en la **Figura 3.8**, donde se muestran las matrices de confusión de cada modelo. Vemos que, efectivamente, los resultados son bastante buenos. El modelo VGG16 consigue clasificar correctamente 2623 ejemplos de 3000 en el conjunto de prueba. Entre los ejemplos que el modelo ha clasificado incorrectamente, vemos que el número de falsos negativos es menor que el de falsos positivos. Por otro lado, el modelo VGG19 ha conseguido clasificar 2597 ejemplos correctamente y obtiene un número más alto de falsos negativos que de falsos positivos.

En la **Figura 3.6** y **Figura 3.7** podemos ver la evolución de los valores de error de los dos modelos. En las dos primeras épocas experimentamos un fuerte descenso del error de entrenamiento. Esto sucede porque la red modifica los pesos de ImageNet iniciales por los pesos que va aprendiendo durante la transferencia de aprendizaje. Durante el resto de épocas el valor del error de entrenamiento va bajando poco a poco; intentando encontrar un mínimo en la función de error. Al usar datos distintos en cada época, el error de validación se vuelve inestable. Dependiendo de la época el error de validación sube o baja. Sin embargo, a causa de la técnica *data augmentation*, conseguimos que la precisión del conjunto validación supere a la del conjunto de entrenamiento. Esto indica que el modelo ha conseguido generalizar las características aprendidas para otros datos de entrada, tal y como hemos visto en los resultados anteriores.



(a) Predicciones reales. **(b)** Predicciones VGG16. **(c)** Predicciones VGG19.

Figura 3.9: Imagen histológica de la paciente 14154.

En la **Figura 3.13**, podemos observar las predicciones realizadas por los modelos sobre la imagen histológica de una paciente. Podemos ver que el modelo VGG16 detecta las región principal donde se sitúa el cáncer de una forma fiel a las predicciones realizadas por el experto; aunque existen varios falsos positivos. En este caso, se puede apreciar que VGG19 ha detectado la región más afectada por el cáncer, y también ha detectado falsos positivos como VGG16; sin embargo, el número de falsos positivos detectados es menor.

Como conclusión, podemos afirmar que el modelo VGG16 funciona mejor que VGG19, debido al número de falsos negativos. En el diagnóstico de enfermedades, se prefiere tener un número más bajo de falsos negativos que de falsos positivos.

3.7.2 GoogLeNet

Para implementar GooLeNet, se ha utilizado el módulo InceptionV3 proporcionado por Keras. En este experimento, se mantienen la mayoría de las configuraciones usadas para la arquitectura anterior. En el caso de *InceptionV3* se redimensiona la imagen de entrada a 299×299 y se añade una capa de *average pooling* con un filtro de tamaño 8×8 al inicio de la capa totalmente conectada. A continuación, se entrena la capa totalmente conectada con los pesos de ImageNet. El optimizador usado es *Adam* con $\alpha = 0.0001$ y con *decay*. El optimizador *Adam* se suele usar bastante en *deep learning* y funciona mejor que SGD en muchas ocasiones. La idea básica de *Adam* consiste en usar una tasa de aprendizaje adaptativa durante el entrenamiento para mejorar la precisión del modelo.

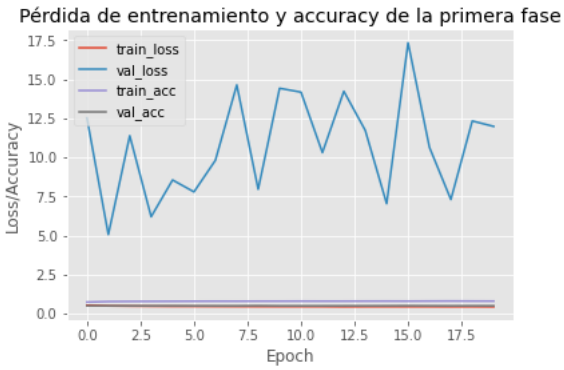


Figura 3.10: Evolución del error de entrenamiento y validación en *InceptionV3*

	precision	recall	f1-score	support
non-cancer	0.50	0.99	0.66	1500
cancer	0.49	0.01	0.02	1500
accuracy			0.50	3000
macro avg	0.49	0.50	0.34	3000
weighted avg	0.49	0.50	0.34	3000

Tabla 3.3: Resultados en diferentes métricas del modelo *InceptionV3*

En la **Figura 3.10** vemos la evolución del error durante el entrenamiento. Podemos observar que existe un gran sobreajuste de los datos. Vemos como el error de entrenamiento es muy bajo mientras que el error de validación es muy alto. Por tanto, el modelo se ha ajustado a los datos de entrenamiento y no ha conseguido generalizar las características. Una de las causas del sobreajuste puede ser el número de capas de la arquitectura. A medida que crece el número de capas en una arquitectura, el riesgo de sobreajuste aumenta.

Por último, si observamos la **Tabla 3.3** vemos las medidas del modelo. Los resultados son bastante malos, ya que obtenemos una precición (*accuracy*) de 0.5 y un *recall* de 0.01. Estos resultados reflejan que el modelo es muy poco fiable y con un gran número de falsos negativos.

3.7.3 ResNet

Para realizar la implementación de la arquitectura ResNet, se ha utilizado el módulo ResNet50 de Keras. ResNet50 es una variante de la arquitectura con 48 capas convolucionales, 1 capa *max-pooling* y una capa *average-pooling*. Para el entrenamiento se ha utilizado un tamaño de *batch* de 64 y se han redimensionado las imágenes a 224×224. Se ha añadido a la capa totalmente conectada una capa *average-pooling* con un tamaño de filtro 7×7. Finalmente, se ha entrenado la capa totalmente conectada realizando *fine-tuning* durante 20 épocas y con el optimizador *Adam*. El optimizador se ha aplicado con $\alpha = 0.0001$ y con *decay*.

	precision	recall	f1-score	support
non-cancer	0.85	0.75	0.80	1500
cancer	0.78	0.87	0.82	1500
accuracy			0.81	3000
macro avg	0.82	0.81	0.81	3000
weighted avg	0.82	0.81	0.81	3000

Tabla 3.4: Resultados en diferentes métricas del modelo *ResNet50*

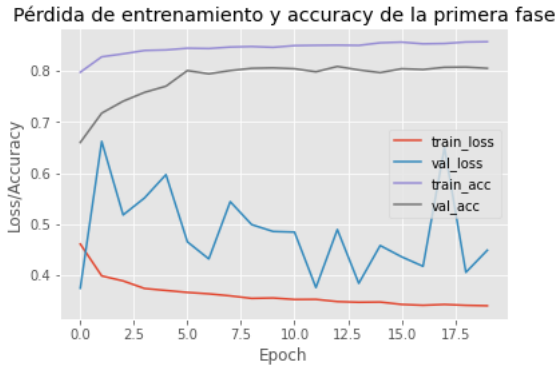


Figura 3.11: Evolución del error de entrenamiento y validación en *Resnet50*

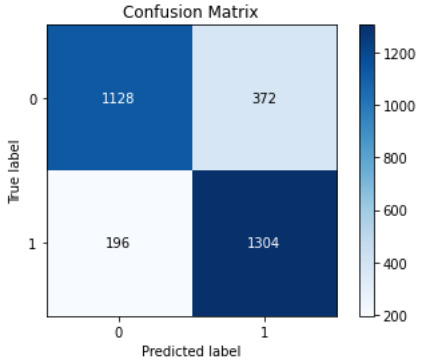


Figura 3.12: Matriz de confusión del modelo *ResNet50*

(a) Predicciones reales. **(b)** Predicciones ResNet50.

Figura 3.13: Imagen histológica de la paciente 14191.

Vemos que los resultados que ofrece ResNet50 son bastante aceptables. Si nos fijamos en la **Figura 3.11** vemos que el error de entrenamiento es muy bajo y la precisión de validación alta, superando a la precisión obtenida por los datos de entrenamiento. En la **Tabla 3.4** vemos que el modelo ha obtenido un 81 % de *accuracy* y un 87 % de *recall*. Estos resultados mejoran notablemente a los del modelo InceptionV3. Si observamos la matriz de confusión, vemos que el modelo penaliza más los falsos negativos que los positivos, ya que el número de falsos negativos es inferior. No obstante, los resultados son un poco peores que los que ofrecen los modelos *VGG16* y *VGG19*.

3.7.4 Conclusiones

Modelo	VGG16	VGG19	InceptionV3	ResNet50	Cruz-Roa et al. [2]
Accuracy	**0.87**	**0.87**	0.5	0.81	
BAC	**0.88**	0.87	0.5	0.81	0.84

Tabla 3.5: Resumen de los resultados obtenidos

Como conclusión, podemos decir que el modelo VGG16 es el mejor modelo, ya que obtiene un menor número de falsos negativos que VGG19. Este modelo supera a la arquitectura de tres capas propuesta por Cruz-Roa et al. [2] y, por tanto, supera a los métodos manuales de extracción de características presentados en el artículo. Por otro lado, el peor modelo corresponde a InceptionV3, debido al gran sobreajuste de los datos de entrenamiento.

4 Segmentación

En el capítulo anterior hemos desarrollado varios de modelos de clasificación que detectan si existe cáncer en una imagen histológica. Sin embargo, el modelo no indica las zonas precisas donde se sitúa el cáncer. Para resolver esta tarea, es necesario usar técnicas de segmentación. En este capítulo desarrollaremos otro modelo que permitirá detectar la localización de células cancerosas en imágenes histológicas.

4.1 Segmentación de imágenes

La segmentación es el proceso de dividir una imagen en distintas zonas o regiones. A continuación, se muestra un ejemplo para explicar mejor este concepto y sus tipos:

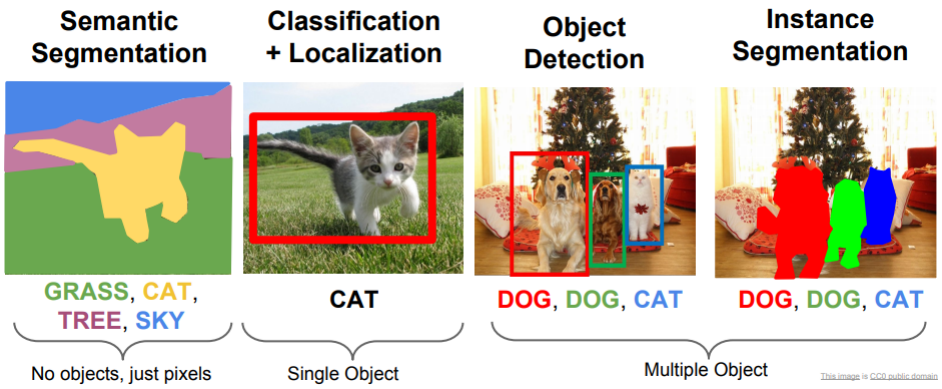


Figura 4.1: Comparativa de técnicas de segmentación. Fuente: Li et al. [17]

- **Clasificación**: Como se ha explicado anteriormente, el modelo del capítulo anterior clasifica una imagen completa. De esta forma, la imagen solo puede pertenecer a una clase. En este caso, estaríamos hablando de clasificación, ya que no dividimos la imagen en zonas según la clase.

- **Segmentación semántica**: Consiste en etiquetar cada uno de los píxeles de la imagen. De esta forma, puede haber múltiples clases por imagen. En la figura anterior vemos que la imagen del gato se puede dividir en zonas distintas según la clase. Cada zona está asociada con un color; de esta forma, podemos identificar las clases gato, cielo, árbol y césped. Podemos ver que, con la segmentación semántica, podemos identificar de manera precisa la localización de las clases de la imagen.

- **Clasificación + localización**: En este caso, se realiza la clasificación de la imagen completa, indicando que pertenece a una sola clase (gato), y se localiza la posición del

objeto mediante un cuadro delimitador (*bounding box*).

- **Detección de objetos**: Se detecta la posición de los posibles objetos de una imagen con un cuadro delimitador.

- **Segmentación de instancias**: Es la combinación de las técnicas de segmentación semántica y detección de objetos. Con segmentación de instancias, se etiquetan los píxeles de cada clase de objetos, delimitando la posición del objeto de forma precisa. Si existen varios objetos de la misma clase, se indican con un color distinto, mostrando el número de instancias de la clase. En el ejemplo de la figura superior, existen dos instancias de la clase perro, ambas con un color diferente. Entre las aplicaciones de la segmentación de instancias, destaca su uso en el desarrollo de vehículos autónomos.

4.2 Arquitecturas de redes neuronales para segmentación semántica

Para segmentar una imagen, es necesario recurrir a otro tipo de arquitecturas de redes neuronales. A continuación, se describen las principales arquitecturas utilizadas actualmente:

4.2.1 Arquitecturas completamente convolutivas (FCN)

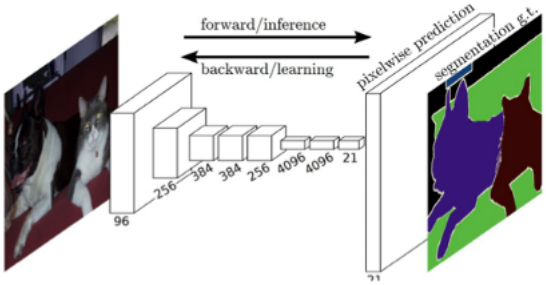


Figura 4.2: Ejemplo de arquitectura completamente convolutiva. Fuente: Long et al. [18]

Una red completamente convolucional (*fully convolutional network* en inglés) se compone de capas convolucionales, que van extrayendo las características de la imagen de entrada y reduciendo el tamaño de las salidas en cada capa. El proceso anterior se denomina *down-sampling*. A continuación, se eliminan las capas completamente conectadas y se aumenta el tamaño de las salidas mediante *upsampling*. Para hacer *upsampling*, se utiliza la deconvolución o convolución traspuesta. Finalmente, se genera el mapa de segmentación de la imagen de entrada como salida de la red, combinando la información obtenida en el proceso de *down-sampling* mediante *skip connections*. A causa de la pérdida de información espacial de las capas convolucionales y de *pooling*, es necesario obtener la información de las capas previas para construir el mapa de segmentación.

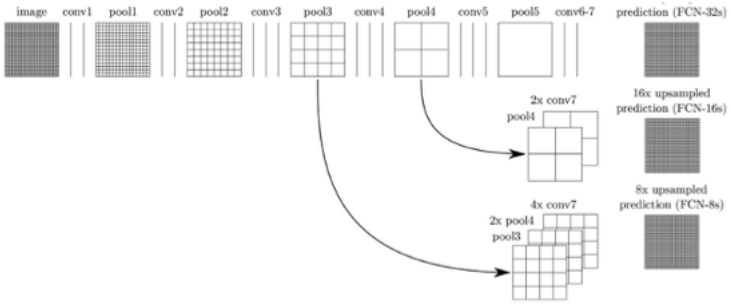


Figura 4.3: Fusión de la información de las capas previas mediante *skip connections*. Fuente: Minaee et al. [19]

4.2.2 U-Net

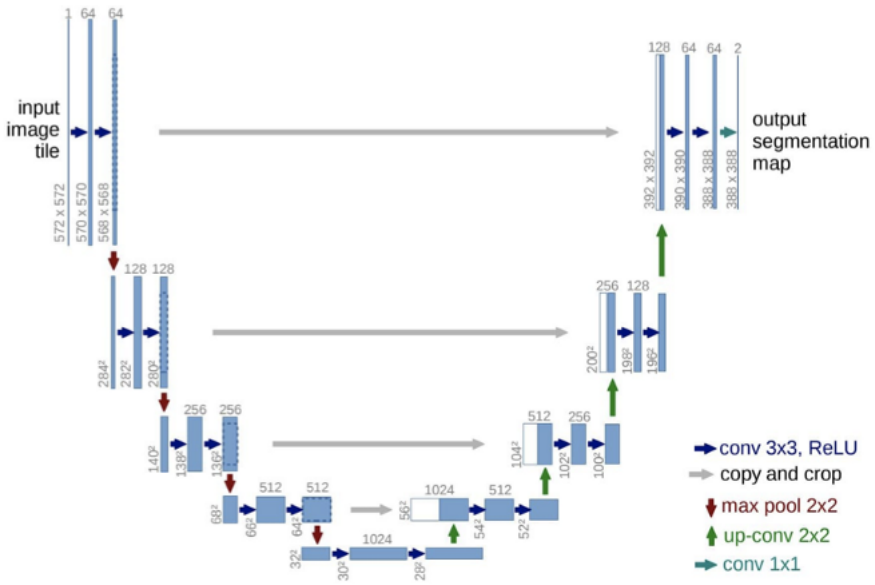


Figura 4.4: Arquitectura U-Net. Fuente: Ronneberger et al. [20]

U-Net es una arquitectura para segmentación de imágenes biomédicas y se caracteriza por sus buenos resultados de entrenamiento con pocas imágenes de entrada. La arquitectura recibe ese nombre por su forma de «U». Está basada en una arquitectura FCN y está formada por las siguientes partes:

- **Codificador** (*encoder*): Se encarga del proceso de *downsampling*. La red recibe como entrada la imagen a segmentar y se aplican de forma escalonada dos capas convolucionales, con filtros de tamaño 3×3, seguidas de una capa *max-pooling*, con filtros de

tamaño 2×2. De esta forma, a medida que se va bajando de nivel, el tamaño de la salida se reduce.

- **Decodificador** (*decoder*): En esta etapa se lleva a cabo el proceso de *upsampling* de forma simétrica a la etapa anterior. Para ello, la información de cada salida de las capas convolutivas se combina para recuperar la información espacial perdida, mediante el uso de *skip connections*. Finalmente, se aplica una última capa convolucional de tamaño 1×1 para obtener el mapa de segmentación.

U-Net es la arquitectura más usada en segmentación de núcleos celulares debido a sus buenos resultados. Naylor et al. [21], Hossain [22] y Long [23] son ejemplos de su uso. Naylor et al. [21] y Long [23] llegan a obtener un 0.77 (Dice *score*) y 0.53 (media IoU), respectivamente. Por otro lado, Hossain [22] usa una modificación de U-Net para detectar microcalcificaciones en mamografías, obteniendo un 97,8 % de Dice *score*.

4.2.3 V-Net

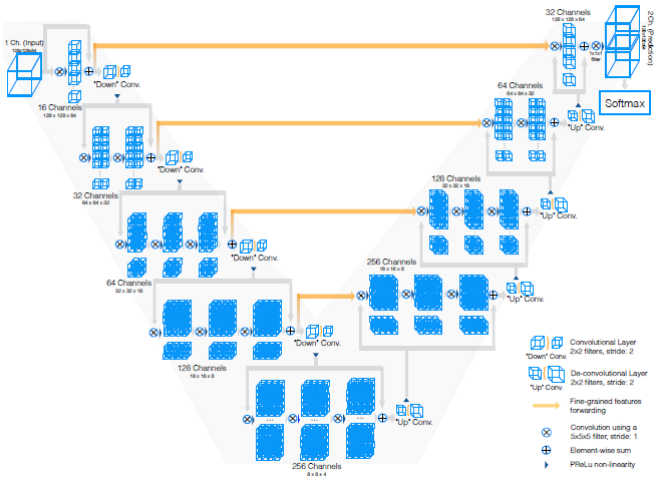


Figura 4.5: Arquitectura V-Net. Fuente: Milletari et al. [24]

V-Net es una arquitectura pensada para segmentación semántica de imágenes biomédicas 3D, como imágenes de resonancia magnética de próstata. Es una arquitectura muy parecida a U-Net, ya que ambas siguen la misma estructura. Sin embargo, también hay algunas diferencias entre ellas:

- En cada nivel se aprende una función residual para garantizar la convergencia. A diferencia de U-Net, la entrada de cada nivel se combina con la salida de la última capa convolucional del nivel.
- Se aplican convoluciones con filtros $5 \times 5 \times 5$, ya que estamos usando volúmenes. En esta ocasión, no se utilizan capas de *pooling*, sustituyéndolas por capas convolucionales con filtros $2 \times 2 \times 2$. La función de activación usada es PReLU.

- En la parte decodificadora se utiliza la operación de deconvolución —con filtros $2 \times 2 \times 2$— para aumentar el tamaño del volumen. Al igual que U-Net, se recupera parte de la información de la parte codificadora usando conexiones horizontales. También se aplica una convolución $1 \times 1 \times 1$ en la última capa convolucional para tener el mismo volumen que la entrada de la red. Finalmente, se utiliza una capa *softmax* para obtener la probabilidad de pertenecer a una determinada clase por cada vóxel.

4.2.4 CUMedVision1

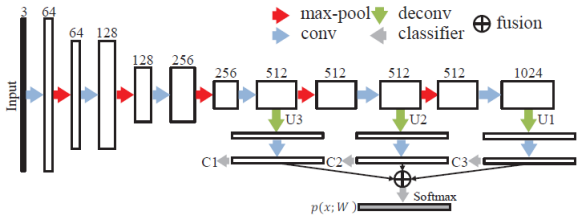


Figura 4.6: Arquitectura CUMedVision1. Fuente: Chen et al. [25]

CUMedVision1 es una arquitectura para realizar segmentación de glándulas. En primer lugar, se reduce el tamaño de la imagen de entrada a través de las capas de convolución y *pooling*. A continuación, en determinadas capas antes de aplicar *max pooling*, se realiza *upsampling* con capas deconvolucionales para recuperar detalles como los límites de cada glándula, y se obtienen los mapas de características C1, C2 y C3. Finalmente se fusionan los mapas anteriores y se aplica la función *softmax* para obtener el mapa de segmentación. Existe una versión mejorada de esta arquitectura, llamada CUMedVision2 (DCAN), que mejora la segmentación de los límites de las glándulas.

4.2.5 *Feature Pyramid Network* (FPN)

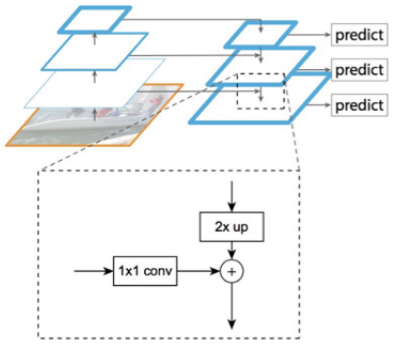


Figura 4.7: Arquitectura FPN. Fuente: Minaee et al. [19]

Feature Pyramid Network (FPN) es una de las arquitecturas más populares para segmentación. Se caracteriza por su estructura con forma de pirámide y está compuesta por dos caminos: de abajo hacia arriba y de arriba hacia abajo. En el camino de abajo hacia arriba se van extrayendo las características reduciendo la información espacial hasta llegar a la cúspide (*downsampling*). La información de cada una de las capas se conectará a las capas simétricas del camino de arriba hacia abajo mediante conexiones laterales.

A continuación, se realiza la fase de *upsampling*, aumentando la resolución espacial en un factor de 2 usando la técnica del vecino más cercano. Después, se mezcla la información procedente de las conexiones laterales con las capas del mismo tamaño espacial. En esta fase se aplican convoluciones de tamaño 1×1 para reducir las dimensiones de cada canal. Finalmente, se aplica una convolución 3×3 para generar la salida de cada nivel. Por último, en cada nivel se realiza la predicción para detectar un objeto presente en la imagen.

4.2.6 Mask R-CNN

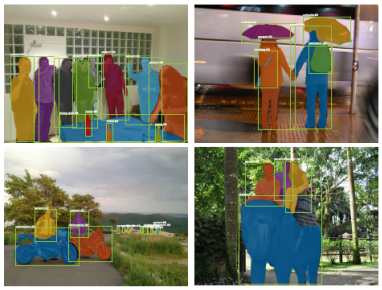


Figura 4.8: Resultados de Mask R-CNN sobre el *dataset* COCO. Fuente: He et al. [26]

Mask R-CNN es una de las arquitecturas más recientes del estado del arte para segmentación de instancias en imágenes o vídeo. Mask R-CNN se compone de dos fases. En primera de ellas se proponen regiones donde podría estar un determinado objeto. La segunda predice la clase del objeto, se establece el cuadro delimitador y se genera la máscara de segmentación para indicar los píxeles que pertenecen a la clase del objeto. En ambas fases se utiliza una estructura *backbone* tipo FPN; de esta forma, se puede implementar con cualquier arquitectura convolucional como VGG, ResNet o Inception.

4.3 Arquitecturas de redes neuronales para detección de objetos

4.3.1 YOLO

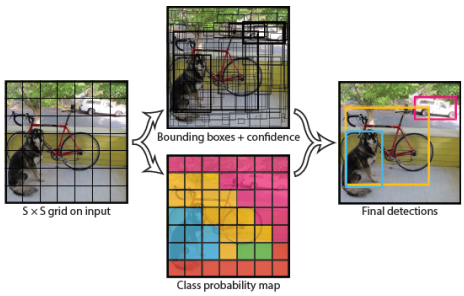


Figura 4.9: Modelo usado por YOLO. Fuente: Redmon et al. [27]

YOLO (*You Look Once*) es un sistema del estado del arte para detección de objetos en tiempo real. De esta forma, el punto fuerte de este sistema es su aplicación en vídeos. Este sistema es uno de los detectores de objetos en tiempo real más potentes, ya que solo es necesario pasar la imagen o el vídeo a la red una sola vez. A diferencia de otros detectores de objetos como R-CNN, YOLO utiliza una única red neuronal, y usa las características de la imagen para predecir múltiples cuadros delimitadores, cada uno con un objeto específico. Para ello, cada imagen se divide en celdas de tamaño $S \times S$. Cada una de las celdas predice B cuadros delimitadores, calculando el nivel de confianza de un solo objeto presente en la celda. Si la celda no contiene ningún objeto, el nivel de confianza será cero. En cambio, si la celda contiene un objeto, el nivel de confianza será igual a la IoU del cuadro delimitador predicho y la predicción real. Cada una de las puntuaciones de confianza se obtiene mediante una red convolucional, de tipo GoogLeNet.

Actualmente existen varias versiones de YOLO más rápidas y precisas, como YOLO V2 y YOLO V3. También existen implementaciones de este sistema en C como DarkNet [28]. Entre las aplicaciones de YOLO, podemos destacar su uso en el desarrollo de vehículos autónomos, detección de equipajes abandonados en lugares públicos [29] o detección de tráfico.

4.4 Experimentación

Como se ha podido observar en la sección anterior, existen múltiples arquitecturas de redes neuronales para segmentación. Sin embargo, no todas son adecuadas para el problema a resolver. A lo largo de esta sección, utilizaremos U-Net para aplicar segmentación semántica de células cancerosas, debido a los buenos resultados que presenta esta arquitectura en imágenes biomédicas.

4.4.1 Descripción del dataset

El *dataset* [30] utilizado en esta segunda parte del trabajo ha sido generado por los autores del artículo Naylor et al. [21]. El *dataset* está formado por 50 imágenes histológicas pertenecientes a 11 pacientes con cáncer de mama triple negativo. Las imágenes histológicas han sido obtenidas mediante microscopía virtual con un aumento de 40x en el Instituto Curie, uno de los mejores centros de investigación de cáncer del mundo.

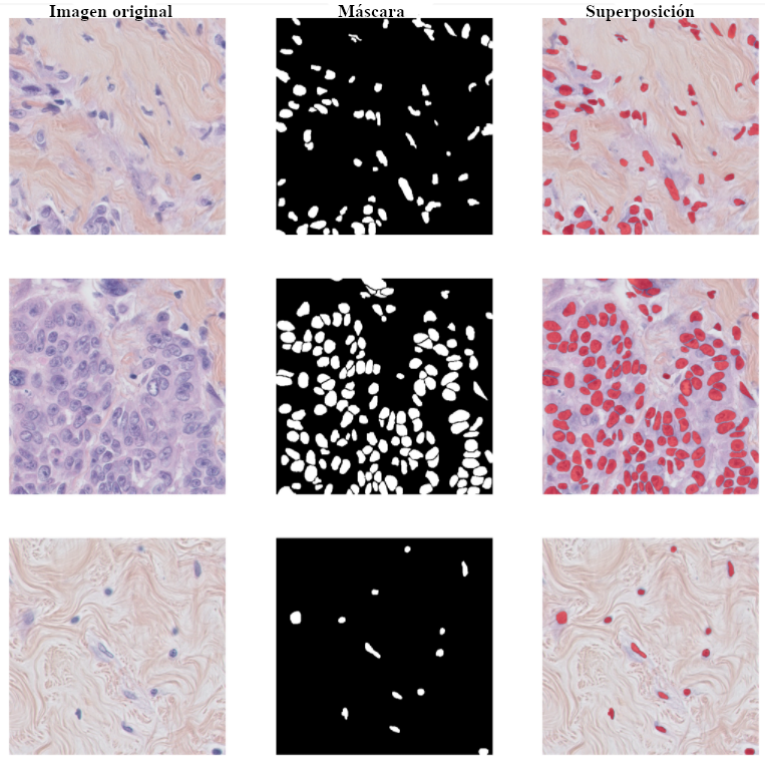


Figura 4.10: Comparativa de imágenes del *dataset*: a la izquierda, la imagen de entrada; en el centro, la máscara binaria, y a la derecha, la aplicación de la máscara en la imagen, realizando la detección de células.

Con respecto al análisis exploratorio, por cada una de las 11 pacientes, se han tomado desde tres hasta ocho trozos (*patches*) de la imagen histológica, con una resolución de 512×512 cada uno. Las imágenes han sido seleccionadas por tres expertos de forma heterogénea, es decir, incluyen zonas del tejido con poca presencia celular, y otras regiones donde hay una mayor cantidad de células cancerosas invasivas. De esta forma, la variabilidad de los datos es mucho mayor, ya que todas las pacientes tienen el mismo tipo de cáncer. Cada una de las imágenes tiene una máscara binaria asociada para indicar la localización de las células. En este problema existirán dos clases: fondo (*background*) y célula, por lo que estaremos ante un problema de segmentación binaria.

En cuanto a las células seleccionadas, todas son cancerosas. Sin embargo, se han seleccionado células de varios tipos: epiteliales y mioepiteliales, localizadas en los conductos de la leche materna; células invasivas, fibroblastos, endoteliales, adipocitos, macrófagos y células

inflamatorias. En total se han anotado 4093 células, donde el máximo número de células en un ejemplo es de 293 y el mínimo es 5; además, la media de células por imagen es de 80 y una desviación típica de 58. Debido la amplia variedad de tipos de células, sería interesante desarrollar un modelo de segmentación de células para estudiar diversas propiedades como el tamaño, el número de células etc., permitiendo a los especialistas médicos ofrecer un tratamiento u otro según el estado de la enfermedad.

4.4.2 Herramientas utilizadas

Además de las herramientas usadas en la primera parte de este trabajo, se usarán las siguientes para implementar las distintas arquitecturas:

- **Google Colab**: Es un servicio en la nube desarrollado por Google para el desarrollo de modelos de *machine learning* y *deep learning*. En dicho servicio se pueden crear entornos de Python de una forma rápida, con todas las librerías más usadas de *deep learning*, y con la posibilidad de utilizar las GPU de Google de forma totalmente gratuita. Se ha usado debido a los costes computacionales necesarios para entrenar la arquitectura U-Net.
- **Segmentation Models** [31]: Es una API de alto nivel para crear modelos de segmentación utilizando la librería Keras. La librería implementa 4 modelos de arquitecturas para segmentación (incluidas U-Net y FPN), varias funciones de error y las métricas más usadas en segmentación para la evaluación del modelo. Además, tiene la posibilidad de modificar las arquitecturas mediante *backbones*, utilizando las arquitecturas convolucionales más famosas y con los pesos de ImageNet.
- **Keras-unet** [32]: Se usa para generar las gráficas comparativas de las predicciones obtenidas por el modelo y las predicciones reales.

Todos los experimentos realizados en esta sección se han ejecutado en un entorno virtual de Python 3 en Google Colab. La GPU y la RAM del entorno pueden cambiar de una ejecución a otra, ya que se utilizan los recursos disponibles en ese momento. En este caso, las prestaciones para todos los experimentos son las siguientes:

GPU	Tesla P100-PCIE 16GB
RAM	12GB

4.4.3 Preprocesamiento

Al tener un *dataset* pequeño, no es necesario crear una base de datos HDF5 para la gestión eficiente de los datos. En este caso, las imágenes y las máscaras se han almacenado en dos archivos con formato NPY (NumPy *Data File*). El *dataset* se ha dividido de la siguiente forma: 30 imágenes para el conjunto de entrenamiento (60 %) y 10 imágenes en los conjuntos de validación y prueba (40 %). Se utilizarán técnicas de *data augmentation* en todos los experimentos debido al tamaño del *dataset*. También se ha utilizado el generador de Keras implementado para generar nuevos datos y realizar el preprocesamiento durante el entrenamiento. Con respecto a la normalización, se ha usado la función `get_preprocessing`

de la librería Segmentation Models para aplicar el preprocesamiento necesario dependiendo del *backbone*. Cada *backbone* representa a una arquitectura de redes convolucionales (p. ej. VGG16 o ResNet34); de esta forma, en la parte codificadora se aplica el *backbone* seleccionado y en la parte decodificadora se replica la arquitectura de forma simétrica. Por otro lado, cada una de las máscaras se normaliza dividiendo los píxeles por 255, para que el rango de valores sea $[0, 1]$, facilitando los cálculos de la red. Finalmente, se inicializa cada red con los pesos de ImageNet para aprovechar la transferencia de aprendizaje. Con respecto a la evaluación, los modelos se evalúan sobre dos conjuntos: el conjunto de prueba creado anteriormente y un nuevo conjunto aplicando *data augmentation* al conjunto de prueba anterior; de esta forma, obtenemos más ejemplos.

4.4.4 Análisis de resultados

4.4.4.1 Métricas de evaluación y función de error

Pixel accuracy

En el capítulo anterior usamos la precisión (*accuracy*) como medida principal para evaluar la calidad del modelo. Sin embargo, en segmentación esta medida no es fiable debido al desbalanceo de clases. Dado que en el *dataset* la mayor parte de los píxeles de la máscara pertenecen al fondo (*background*), los píxeles estarían desbalanceados con respecto a la clase «célula». De esta forma, si usamos una máscara completa de color negro, ganaríamos una precisión que no se correspondería con la realidad, ya que no estaríamos detectando ninguna célula. Además, no existe el mismo número de células en cada máscara, por lo que el desbalanceo aumenta aún más. Por este motivo, es necesario utilizar otras medidas de rendimiento más adecuadas ante el desbalanceo de clases que tienen la mayoría de los problemas de segmentación.

Intersección sobre la unión (IoU)

$$IoU = \frac{A \cap B}{A \cup B}$$

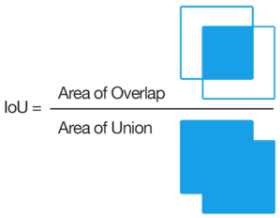


Figura 4.11: Cálculo de IoU sobre dos conjuntos. Fuente: Tiu [33]

La intersección sobre la unión, también llamada índice de Jaccard, es una de las métricas más usadas por su sencillez y fiabilidad. IoU relaciona el área de solapamiento entre la máscara predicha por el modelo y la predicción real, y el área de la unión de la región segmentada por el modelo y la región real. De esta forma, si la máscara que ha generado el modelo coincide exactamente con la máscara real $IoU = 1$, y valdría 0 si no coincide nada. En

general, los valores superiores a 0.5 obtienen buenos resultados. Para evaluar nuestro modelo de segmentación binaria, tenemos que calcular la media de los valores IoU de cada clase.

Coeficiente Dice (F1 *score*)

$$F1 = \frac{2 \cdot |A \cap B|}{|A| + |B|}$$

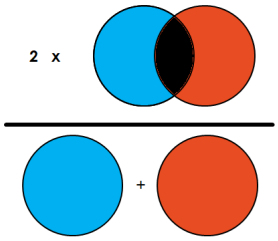


Figura 4.12: Cálculo del coeficiente de Dice sobre dos conjuntos. Fuente: Tiu [33]

El coeficiente de Dice o F1 *score* es otra de las medidas más utilizadas. Es muy similar a IoU: se multiplica por dos el área de la intersección y se divide por la suma del área de las dos máscaras. La métrica también devuelve valores comprendidos en el intervalo $[0, 1]$, midiendo la similitud entre las dos máscaras.

Función de error: Jaccard *loss*

$$L(A, B) = 1 - \frac{A \cap B}{A \cup B}$$

La función de error anterior se utilizará en la implementación de las arquitecturas, combinada con la función de entropía cruzada, y está basada en la métrica IoU. De esta forma, si la máscara de salida no coincide, el ajuste de los pesos será mayor; y si coincide exactamente, no habrá ningún ajuste.

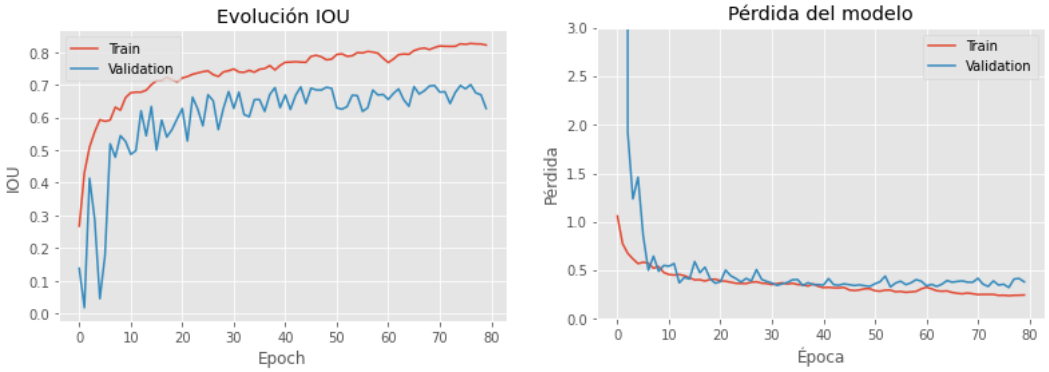
4.4.4.2 U-Net

Para la implementación de U-Net se ha usado la arquitectura implementada en Segmentation Models. Se han realizado tres experimentos con diferentes *backbones*: *resnet34*, *vgg16* y *inceptionv3*.

En todos los experimentos se ha utilizado un tamaño de *batch* igual a 2 debido a la cantidad de datos disponibles; y la red se ha entrenado durante 80 épocas. El optimizador usado es Adam con $\alpha = 0.0001$. En cuanto a las técnicas de *data augmentation*, se ha aumentado el conjunto de entrenamiento a 60 ejemplos, volteando las imágenes horizontal y verticalmente, realizando *zoom* del 50 % y girando las imágenes 90º. Para el conjunto de prueba aumentado, se han generado 400 imágenes aplicando las transformaciones anteriores.

	Pérdida	Media IoU	Media F1 Score
Conjunto de prueba	0.50	0.66	0.79
Conjunto de prueba aumentado	0.65	0.66	0.80

Tabla 4.1: Resultados de los conjuntos de prueba usando *resnet34*.



(a) Evolución IoU **(b)** Evolución del error

Figura 4.13: Gráficas de evolución

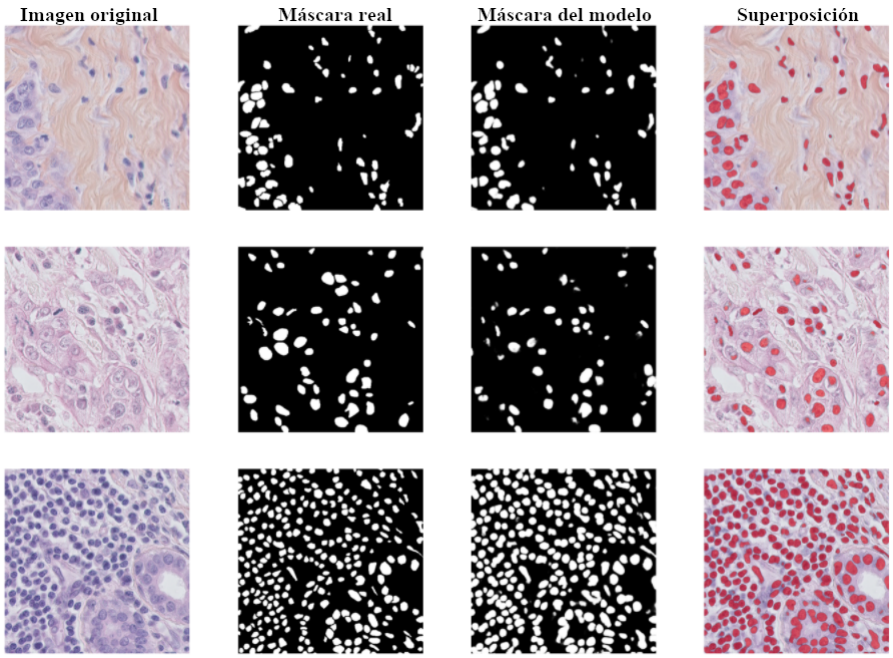


Figura 4.14: Comparativa de máscaras de segmentación en el conjunto de prueba, utilizando ResNet.

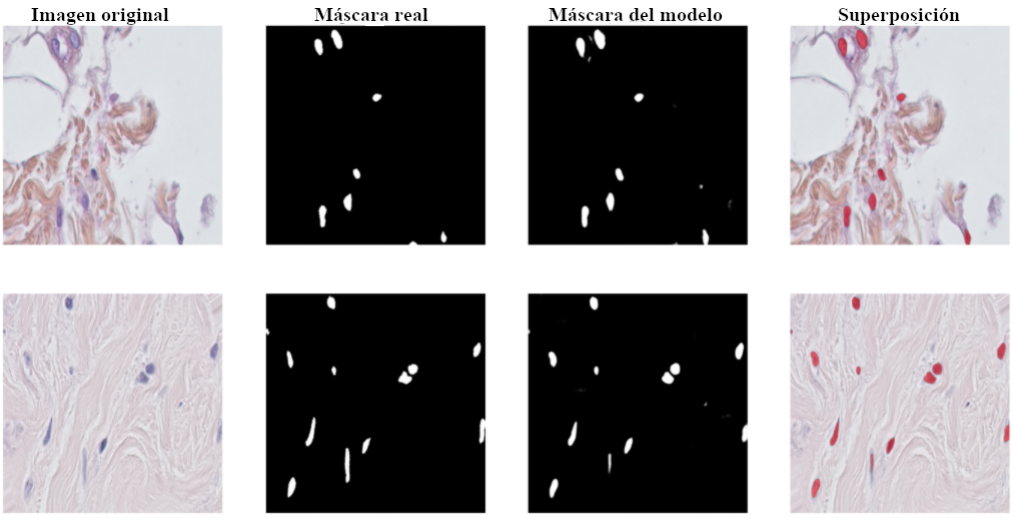
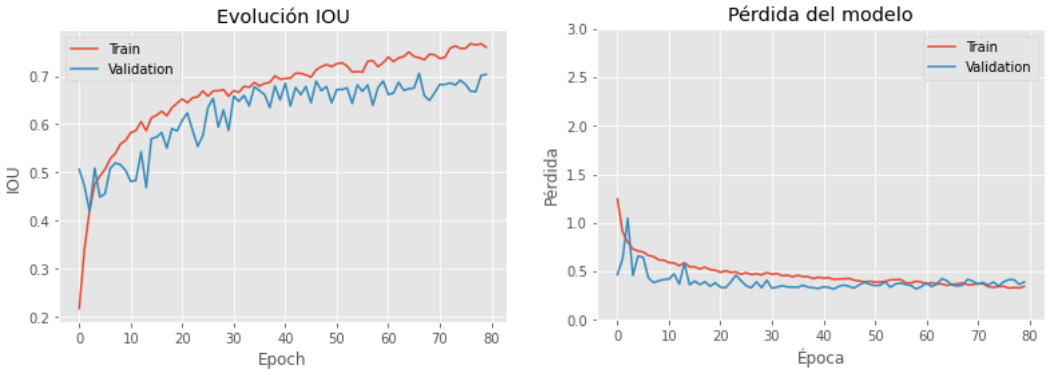


Figura 4.15: Comparativa de máscaras de segmentación en el conjunto de prueba aumentado, utilizando ResNet.

En la **Tabla 4.1** podemos ver las métricas de evaluación sobre los conjuntos de prueba. Podemos observar que el modelo obtiene resultados bastante aceptables, llegando a obtener 0.66 de media IoU. Esto último lo podemos comprobar observando las **Figuras 4.14 y 4.15** , donde vemos que las máscaras generadas por el modelo indican la localización de las células de forma muy parecida a las máscaras reales. El modelo consigue detectar la forma de cada célula, aunque no de manera exacta. Por otro lado, si observamos las gráficas de evolución del entrenamiento (**Figura 4.13**), vemos que el entrenamiento se realiza correctamente, disminuyendo el error de validación a medida que aumenta el número de épocas. Al principio existe un error de entrenamiento y validación alto, debido a los pesos iniciales de ImageNet. En cuanto a la evolución de la IoU, existe un ligero sobreajuste, ya que las curvas de entrenamiento y validación divergen un poco. Sin embargo, a la hora de evaluar el modelo sobre el conjunto de prueba aumentado, los resultados siguen siendo buenos, ya que se obtiene la misma media IoU y la forma de cada célula se detecta correctamente.

	Pérdida	**Media IoU**	**Media F1 Score**
Conjunto de prueba	0.47	0.66	0.79
Conjunto de prueba aumentado	0.67	0.63	0.77

Tabla 4.2: Resultados de los conjuntos de prueba usando *vgg16*.



(a) Evolución IoU **(b)** Evolución del error

Figura 4.16: Gráficas de evolución

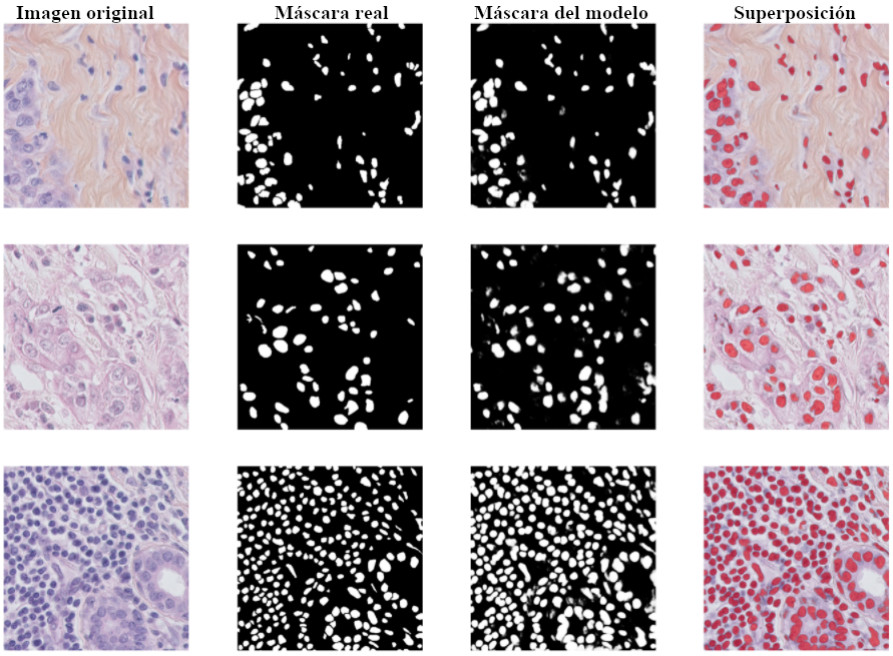


Figura 4.17: Comparativa de máscaras de segmentación en el conjunto de prueba, utilizando VGG16.

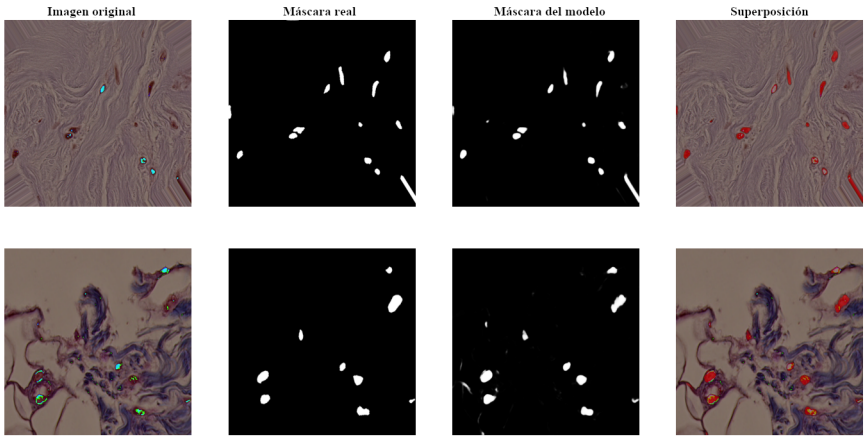
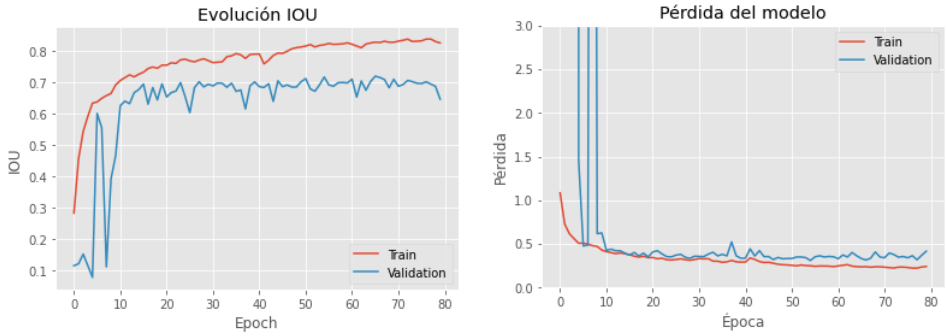


Figura 4.18: Comparativa de máscaras de segmentación en el conjunto de prueba aumentado, utilizando VGG16.

El *backbone* VGG16 obtiene resultados muy similares a ResNet (véase la **Tabla 4.2**). En este caso, vemos que la media IoU es igual, aunque la media del error baja hasta 0.47 en el conjunto de prueba. Si nos fijamos en la **Figura 4.16**, vemos que existe muy poco sobreajuste, ya que la curva de error de validación es muy cercana a la de entrenamiento. En cuanto a la evolución de la IoU, los valores de validación siguen siendo más bajos que los de entrenamiento, empezando a divergir en la época 40. Finalmente, observando la comparativa de las máscaras, podemos ver que se siguen detectando las células correctamente, ya que las máscaras generadas son muy parecidas a las reales; aunque en este caso, el valor de la media IoU en el conjunto de prueba aumentado baja con respecto al *backbone resnet34*.

	Pérdida	**Media IoU**	**Media F1 Score**
Conjunto de prueba	0.46	0.68	0.81
Conjunto de prueba aumentado	0.58	0.67	0.80

Tabla 4.3: Resultados de los conjuntos de prueba usando *inceptionv3*.



(a) Evolución IoU **(b)** Evolución del error

Figura 4.19: Gráficas de evolución

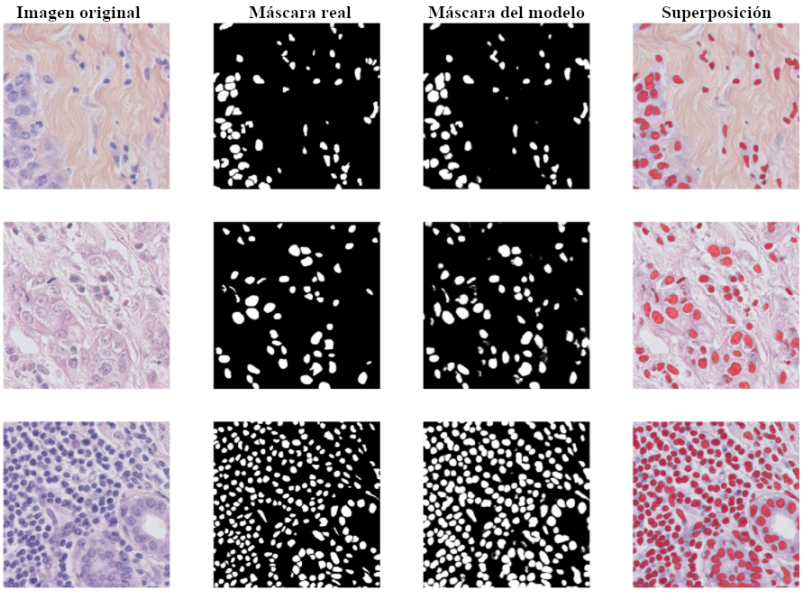


Figura 4.20: Comparativa de máscaras de segmentación en el conjunto de prueba, utilizando InceptionV3.

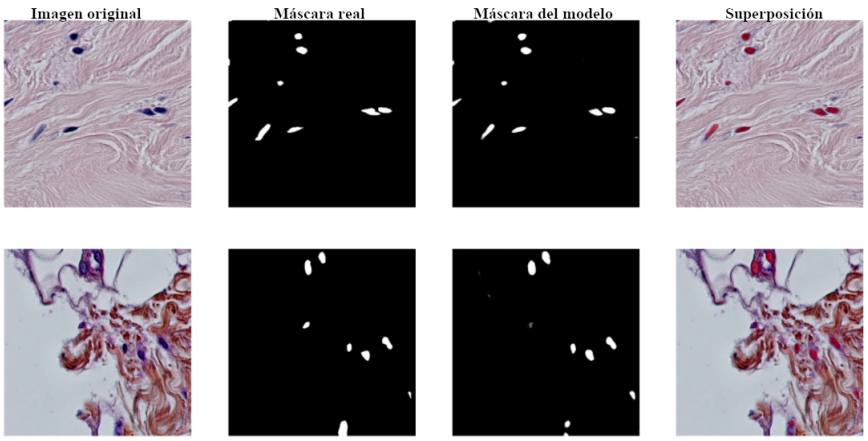


Figura 4.21: Comparativa de máscaras de segmentación en el conjunto de prueba aumentado, utilizando InceptionV3.

Por último, si usamos el *backbone inceptionv3*, obtenemos los mejores resultados (**Tabla 4.3**). En este caso, la media IoU sobre el conjunto de prueba aumenta hasta 0.68, siendo mayor que los dos *backbones anteriores*. Observando las gráficas de evolución (**Figura 4.19**), experimentamos un fuerte descenso del error de validación al principio del entrenamiento, debido a los pesos de ImageNet. Por otro lado, en la gráfica de evolución de IoU, vemos un poco de sobreajuste al ser la curva de validación más baja que la de entrenamiento. Finalmente, si aplicamos transformaciones aleatorias en el conjunto de prueba, los resultados siguen siendo aceptables, llegando a obtener 0.67 de IoU.

4.4.4.3 Conclusiones

Backbone	**IoU (Jaccard score)**	**F1 score (Dice score)**
ResNet34	0.66	0.79
VGG16	0.66	0.79
Inception V3	**0.68**	**0.81**

Tabla 4.4: Resumen de resultados sobre el conjunto de prueba.

Como podemos ver, obtenemos buenos resultados en cualquiera de los tres experimentos, siendo el *backbone inceptionv3* el que mejor resultados obtiene. Cada modelo logra generar un mapa de segmentación que clasifica los píxeles de las células de forma muy parecida a las predicciones reales. Por otro lado, vemos que U-Net es una arquitectura sencilla que logra obtener buenos resultados con pocos datos de entrada, a diferencia de la mayoría de arquitecturas convolucionales. Para este problema también se han realizado experimentos con otras arquitecturas implementadas en Segmentation Models, como FNP. Sin embargo, los resultados eran muy pobres —con IoU menor de 0.1— en comparación con U-Net.

5 Conclusiones

A lo largo de este trabajo hemos visto como el *deep learning* facilita el diagnóstico de enfermedades como el cáncer de mama. En primer lugar, se realiza un trabajo de documentación para explicar los fundamentos básicos del *deep learning*. Esto me ha permitido aprender bastantes cosas de un campo del que apenas sabía nada hasta ahora, y que me interesaba desde que comencé el grado.

A la hora de desarrollar cada uno de los modelos, he podido aprender las fases de desarrollo de un proyecto de *deep learning*, siendo la fase de preprocesamiento la más compleja. En este caso, la preparación y el procesamiento de los datos para la red es la parte que más me ha costado, ya que de ello depende la eficiencia computacional y la calidad de los resultados del modelo. Gracias al análisis exploratorio he podido darme cuenta del problema que supone el desbalanceo de clases, provocando confusión sobre el rendimiento real del modelo. También he tenido la oportunidad de usar las herramientas más utilizadas en el mundo laboral, haciendo que mis posibilidades laborales aumenten en el futuro.

En cuanto a las técnicas, he podido comprobar que existen multitud de formas para poder desarrollar un modelo con éxito, siendo la transferencia de aprendizaje y *data augmentation* dos de las más útiles. Con las dos técnicas anteriores, he podido mejorar bastante los resultados y reducir el sobreajuste. Otro aspecto a destacar es la elección de la arquitectura para un determinado problema. No es lo mismo usar una arquitectura que otra, ya que depende de la estructura, profundidad, el tipo de capas y los hiperparámetros que se utilizan. Esto último lo hemos podido comprobar en el capítulo 3, con las diferencias de resultados entre las arquitecturas, y en el capítulo 4, donde la arquitectura más adecuada para segmentar imágenes médicas es U-Net. La segmentación es otro de los retos más interesantes y actuales del *deep learning*, ya que no basta con clasificar una imagen completa, sino de localizar el objeto que nos interesa dentro de la imagen. En este sentido, los casos de uso van más allá del diagnóstico de enfermedades, como la detección de rostro, reconocimiento de iris o su aplicación en sistemas de videovigilancia.

Con los resultados obtenidos hemos visto como el *deep learning* supera a las técnicas clásicas, como la extracción manual de características. Dichas características tendrían que ser seleccionadas por un experto, haciendo el desarrollo del modelo más complejo, mientras que con el *deep learning* no hace falta. Con ello, se ha obtenido un modelo de clasificación con un 87 % de *accuracy*, y un modelo de segmentación con un 0.68 de IoU. Unos resultados bastante aceptables. No obstante, todo se puede mejorar, y se pueden hacer más experimentos como trabajos futuros, que se explicarán en la siguiente sección.

Este trabajo no ha estado exento de problemas. Los más destacados están relacionados con la carga y el preprocesamiento de los datos. Al principio, no encontraba la forma de procesar la gran cantidad de datos en memoria y aplicar a continuación el preprocesamiento de los datos para lograr un entrenamiento eficiente. Los entrenamientos, por tanto, eran muy lentos. Además, me di cuenta de que estaba usando solo la CPU de mi equipo. Para solucionarlo, reduje el número de ejemplos del *dataset* y creé una base de datos HDF5. También instalé la versión GPU de Tensorflow —reconozco también que la instalación fue compleja— . Posteriormente, conocí la existencia de Google Colab, una herramienta que soluciona los problemas anteriores del entorno de desarrollo. También ha habido problemas a la hora de entrenar modelos en algunas arquitecturas, como es el caso de InceptionV3 y FNP, dando malos resultados. El resto de fases del desarrollo han sido sencillas de implementar, ya que las librerías ofrecen una interfaz de alto nivel que facilita mucho el proceso.

Por último, me gustaría añadir que he disfrutado bastante con los conocimientos que he adquirido durante la elaboración de este trabajo, a pesar de los problemas. Esto me anima a seguir adelante para profundizar más en este campo y a realizar otros proyectos futuros. Este documento también puede verse como una especie de guía para ver qué es el *deep learning*, sus técnicas más importantes y una aplicación práctica sobre uno de los problemas más actuales. De esta forma, cualquier lector interesado puede sentirse libre de consultarlo.

Trabajos futuros

Con respecto a los trabajos futuros, existen varias posibilidades. Al principio se tenía pensado realizar segmentación semántica en el *dataset* CBIS-DDSM [34] de mamografías para detectar microcalcificaciones. Sin embargo, a la hora de aplicar la arquitectura U-Net, los resultados no tuvieron éxito. Esto último puede deberse a varias razones, como las características presentes en la imagen. Si comparamos las imágenes de CBIS-DDSM con el *dataset* TNBC, vemos que están en escala de grises, y el tamaño de los píxeles es mucho menor. Por otro lado, las células del *dataset* TNBC son más sencillas de segmentar debido a su forma y color. Para intentar resolver este problema, se podrían aplicar las propuestas de Hossain [22] y Ahmed et al. [35]. La idea básica consiste en aplicar técnicas de preprocesamiento más avanzadas como reducción de ruido y usar la arquitectura Mask-RCNN. Por otro lado, se podría segmentar la región donde se sitúa el cáncer y luego clasificar las microcalcificaciones de forma precisa, usando técnicas de agrupamiento difuso como el algoritmo *Fuzzy C-Means*, clasificándolas en ejemplos positivos y negativos. Posteriormente, se entrena en una arquitectura U-Net modificada. También se podría entrenar una red neuronal no convolucional para obtener características de cada píxel, como el contraste, luminosidad y entropía.

Por otro lado, se pueden mejorar los modelos desarrollados, realizando nuevos experimentos con arquitecturas del estado del arte. En el caso del problema de clasificación de IDC, se pueden usar métodos combinados, que combinan varios modelos con el objetivo de aumentar el rendimiento general. Para el problema de segmentación de células cancerosas, se puede experimentar con redes generativas adversarias (GAN).

Con los modelos implementados en este trabajo, se puede desarrollar un *software* de diagnóstico asistido por ordenador —en inglés *Computer aided diagnosis* (CAD)—. Dicho *software* será una aplicación web donde se subirá la imagen histológica de la paciente, y se realizarán las fases de preprocesamiento y clasificación automáticamente, mostrando las zonas donde se sitúa el cáncer. Además, se podrá realizar segmentación de las células cancerosas y, si el modelo de segmentación de microcalcificaciones tiene éxito, añadirlo también. De esta forma, en una única plataforma, se podría diagnosticar cáncer de mama en distintos tipos de imágenes. También sería interesante añadir una sección para pacientes, dando acceso a los informes realizados por los médicos. Al ser una aplicación web, también se podría adaptar para dispositivos móviles y aumentar el número de plataformas.

Bibliografía

[1] "Cáncer de mama." [Online]. Available: https://www.aecc.es/es/todo-sobre-cancer/tipos-cancer/cancer-mama

[2] A. Cruz-Roa, A. Basavanhally, F. González, H. Gilmore, M. Feldman, S. Ganesan, N. Shih, J. Tomaszewski, and A. Madabhushi, "Automatic detection of invasive ductal carcinoma in whole slide images with convolutional neural networks," M. N. Gurcan and A. Madabhushi, Eds., San Diego, California, USA, Mar. 2014, p. 904103. [Online]. Available: http://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.2043872

[3] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," *arXiv:1603.07285 [cs, stat]*, Jan. 2018, arXiv: 1603.07285. [Online]. Available: http://arxiv.org/abs/1603.07285

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[5] F. A. Spanhol, L. S. Oliveira, C. Petitjean, and L. Heutte, "Breast cancer histopathological image classification using Convolutional Neural Networks," in *2016 International Joint Conference on Neural Networks (IJCNN)*, Jul. 2016, pp. 2560–2567, iSSN: 2161-4407.

[6] B. Wei, Z. Han, X. He, and Y. Yin, "Deep learning model based breast cancer histopathological image classification," in *2017 IEEE 2nd International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*, Apr. 2017, pp. 348–353.

[7] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv:1409.1556 [cs]*, Apr. 2015, arXiv: 1409.1556. [Online]. Available: http://arxiv.org/abs/1409.1556

[8] S. Khan, N. Islam, Z. Jan, I. Ud Din, and J. J. P. C. Rodrigues, "A novel deep learning based framework for the detection and classification of breast cancer using transfer learning," *Pattern Recognition Letters*, vol. 125, pp. 1–6, Jul. 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167865519301059

[9] "Classification of Breast Cancer using Deep Learning Architecture," *International Journal of Recent Technology and Engineering*, vol. 8, no. 4, pp. 7451–7454, Nov. 2019. [Online]. Available: https://www.ijrte.org/wp-content/uploads/papers/v8i4/D5317118419.pdf

[10] E. Deniz, A. Şengür, Z. Kadiroğlu, Y. Guo, V. Bajaj, and m. Budak, "Transfer learning based histopathologic image classification for breast cancer detection," *Health Information Science and Systems*, vol. 6, no. 1, p. 18, Sep. 2018. [Online]. Available: https://doi.org/10.1007/s13755-018-0057-x

[11] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper with Convolutions," *arXiv:1409.4842 [cs]*, Sep. 2014, arXiv: 1409.4842. [Online]. Available: http://arxiv.org/abs/1409.4842

[12] J. Chang, J. Yu, T. Han, H.-j. Chang, and E. Park, "A method for classifying medical images using transfer learning: A pilot study on histopathology of breast cancer," in *2017 IEEE 19th International Conference on e-Health Networking, Applications and Services (Healthcom)*, Oct. 2017, pp. 1–4.

[13] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, Jun. 2016, pp. 2818–2826. [Online]. Available: http://ieeexplore.ieee.org/document/7780677/

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *arXiv:1512.03385 [cs]*, Dec. 2015, arXiv: 1512.03385. [Online]. Available: http://arxiv.org/abs/1512.03385

[15] A. Jerónimo Fuentes, "ajf97/breast_histopathology_cancer." [Online]. Available: https://github.com/ajf97/breast_histopathology_cancer

[16] "Breast Histopathology Images." [Online]. Available: https://kaggle.com/paultimothymooney/breast-histopathology-images

[17] F.-F. Li, J. Johnson, and S. Yeung, "Lecture 11: Detection and Segmentation," p. 95.

[18] J. Long, E. Shelhamer, and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," *arXiv:1411.4038 [cs]*, Mar. 2015, arXiv: 1411.4038. [Online]. Available: http://arxiv.org/abs/1411.4038

[19] S. Minaee, Y. Boykov, F. Porikli, A. Plaza, N. Kehtarnavaz, and D. Terzopoulos, "Image Segmentation Using Deep Learning: A Survey," *arXiv:2001.05566 [cs]*, Apr. 2020, arXiv: 2001.05566. [Online]. Available: http://arxiv.org/abs/2001.05566

[20] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," *arXiv:1505.04597 [cs]*, May 2015, arXiv: 1505.04597. [Online]. Available: http://arxiv.org/abs/1505.04597

[21] P. Naylor, M. Laé, F. Reyal, and T. Walter, "Segmentation of Nuclei in Histopathology Images by Deep Regression of the Distance Map," *IEEE Transactions on Medical Imaging*, vol. 38, no. 2, pp. 448–459, Feb. 2019, conference Name: IEEE Transactions on Medical Imaging.

[22] M. S. Hossain, "Microc alcification Segmentation Using Modified U-net Segmentation Network from Mammogram Images," *Journal of King Saud University - Computer and Information Sciences*, Nov. 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1319157819307347

[23] F. Long, "Microscopy cell nuclei segmentation with enhanced U-Net," *BMC Bioinformatics*, vol. 21, no. 1, p. 8, Jan. 2020. [Online]. Available: https://doi.org/10.1186/s12859-019-3332-1

[24] F. Milletari, N. Navab, and S.-A. Ahmadi, "V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation," *arXiv:1606.04797 [cs]*, Jun. 2016, arXiv: 1606.04797. [Online]. Available: http://arxiv.org/abs/1606.04797

[25] H. Chen, X. Qi, L. Yu, and P.-A. Heng, "DCAN: Deep Contour-Aware Networks for Accurate Gland Segmentation," *arXiv:1604.02677 [cs]*, Apr. 2016, arXiv: 1604.02677 version: 1. [Online]. Available: http://arxiv.org/abs/1604.02677

[26] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," *arXiv:1703.06870 [cs]*, Jan. 2018, arXiv: 1703.06870. [Online]. Available: http://arxiv.org/abs/1703.06870

[27] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," *arXiv:1506.02640 [cs]*, May 2016, arXiv: 1506.02640. [Online]. Available: http://arxiv.org/abs/1506.02640

[28] "Darknet: Open Source Neural Networks in C." [Online]. Available: https://pjreddie.com/darknet/

[29] T. Santad, P. Silapasupphakornwong, W. Choensawat, and K. Sookhanaphibarn, "Application of YOLO Deep Learning Model for Real Time Abandoned Baggage Detection," in *2018 IEEE 7th Global Conference on Consumer Electronics (GCCE)*, Oct. 2018, pp. 157–158, iSSN: 2378-8143.

[30] N. P. Jack, W. Thomas, L. Marick, and R. Fabien, "Segmentation of Nuclei in Histopathology Images by deep regression of the distance map," Feb. 2018, type: dataset. [Online]. Available: https://zenodo.org/record/2579118#.X4bmweZxeiP

[31] P. Yakubovskiy, "qubvel/segmentation_models," Oct. 2020, original-date: 2018-06-05T13:27:56Z. [Online]. Available: https://github.com/qubvel/segmentation_models

[32] K. Żak, "karolzak/keras-unet," Oct. 2020, original-date: 2019-03-13T07:50:21Z. [Online]. Available: https://github.com/karolzak/keras-unet

[33] E. Tiu, "Metrics to Evaluate your Semantic Segmentation Model," Oct. 2020. [Online]. Available: https://towardsdatascience.com/metrics-to-evaluate-your-semantic-segmentation-model-6bcb99639aa2

[34] R. S. Lee, F. Gimenez, A. Hoogi, K. K. Miyake, M. Gorovoy, and D. L. Rubin, "A curated mammography data set for use in computer-aided detection and diagnosis research," *Scientific Data*, vol. 4, no. 1, p. 170177, Dec. 2017, number: 1 Publisher: Nature Publishing Group. [Online]. Available: https://www.nature.com/articles/sdata2017177

[35] L. Ahmed, M. M. Iqbal, H. Aldabbas, S. Khalid, Y. Saleem, and S. Saeed, "Images data practices for Semantic Segmentation of Breast Cancer using Deep Neural Network," *Journal of Ambient Intelligence and Humanized Computing*, Jan. 2020. [Online]. Available: https://doi.org/10.1007/s12652-020-01680-1

[36] F. Berzal, *Redes Neuronales & Deep Learning*, Nov. 2018.

[37] P. L. Arancibia Hernández, T. Taub Estrada, A. López Pizarro, M. L. Díaz Cisternas, and C. Sáez Tapia, "Calcificaciones mamarias: descripción y clasificación según la 5.a edición BI-RADS," *Revista Chilena de Radiología*, vol. 22, no. 2, pp. 80–91, Apr. 2016. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0717201X16300288

[38] "Python Data Science Handbook | Python Data Science Handbook." [Online]. Available: https://jakevdp.github.io/PythonDataScienceHandbook/

[39] "Cáncer de seno triple negativo." [Online]. Available: https://www.cancer.org/es/cancer/cancer-de-seno/comprension-de-un-diagnostico-de-cancer-de-seno/tipos-de-cancer-de-seno/triple-negativo.html

[40] CDCespanol, "Cáncer de mama triple negativo," Sep. 2020. [Online]. Available: https://www.cdc.gov/spanish/cancer/breast/triple-negative.htm

[41] "OpenCV: OpenCV modules." [Online]. Available: https://docs.opencv.org/4.4.0/

[42] "Overview — NumPy v1.19 Manual." [Online]. Available: https://numpy.org/doc/1.19/

[43] "scikit-learn: machine learning in Python — scikit-learn 0.23.2 documentation." [Online]. Available: https://scikit-learn.org/stable/

[44] J. M. R. Plens, "Plantilla TFG-TFM_eps," Oct. 2020, original-date: 2018-01-21T03:43:50Z. [Online]. Available: https://github.com/jmrplens/TFG-TFM_EPS