UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADO EN INGENIERÍA DE LA SALUD

Clasificación de imágenes histopatológicas de cáncer de mama utilizando técnicas de aprendizaje profundo

Histopathological image classification of breast cancer using deep learning techniques

Realizado por
Paula Andújar Zambrano

Tutorizado por
Rafael Marcos Luque Baena
Miguel Ángel Molina Cabello

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, SEPTIEMBRE DE
2022

Fecha defensa:

Resumen

En la actualidad, el uso de algoritmos clasificadores para el diagnóstico del cáncer es una práctica extendida en la rama de la oncología. Existen una gran cantidad de algoritmos y modelos que ayudan a los profesionales de la salud a determinar este tipo de patologías en los pacientes.

En el caso de la clasificación del cáncer por imágenes, los algoritmos desarrollados para ello son las redes neuronales convolucionales, de los que han ido surgiendo variantes con diferentes características como el número de capas de la red, el tipo o el orden en el que estas se disponen.

En este proyecto se estudiarán y aplicarán diferentes tipos de algoritmos de redes neuronales convolucionales ya implementados que nos permitirán clasificar imágenes histopatológicas del conjunto de BreaKHis. Con estos modelos construidos podremos escoger el que proporcione una mayor precisión para, posteriormente, implementar una aplicación web que permita a los usuarios un fácil uso de este clasificador.

Palabras clave: Aprendizaje profundo, Cáncer de mama, Modelo predictivo, Imágenes histopatológicas.

Índice

Índice de figuras

1

Introducción

El cáncer de mama en mujeres es el segundo tipo de cáncer más común que conocemos, después del cáncer de piel. Es bastante invasivo y tiene una alta tasa de mortalidad. Por este motivo, su detección temprana es fundamental para la supervivencia y el tratamiento de las personas que lo padecen. [1, 5]

Hoy en día se utilizan técnicas como mamografías o imágenes por ultrasonidos para la detección y el control del cáncer de mama. Sin embargo, estas técnicas no analizan los tumores a nivel celular. Para esta tarea es necesario realizar una biopsia sobre el tejido afectado y evaluar las imágenes histopatológicas resultantes en busca de alguna variación anómala de las células. [5]

Las imágenes histopatológicas tienen una característica importante que las convierte en las más confiables para el reconocimiento de cualquier tipo de cáncer. Estas tienen una resolución notablemente mayor que otros tipos de imágenes biológicas, contando con un tamaño de varios giga-píxeles. Esta característica permite a las imágenes revelar la estructura detallada de los tejidos a nivel microscópico. El inconveniente de estas imágenes es el análisis de las mismas ya que es una tarea tediosa para los patólogos y susceptible al error humano.

Por esta razón, el aprendizaje profundo (Deep Learning) está ganando importancia en el campo del análisis de imágenes histopatológicas. Con el conjunto de técnicas adecuado podemos predecir diagnósticos mejorando la objetividad y eficiencia del análisis. Además, esta predicción tendría en cuenta un conjunto de registros anteriores, por lo que en la mayoría de los casos aumentaría la precisión de los diagnósticos.

El uso de redes neuronales convolucionales en el aprendizaje profundo ha conseguido un gran éxito en los problemas de clasificación de imágenes. Estas redes consisten en un conjunto

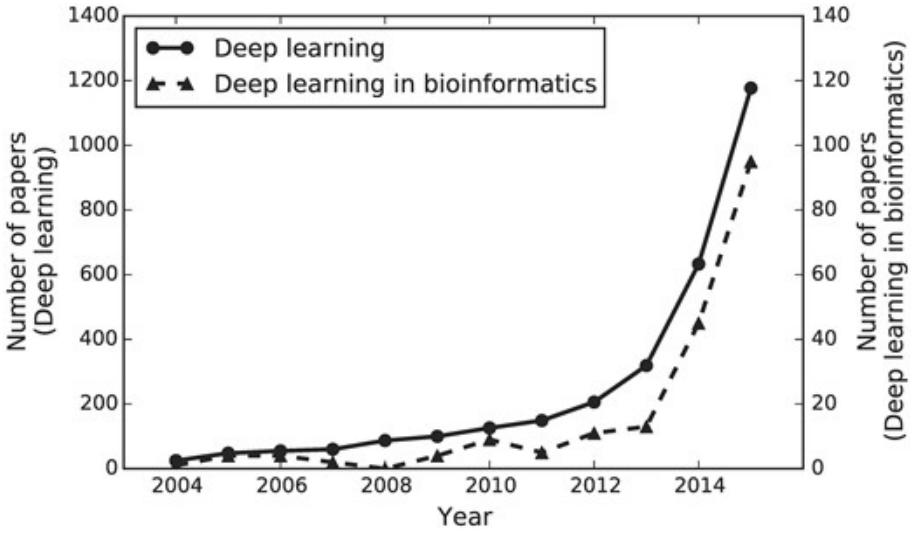


Figura 1: Evolución del Deep Learning [8]

de etapas apiladas que se van entrenando y se encuentran posicionadas entre clasificadores de tipo supervisado los cuales se representan con un mapa y cada celda es una característica extraída de la entrada. [2]

1.1. Motivación

Actualmente, el aprendizaje profundo está creciendo a gran velocidad en numerosas ramas del conocimiento, una de ellas es la medicina. En el sector sanitario el uso de técnicas computacionales ha contribuido a poder explotar cantidades de datos masivos para ayudar a la calidad de los diagnósticos, hacer nuevas estadísticas o procesar imágenes de alta resolución. Todas estas tareas que resultaban difíciles y tediosas de ejecutar ahora son rápidas y se realizan sin errores. Por ejemplo, se ha mejorado la clasificación de enfermedades a partir de datos clínicos de entrada, sobre la cual vamos a desarrollar el trabajo. Específicamente, nos centraremos en la clasificación de cáncer de mama mediante el entrenamiento de un algoritmo de redes neuronales convolucionales. [6]

En muchos casos, la detección temprana del cáncer de mama ha hecho posible la supervivencia de los pacientes gracias al tratamiento inmediato del tumor. Es por esto que el reto de este trabajo es clasificar automáticamente las imágenes con buena precisión y así poder actuar lo antes posible aumentando la probabilidad de supervivencia de los pacientes afectados.

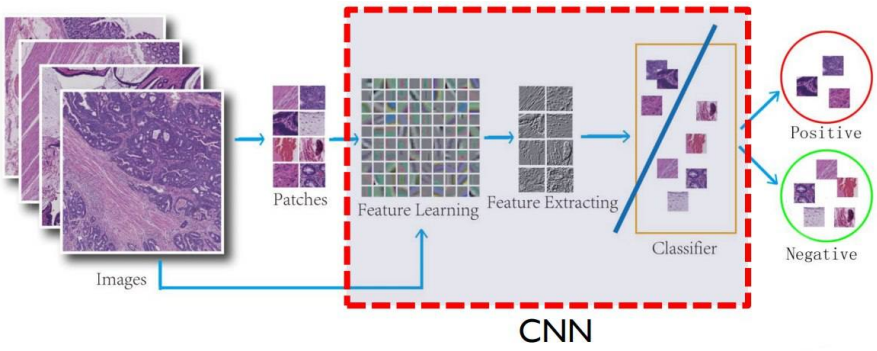


Figura 2: Esquema de predicción mediante Deep Learning [7]

El conjunto de imágenes que se va a utilizar de entrada (BreakHis) esta compuesto por 9109 imágenes histopatológicas de tejido tumoral mamario recolectadas de 82 pacientes utilizando diferentes factores de aumento, este trabajo se centrará exclusivamente en las imágenes de 40 aumentos.

1.2. Objetivos

El objetivo principal es el estudio y desarrollo de un modelo predictor basado en redes neuronales convolucionales que esté entrenado para clasificar imágenes histopatológicas de cáncer de mama. Para ello, se probarán distintos algoritmos de aprendizaje profundo ya diseñados con el fin de conseguir el mejor rendimiento del modelo.

Para simplificar el uso del modelo, también se diseñará una aplicación web de recogida de datos y visualización de los resultados de la clasificación que funcionará mediante una API entre la interfaz gráfica y el algoritmo entrenado.

Por tanto, el desglose de objetivos que hay que alcanzar en este trabajo son:

1. Estudio previo del aprendizaje profundo: Se introducirán los conceptos básicos generales y específicos de las redes neuronales convolucionales.

2. Diseño del algoritmo: Se implementarán los algoritmos con distintos modelos de neuronas y se ajustarán los mejores parámetros.

3. Discusión de los resultados: Se procederá a elegir el algoritmo con mejor resultado clasificador y se descartarán los demás.

4. Construcción de la aplicación de recogida de datos: Se diseñará la interfaz de usuario con la que se podrá hacer uso del mejor modelo entrenado.

2

Estado del arte

2.1. Aprendizaje automático o Machine learning

El aprendizaje automático es una aplicación de la inteligencia artificial en la que los programas informáticos utilizan algoritmos para encontrar patrones en los datos mediante la experiencia, sin estar programados explícitamente para ello.

Con estos algoritmos se logra mejorar el rendimiento de los programas y hacer predicciones precisas de los datos, lo cual mejora la toma de decisiones. En resumen, el aprendizaje automático sirve para clasificar datos, encontrar patrones, proyectar resultados y tomar decisiones.

En cuanto al sistema de aprendizaje de estos algoritmos, se podría clasificar en tres etapas principales [9]:

- **Proceso de decisión:** Es la primera etapa del aprendizaje y del entrenamiento del modelo, en ella el algoritmo genera una estimación inicial del patrón de los datos en base a los datos de entrada.

- **Función de error:** La función de error o de pérdida nos permite evaluar la precisión del algoritmo. Esta función no es única para cualquier algoritmo de aprendizaje, dependiendo del problema a tratar será adecuado usar un tipo de función u otra. Por ejemplo, para problemas de regresión se puede usar el error cuadrático medio mientras que para problemas de clasificación sería más acertado usar la pérdida o función logarítmica.

- **Proceso de optimización:** En esta etapa los parámetros del algoritmo se ajustan mejor al patrón de los datos. Esto se realiza iterativamente ajustando los pesos de las neuronas hasta que la función de pérdida alcance el umbral de precisión deseado.

Los algoritmos también se pueden clasificar en función del modelo de aprendizaje que utilicen, de esta forma un modelo podría ser: **supervisado, no supervisado o de refuerzo.**

- **Aprendizaje supervisado:** El algoritmo aprende mediante ejemplos, se le proporciona datos de entrada y salida, en la cual la salida está etiquetada con el valor correcto.
- **Aprendizaje no supervisado:** No se le proporciona valores de salida, el algoritmo debe encontrar un patrón en función de la estructura de los datos de entrada.
- **Aprendizaje por refuerzo:** En este tipo de aprendizaje no se proporcionan datos de salida etiquetados, sino que a medida que el algoritmo se entrena se proveen señales de si el sistema está funcionando correctamente o no.

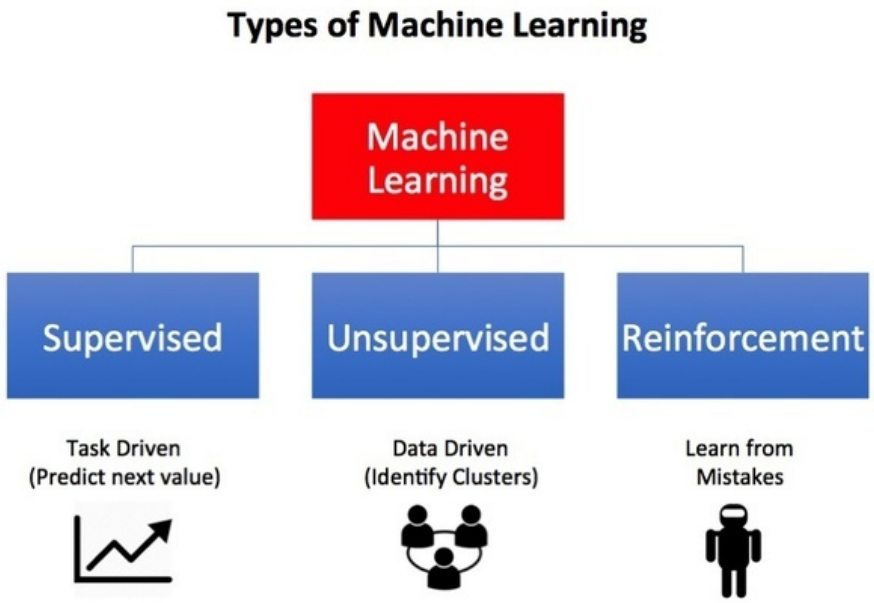


Figura 3: Tipos de aprendizaje automático [11]

2.1.1. Aprendizaje profundo o Deep learning

El aprendizaje profundo es un subcampo del aprendizaje automático que se centra en crear amplios modelos de redes neuronales capaces de tomar decisiones precisas basadas en datos.

Las redes neuronales artificiales son modelos computacionales basados esencialmente en el funcionamiento del sistema nervioso biológico (como el cerebro humano). Se componen de una gran cantidad de nodos computacionales (basados en las neuronas) interconectados entre

sí, que aprenden conjuntamente de la entrada para optimizar al máximo la salida. Podemos observar la estructura de neuronas en la Figura 4.

De esta forma, el proceso de entrenamiento del algoritmo sería cargar los datos a la capa de entrada, generalmente en forma de un vector multidimensional, que se distribuirá a las capas ocultas. Estas capas internas permiten aprender representaciones de datos con múltiples niveles de abstracción, con las que podemos encontrar patrones en grandes conjuntos de datos mediante el uso del algoritmo de propagación hacia atrás, el cual indica los parámetros internos que se deben ir modificando para calcular la representación en cada capa anterior. Esto se conoce como proceso de aprendizaje. Tener varias capas ocultas apiladas una sobre otra es comúnmente llamado aprendizaje profundo.

Se han desarrollado diferentes tipos de algoritmos de aprendizaje profundo que utilizan las redes neuronales artificiales, dentro de los más comunes podemos encontrar las **redes neuronales recurrentes**, las **redes generativas antagónicas** o las **redes neuronales convolucionales**, el último mencionado es sobre el que basaremos este estudio.

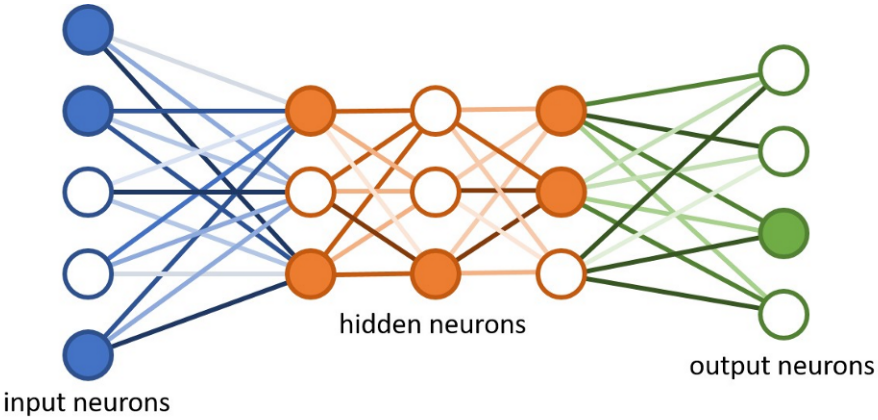


Figura 4: Estructura de capas o neuronas [12]

2.1.2. Redes neuronales convolucionales

Las redes neuronales convolucionales o *CNN* son un tipo de redes neuronales artificiales que se usan en el campo del reconocimiento de patrones en imágenes. Los datos de entrada en este caso son imágenes, y por tanto, la arquitectura del algoritmo está configurada de la manera que mejor se adapte a este tipo específico de datos.

Un algoritmo de CNN coge una imagen de entrada, le asigna valores a varios aspectos de la imagen según su importancia y la clasifica en base a las clases que se hayan definido con anterioridad.

Las imágenes en el campo de las redes neuronales se representa mediante una matriz de vectores equivalentes a los píxeles de la misma. Si las imágenes escalas de grises solo necesitaríamos dos canales para representar los píxeles de ella, en cambio, si la imagen es a color necesitaríamos tres canales (rojo verde y azul).

La arquitectura de una CNN se compone de una capa de entrada, varias capas ocultas y una capa final de salida. Dentro de las capas ocultas, las primeras pueden detectar líneas y curvas, y se van especializando hasta llegar a las capas más profundas que pueden llegar a reconocer formas complejas.

Las capas ocultas de una red neuronal convolucional pueden ser de varios tipos, dependiendo de la acción que realizan:

- **Capa convolucional**. La capa convolucional realiza la operación matemática de convolución sobre la imagen. Esta capa aplica filtros aprendidos a las imágenes que van creando un mapa de características que resume la presencia de esas características en la entrada. Estos filtros suelen ser de menor dimensión que la imagen pero de igual profundidad. A la matriz resultante de este proceso se le denomina **mapa de activación**.

  Estas capas suelen tener varios filtros ya que cada uno reconoce una característica diferente. Cuantos más filtros se utilicen, más características se extraerán de la imagen y por tanto, el patrón que reconozca será mejor.

  La capa de convolución suele venir con una **unidad lineal rectificada** (comúnmente abreviada como ReLu), que tiene como objetivo aplicar una función elemental de activación, como la función sigmoide, a la salida de la activación producida por la capa anterior.

- **Capa de agrupación**. La capa de agrupación o *pooling* realiza una reducción de la dimensión de los mapas de activación, es decir, reduce la cantidad de parámetros a tener en cuenta, conservando información de mayor importancia. Está agrupación puede hacerse de varias formas, definiendo el máximo, la media o la suma.

- **Capa totalmente conectada** En esta capa, como su nombre indica, cada neurona de la capa anterior está conectada a todas las neuronas de la capa. En esta capa neurona aplica una transformación lineal al vector de entrada a través de una matriz de peso. El propósito de esta capa es usar las características obtenidas en las capas anteriores para clasificar la imagen a la clase a la que pertenezca.

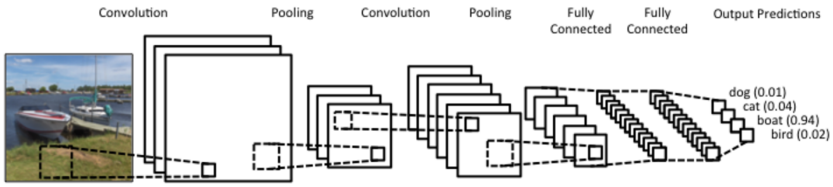


Figura 5: Arquitectura de una red neuronal convolucional [23].

2.2. Conceptos clave

2.2.1. Entrenamiento, validación y prueba

A la hora de entrenar un modelo de aprendizaje profundo es necesario dividir el conjunto total de datos en tres subconjuntos: **entrenamiento, validación y prueba.**

- **Conjunto de datos de entrenamiento:** Este subconjunto se utiliza para la construcción inicial de los modelos en base a los patrones que se extraigan de los datos de entrada. El entrenamiento se realiza mediante un método de aprendizaje supervisado, como la optimización mediante el descenso del gradiente, y finalmente, se seleccionan uno o varios modelos finales.
- **Conjunto de datos de validación:** Con este conjunto de datos se realiza la validación del modelo elegido en el entrenamiento, ajustando así los hiperparámetros de este para aumentar su precisión. Una vez terminada esta etapa, el modelo de validación sería el modelo seleccionado definitivo.
- **Conjunto de datos de prueba:** Los datos de prueba se utilizan para realizar pruebas reales del modelo y así determinar su precisión y error real. Si la precisión del modelo con los datos de prueba difiere bastante de la precisión con los datos de entrenamiento es que el modelo está sobreajustado y no da buenos resultados.

No existe una sola forma adecuada de dividir estos subconjuntos. Sin embargo, por lo general el subconjunto más grande suele ser el de entrenamiento para poder entrenar el modelo inicial con datos suficientes. Los otros dos subconjuntos se suelen dividir a partes iguales con el resto de los datos, una división común en el aprendizaje profundo es 60 %-20 %-20 % (entrenamiento, validación y prueba respectivamente).

2.2.2. Sobreajuste

El *overfitting* o sobreajuste en un modelo se da cuando se han utilizado demasiados datos y variables para el entrenamiento, de tal forma que al introducir datos reales el modelo no es capaz de generalizar. En otras palabras, éste solo clasifica de forma correcta los ejemplos con los que ha sido entrenado. Por tanto, el rendimiento de un modelo sobreajustado es mucho peor con los datos de prueba que con los de entrenamiento. En la figura 5 podemos observar como se ajustaría este tipo de modelo a una serie de ejemplos dados en el entrenamiento.

Algunas soluciones para evitar el sobreajuste de un modelo podrían ser aportar nuevos datos para el entrenamiento, eliminar variables que no aportan información significativa de los datos o aplicar técnicas de aumento de datos y de validación cruzada.

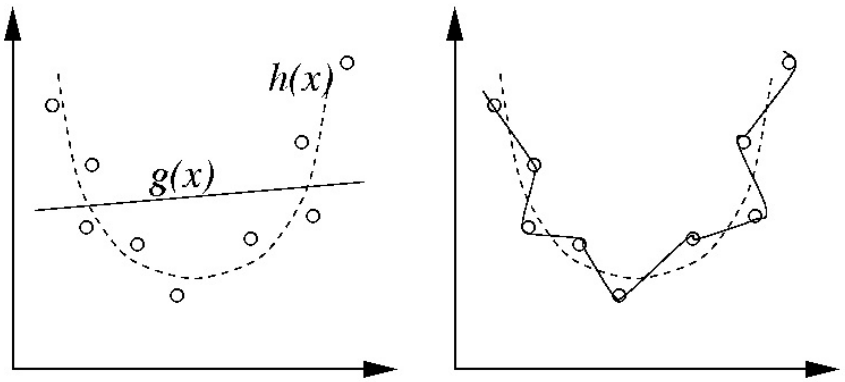


Figura 6: Modelo bien ajustado en la izquierda y sobreajustado en la derecha [13].

2.2.3. Tasa aprendizaje

La tasa de aprendizaje o *learning rate* es un hiperparámetro que controla cuanto se ajustan los pesos de la red con respecto al gradiente de la función de pérdida en el proceso de entrenamiento.

Una tasa de aprendizaje demasiado alta hará que el aprendizaje no se atasque en mínimos locales pero podría tener oscilaciones tan grandes que nunca llegue a alcanzar el punto óptimo o incluso divergir. En cambio, una tasa de aprendizaje demasiado baja recorrerá todos los mínimos locales lentamente pero tardará demasiado en converger o se atascará en un mínimo local indeseable. En la Figura 6 podemos observar el comportamiento de diferentes tasas de aprendizaje enfrentando la pérdida frente a la época de entrenamiento.

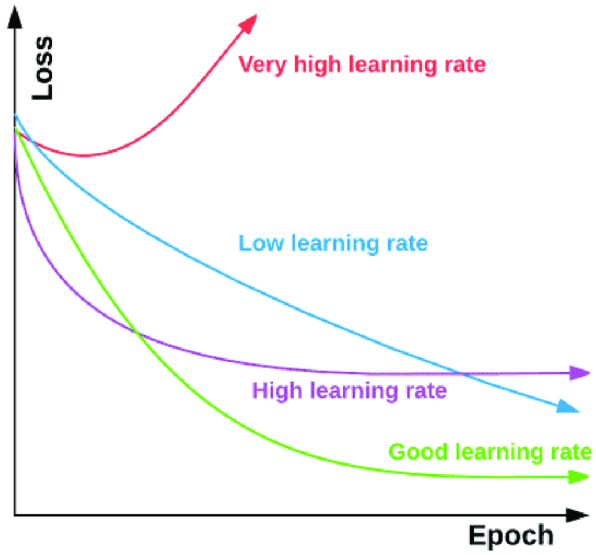


Figura 7: Diferentes tasas de aprendizaje [14]

2.2.4. Época de entrenamiento

La época o *epoch* es un hiperparámetro que indica el número de veces que se realizará un recorrido de entrenamiento completo por la red (propagación hacia delante y hacia atrás). En cada época, todos los datos pasan por la red.

Además, podemos especificar el *batch size*, con el que tendremos iteraciones internas del algoritmo en cada época, este parámetro indica en cuantos lotes se dividirán los datos totales dentro de la época, actualizando un mayor número de veces los pesos de la red.

2.2.5. Propagación hacia atrás

El algoritmo de propagación hacia atrás o *backpropagation* es una red neuronal que emplea un ciclo de propagación y adaptación de dos fases.

En la primera fase, los datos de entrada llegan a las primeras neuronas y se propagan hacia las neuronas de la capa oculta llegando hasta la última capa, generando una salida. Esta salida es comparada con la salida deseada, calculando una señal de error en función de la diferencia entre estas.

En la segunda fase, la señal de error se propaga hacia atrás partiendo desde la salida. Esta señal de error llega a las neuronas de la capa oculta, pero la intensidad de la señal que recibe cada neurona va en función de la contribución de esta a la salida original, por lo que de esta forma las neuronas puede recalcular sus pesos en función del error que han obtenido en la época anterior.

Este proceso se repite hasta que el error que se obtiene llega al umbral máximo de error establecido.

3

Materiales y métodos

3.1. Introducción

Una vez analizado el estado de arte con los conceptos clave que abordan este trabajo se procederá explicar los materiales y métodos utilizados para el desarrollo del algoritmo de redes neuronales.

En cuanto a los materiales usados se ha utilizado la plataforma de **Google Colab** como servicio cloud para el desarrollo. Por otra parte la librería de aprendizaje profundo utilizada para definir los métodos necesarios del algoritmo es **Pytorch**.

En el desarrollo del algoritmo se han utilizado varios métodos o técnicas como la división de imágenes en **patches**, la técnica de **early stopping** en la fase de entrenamiento o la **validación cruzada** para medir el rendimiento de los modelos y garantizar la independencia de los datos entre las fases de entrenamiento y prueba.

También cabe destacar que en este proyecto se ha seguido una metodología científica que consta de una serie de principios:

1. **Método científico**: Es el principal método que se seguirá durante todo el proyecto. Gracias a este método podemos adquirir nuevos conocimientos con los que establecer nuevas leyes fundamentales. Este método debe seguir una serie de objetivos para garantizar los resultados. Se basa en la observación empírica, la razón y la demostración. Este es necesario para la validación de los resultados obtenidos. Estos pasos son:
    - Observación y reconocimiento de un problema: Gracias a la observación crítica

de nuestro entorno podemos hacernos preguntas que indiquen la existencia de un problema sin solución.

- Investigación sobre el problema: Un aspecto muy importante es el de recopilar toda la información posible acerca del problema observado.

- Formulación de hipótesis: Mediante la hipótesis podemos plantear una posible explicación al problema que hemos observado.

- Experimentación: Se desarrollan varias pruebas o experimentos para poder determinar la validez de la hipótesis planteada.

- Comunicación de los resultados: Finalmente, un paso importante es la comunicación de los resultados obtenidos, de forma escrita o audiovisual.

2. **Metodología iterativa e incremental**: En este proyecto, la forma de trabajo óptima es mediante la división de tareas en iteraciones. Así obtenemos una fácil administración del tiempo y nos permite realizar cambios sin influir en el resto de la programación.

3. **Metodología de implementación**: Para la implementación del programa implementado se consideran una serie de características básicas, tales como la modularidad, comprensibilidad, modificabilidad, extensibilidad, sencillez, documentación y el cuidado del estilo de programación.

3.1.1. Pytorch

Pytorch es una librería de python de código abierto que ofrece herramientas para el desarrollo de redes neuronales artificiales. Este fue desarrollado por Facebook en 2017 y ha sido una pieza fundamental en el campo de la inteligencia artificial por la sencillez de su interfaz y su capacidad para ejecutarse en GPUs.

Por las ventajas comentadas, hemos elegido este framework para crear nuestro modelo de aprendizaje profundo.

3.1.2. Patches

Las imágenes de gigapíxeles son demasiado grandes para caber en una GPU a la vez. Por lo general, se dividen en *patches* más pequeños para entrenar el modelo de aprendizaje profundo.

La manera más simple de dividir los *patches* de las imágenes es al azar. Durante cada iteración k de entrenamiento, se selecciona un subconjunto aleatorio diferente de *patches*. Si la cantidad y el tamaño de estos son lo suficientemente pequeños, caben en la GPU.

Con las imágenes disponibles en nuestro conjunto de datos, en concreto, se ha utilizado la librería *Patchify*, con la que se han dividido las imágenes en patches solapados de 225 x 225 píxeles. Por cada imagen se han obtenido 15 *patches*, por lo que el conjunto de datos ha aumentado 15 veces.

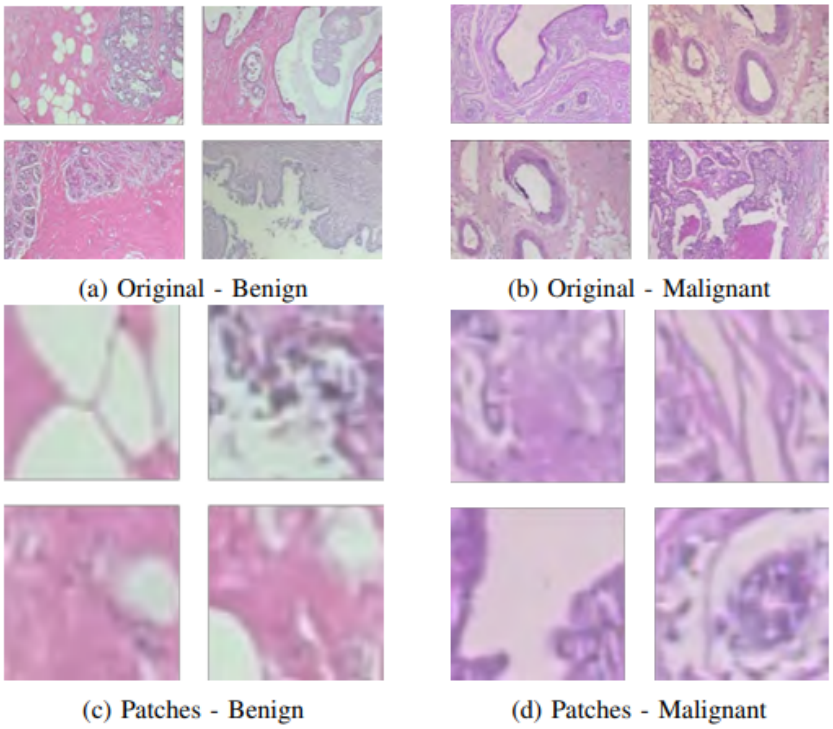


Figura 8: División de imágenes en patches [1]

3.1.3. Early Stopping

El *early stopping* o detención temprana es una técnica usada en el entrenamiento de un modelo predictivo.

Cuando se entrena una red grande, hay un punto en el entrenamiento en el que el modelo deja de generalizar y aprende del ruido de los datos de entrada. Este sobreajuste hace que el modelo disminuya su capacidad de predicción y por tanto este será menos útil.

Por ello, un enfoque para resolver este problema es detener el entrenamiento si el rendimiento del modelo comienza a degradarse, es decir, cuando la precisión disminuye o la pérdida aumenta.

3.1.4. Validación cruzada

La validación cruzada es una técnica que nos permite probar el rendimiento de un modelo predictivo. Esta consiste en dividir los datos de forma aleatoria en k grupos del mismo tamaño, k-1 grupos se emplean para el entrenamiento del modelo y el restante como validación. El proceso se repite k veces generando k estimaciones del error. La estimación final se calcula como el promedio de esas estimaciones.

Con esta técnica garantizamos que los resultados sean independientes de la partición entre los datos de entrenamiento y prueba. Un ejemplo de validación cruzada podemos observarlo en la Figura 9, en el que el conjunto de datos se divide en cuatro partes iguales y en cada iteración o plegamiento el subconjunto de test es diferente.

Para nuestro algoritmo, hemos utilizado la **validación cruzada estratificada**, que se diferencia de la primera en que la división de los datos se hace teniendo en cuenta el balanceo de las etiquetas de clase.

3.2. Algoritmo implementado

La finalidad del algoritmo implementado es la construcción de un modelo de redes neuronales convolucionales a partir de estructuras de red preexistentes del paquete de Pytorch. Esto se ha conseguido mediante una serie de pasos.

Primero, se han **instalado e importado las librerías necesarias** y a continuación, se lleva a cabo un **preprocesamiento de los datos**, consistente en los siguientes pasos:

1. Lectura y guardado de las imágenes en *arrays*

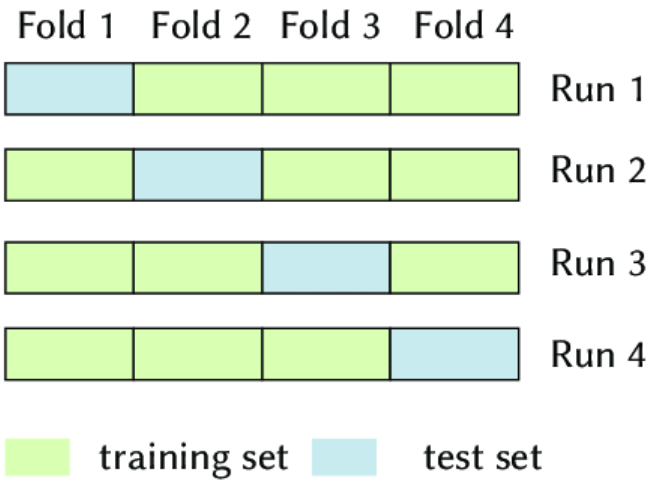


Figura 9: Diagrama del proceso de la validación cruzada.

2. Clasificación de las imágenes por sus etiquetas

3. División de las imágenes en subconjunto de entrenamiento y prueba

4. Creación de *patches* en los dos subconjuntos

5. División del conjunto de entrenamiento en entrenamiento y validación

Una vez terminado el preprocesamiento, ya tenemos los datos preparados para entrenar el algoritmo. En la primera ejecución, se han guardado los datos preprocesados resultantes de las iteraciones de la validación cruzada ya que este proceso es computacionalmente costoso y de esta forma se reduce en espacio y tiempo de ejecución.

Seguidamente, se declaran los parámetros que se van a usar, como el número de épocas, la estructura de la red, el optimizador o la tasa de aprendizaje.

En el **proceso de entrenamiento**, se definirá un bucle por cada iteración k en el que se entrenará el modelo y se guardarán la precisión y la pérdida obtenida, tanto en entrenamiento como en validación.

Finalmente, una vez obtenido los resultados, se **probará el modelo** con el conjunto de test que nos indicará los resultados definitivos.

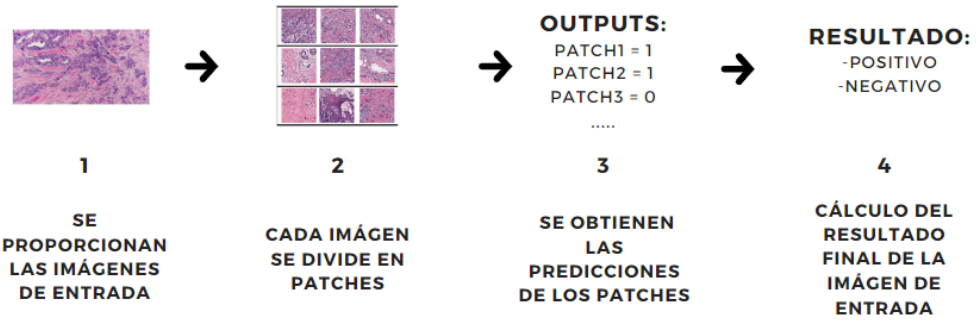


Figura 10: Diagrama de flujo del proceso de clasificación de una imagen.

3.3. Redes utilizadas

3.3.1. AlexNet

AlexNet [16] es el nombre dado a una red neuronal convolucional que ganó la competición LSVRC (*Large Scale Visual Recognition Challenge*) en 2012.

Específicamente, AlexNet se compone de cinco capas convolucionales: la primera capa, la segunda capa, la tercera capa y la cuarta capa seguida de la capa *pooling*, y la quinta capa seguida de tres capas completamente conectadas o *fully connected*.

La razón por la que AlexNet tiene éxito se puede atribuir a algunos de las estrategias prácticas, por ejemplo, la capa de no linealidad ReLU y la técnica de regularización de la dilución (o *dropout*.

ReLU es una función rectificadora de media onda, que puede acelerar significativamente la fase de entrenamiento y evitar el sobreajuste.

La técnica de dilución puede considerarse como una especie de regularización en la que se establece a cero un número de neuronas de entrada o neuronas ocultas para reducir las coadaptaciones de las neuronas. Esta técnica generalmente se utiliza en las capas completamente conectadas de AlexNet. [17]

3.3.2. ResNet

Las redes neuronales residuales o ResNet permiten la construcción de redes complejas y eficientes utilizando atajos o conexiones de salto para moverse entre capas.

Esta arquitectura permite saltar sobre capas no contiguas, atajando dos o tres capas intermedias que contienen normalización por lotes y no linealidad entre ellas. Estos saltos producen **bloques residuales**, cuya finalidad es evitar el problema del desvanecimiento de los gradientes, reutilizando las activaciones de una capa anterior hasta que la capa adyacente aprenda su peso. En la Figura 11, la ruta de la derecha es la residual y la ruta de la izquierda es el atajo o salto.

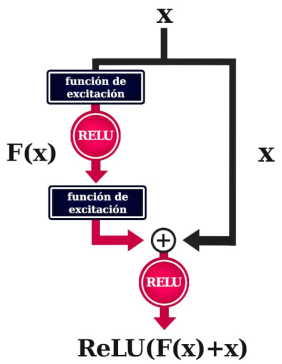


Figura 11: Bloque residual de la red neuronal ResNet. [15]

Este tipo de redes fue propuesto por *Kaiming et al.* en 2015, ganando la competición *Imagenet* en el mismo año.

Este algoritmo cuenta con varias versiones que dependen del número de capas usadas, los más usadas y conocidas son **ResNet152, ResNet56 o ResNet18**. Esta última mencionada es la que utilizaremos en este trabajo.

3.3.3. ShuffleNet V2

ShuffleNet V2 es una red neuronal convolucional que, a diferencia de otras, considera métricas directas como la velocidad o el coste de memoria para medir la complejidad computacional de la red.

Actualmente la mayoría de las arquitecturas de redes neuronales se guían principalmente por la métrica indirecta de la complejidad del cálculo llamada FLOPs (Floating Point Opera-

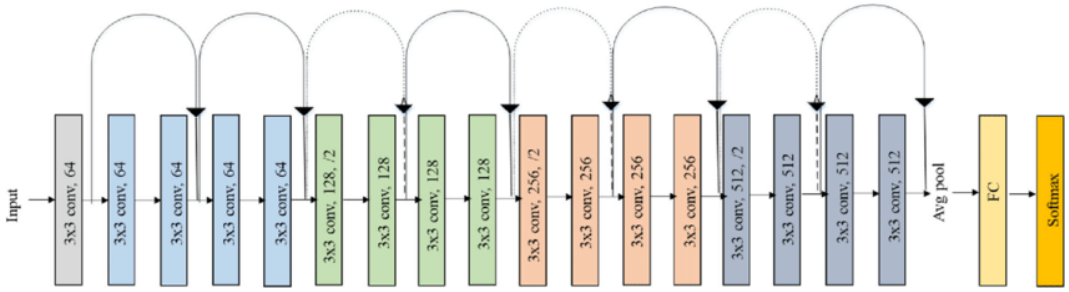


Figura 12: Arquitectura ResNet18. [22]

tions per Second). Está métrica indirecta puede darnos valores similares de redes teniendo tiempos de ejecución muy diferentes como, por ejemplo, MobileNet V2 y NasNet, que tienen FLOPs similares pero la primera es mucho más rápida. Por lo tanto, el uso de FLOPs podría conducir a diseños poco óptimos.

Otros factores que pueden influir a error con la métrica de FLOPs puede ser el grado de paralelismo o las características de la plataforma de destino.

Los autores de la arquitectura de la arquitectura de ShuffleNet V2 han proporcionado unas pautas para la construcción de la red considerando métricas directas:

1. El mismo ancho de canal minimiza el costo de acceso a la memoria: cuando el número de canales de entrada y salida se encuentra en la misma proporción el costo de acceso memoria disminuye.

2. La convolución de grupo excesiva aumenta el costo de acceso a la memoria: el número de grupo debe elegirse cuidadosamente en función de la plataforma y la tarea de destino.

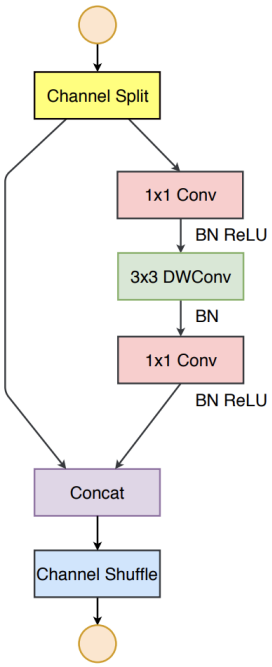
3. La fragmentación de la red reduce el grado de paralelismo: fragmentar la red reduce la eficiencia en la ejecución de cálculos paralelos.

4. Las operaciones por elementos no son despreciables: aunque tengan FLOPs pequeños pueden aumentar el tiempo de acceso a la memoria

Todas estas pautas están implementadas en la arquitectura de ShuffleNet V2.

En la Figura 13 se puede observar que la unidad básica de este tipo de red utiliza un operador simple llamado **divisor de canal**. El objetivo de la división de canales es que las arquitecturas alternativas, donde se utilizan convoluciones de grupos puntuales y estructuras de cuello de

botella, conducen a un mayor costo de acceso a la memoria. La división de canales es una alternativa en la que podemos mantener una gran cantidad de canales igualmente anchos sin tener convoluciones densas o demasiados grupos.



(c)

Figura 13: Unidad básica de ShuffleNet V2. [21]

3.3.4. SqueezeNet

La red neuronal convolucional SqueezeNet, como su propio nombre indica, es una red comprimida. El objetivo de su compresión es poder ser implementada en sistemas integrados con recursos limitados como dispositivos de memoria.

Esta red tiene 50 veces menos parámetros que Alexnet y su rendimiento es bastante similar. El número de cálculos, los requisitos de memoria y los tiempos de inferencia del modelo son también más bajos.

El modelo cuenta con un total de 18 capas (Figura 14), distribuidas en capas de convolución, módulos de fuego y capas de agrupación. La parte especial de SqueezeNet es el **módulo de fuego** (Figura 14.

Este modulo está formado por una capa de **convolución comprimida**, que tiene filtros

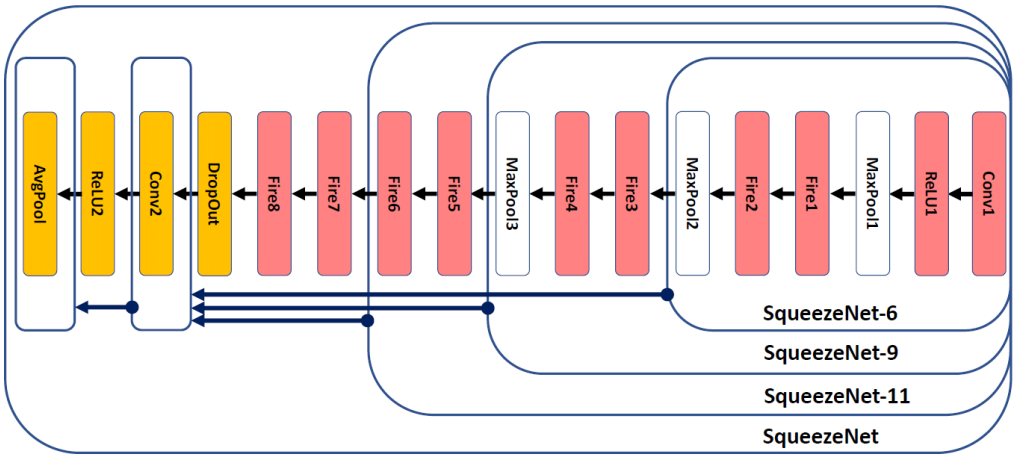


Figura 14: Esquema de la red neuronal SqueezeNet. [19]

1x1, y por una **capa expandida**, con filtros de 1x1 y 3x3 (Figura 15). La capa de compresión se usa para limitar la cantidad de entradas a los filtros 3x3 de la capa expandida, ya que los filtros de 1x1 reducen la dimensión o los canales de las entradas.

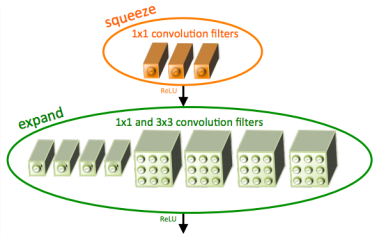


Figura 15: Modulo de fuego de la red neuronal SqueezeNet. [20]

4

Resultados y discusión

4.1. Introducción

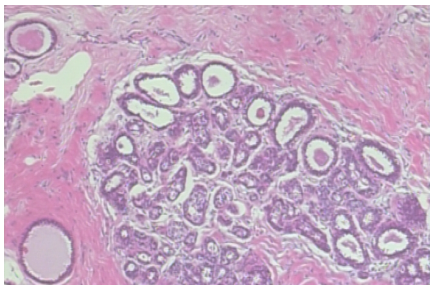
En este capítulo se mostraran los resultados obtenidos de los distintos experimentos que se han ido realizando con el fin de establecer unos parámetros óptimos en el entrenamiento de los modelos.

4.1.1. Conjunto de datos

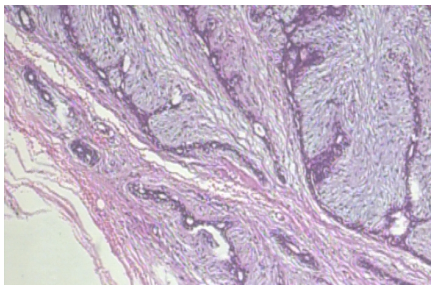
El conjunto de datos usado en este experimento pertenece a la Base de Datos de Imágenes Histopatológicas de Cáncer de Mama (*BreakHis*) [2], compuesto por 9109 imágenes microscópicas de tejido de cáncer de mama, recolectado en colaboración con el laboratorio de anatomía patológica y citología **P&D Laboratory** en Brasil.

De todas las imágenes de esta base de datos solo se han usado las de 40 aumentos, reduciendo nuestro conjunto a un total de 1995 imágenes, de las cuales 625 son benignas y 1370 son malignas (Figura 16). Estas imágenes han sido clasificadas, como se ha expuesto anteriormente, en una clase binaria pudiendo ser benigna (representada con 0) o maligna (representada con 1). Las dimensiones de las imágenes son de 700x460 (ancho x alto) píxeles, aunque estos serán ajustadas para que sirvan de entrada a cada red entrenada.

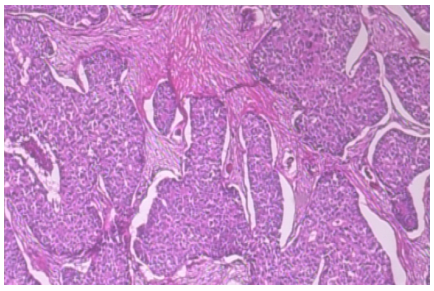
Por otro lado, conjunto total de las imágenes va a ser dividido en tres partes: Conjunto de entrenamiento, validación y prueba, con unos porcentajes correspondientes de 60 % - 20 % - 20 %.



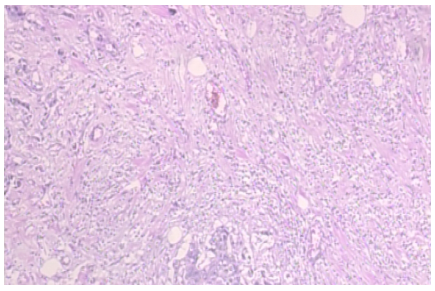
(a) 1a



(b) 1b



(c) 1c



(d) 1d

Figura 16: Ejemplos de imágenes de 40x utilizadas en este proyecto. Las imágenes 1a y 1b son tejidos benignos mientras que las imágenes 1c y 1d pertenecen a tejidos malignos.

4.1.2. Métricas usadas

Para evaluar el rendimiento de los distintos modelos generados se ha hecho uso de las métricas de precisión (*accuracy*) y función de perdida (*loss function*).

La **precisión** o *accuracy* es una métrica de evaluación para modelos clasificadores. Está métrica mide la porción de muestras que nuestro modelo clasifica bien (Ecuación 1). En clasificaciones binarias la precisión también puede ser calculada en función de los valores de la matriz de confusión que se obtenga del entrenamiento (Ecuación 2).

$$\text{Precisión} = \frac{\text{Número de muestras bien clasificadas}}{\text{Número total de muestras}} \tag{1}$$

$$\text{Precisión} = \frac{TP + TN}{TP + TN + FP + FN} \tag{2}$$

Donde *TP* = Verdaderos positivos, *TN* = Verdaderos negativos, *FP* = Falsos positivos, *FN* = Falsos negativos.

Con los cuatro valores de la matriz de confusión también podemos obtener de forma sencilla otras métricas interesantes, que nos darán una mejor visión de los resultados del modelo. Estas métricas se utilizan para medir los resultados positivos y negativos por separado, ya que esto puede ser necesario en algunos estudios. En el nuestro, por ejemplo, es interesante evitar sobretodo los falsos negativos, ya que es importante diagnosticar la enfermedad del cáncer a tiempo.

Estas métricas son la **sensibilidad o tasa de recuperación** y la **especificidad o tasa negativa verdadera**. La sensibilidad se calcula como el número de predicciones positivas correctas entre el número total de positivos (Ecuación 3) y la especificidad se calcula como el número de predicciones negativas correctas entre el número total de negativos (Ecuación 4).

$$\text{Sensibilidad} = \frac{TP}{TP + FN} \tag{3}$$

$$\text{Especificidad} = \frac{TN}{TN + FP} \tag{4}$$

La **función de pérdida** o *loss function* es una función usada para evaluar una solución candidata. Esta calcula un valor que indica cómo de lejos está el valor estimado por el modelo del valor real. Existen diferentes funciones para el cálculo del error por ejemplo el error cuadrático medio, usado para problemas de regresión, o la **pérdida de entropía cruzada**, usado para problemas de clasificación.

La pérdida de entropía cruzada es la que emplearemos para el cálculo de la pérdida. La entropía cruzada describe la distancia entre dos distribuciones de probabilidad de un conjunto de eventos, en cada interacción esta entropía se minimiza reduciendo el error y obteniendo un modelo cada vez más aproximado a la distribución real. [18]

Para dos variables discretas p y q, la entropía cruzada se define como:

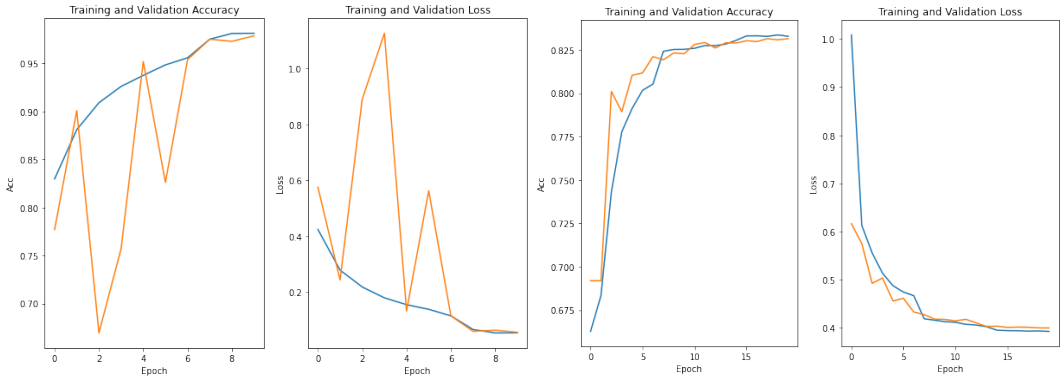
$$H(p, q) = -\sum p(x) \log q(x) \tag{5}$$

4.2. Ajuste de parámetros

Con el fin de obtener el mejor rendimiento posible para nuestro modelo se ha realizado un estudio que establezca unos valores fijos en el entrenamiento del algoritmo. Estás pruebas se han realizado con la red neuronal de AlexNet.

En cuanto el número de épocas, no hay un valor fijo adecuado para obtener el rendimiento óptimo del modelo, además para cada conjunto de datos se obtendrán resultados diferentes.

Por esto se han realizado varias pruebas para tratar de encontrar un valor adecuado. Inicialmente se probó con 10 épocas, pero las métricas de evaluación no convergían. Por lo que finalmente se llegó a la conclusión que con 20 épocas los resultados de precisión y pérdida alcanzaban valores estables, adecuados para el estudio.



(a) 10 épocas (b) 20 épocas

Figura 17: Comparativa de resultados utilizando 10 épocas en la izquierda y 20 épocas en la derecha. Red utilizada = AlexNet, Tasa de aprendizaje = 0.01

Por otro lado, se ha llevado a cabo un estudio para definir la casa de aprendizaje. Se han probado diferentes valores en torno a la tasas consideradas como .[es]tándares.[en] varios problemas de entrenamiento de redes neuronales. En la Tabla 1 se muestra una tabla de los resultados obtenidos para estos valores. Además de las tasas de aprendizaje que se aprecian en la tabla de resultados, se han hecho pruebas con tasas más altas pero los resultados obtenidos no eran convergentes y por ello se han obviado en el estudio.

Finalmente, tras realizar diferentes pruebas para el ajuste de los parámetros definitivos, vamos a utilizar un total de 20 épocas de entrenamiento y una tasa de aprendizaje de 0.001,

Tasa Aprendizaje	Accuracy (test)
0.05	0.6919
0.01	0.8188
0.005	0.8326
0.001	**0.8352**

Tabla 1: Tabla de resultados al aplicar diferentes tasas de aprendizaje. Red utilizada = AlexNet, número de épocas = 20.

ya que han sido los parámetros que nos han proporcionado los mejores resultados. Con estos parámetros fijos, entrenaremos varias redes neuronales diferentes tratando de conseguir la mayor precisión posible en un modelo.

4.3. Resultados de las redes

Una vez obtenidos los parámetros de nuestra red, se procede a entrenar las redes neuronales convolucionales AlexNet, ResNet 18, ShuffleNet V2 y SqueezeNet. En cuanto a la técnica de validación cruzada, se han definido 4 iteraciones para el entrenamiento de cada modelo.

4.3.1. Precisión y pérdida

En la Tabla 2 se muestran los valores de *accuracy* y *loss* obtenidos en la fase de entrenamiento, validación y prueba de cada modelo. Como se puede observar, la precisión disminuye con el conjunto de test respecto a los otros conjuntos ya que las imágenes de test son independientes y no han sido entrenadas anteriormente. En la Figura 18 se puede observar la evolución del entrenamiento a medida que avanzan las épocas para cada modelo.

El modelo que proporciona mejor precisión final es ResNet 18, seguido de AlexNet. Realmente no hay mucha diferencia entre los cuatro modelos, todos han obtenido un rendimiento parecido.

Red neuronal	Accuracy (train)	Loss (train)	Accuracy (val)	Loss (val)	Accuracy (test)	Loss (test)
AlexNet	0.8275 ± 0.00966	0.40719	0.8313 ± 0.00621	0.40344	0.8346 ± 0.044435	0.43973
ResNet 18	0.9948 ± 0.00261	0.01986	0.9832 ± 0.00391	0.04504	**0.8536 ± 0.08664**	0.80704
ShuffleNet V2	0.9772 ± 0.01086	0.06364	0.9625 ± 0.01317	0.09637	0.8249 ± 0.06863	0.82653
SqueezeNet	0.9085 ± 0.00837	0.22691	0.9051 ± 0.00794	0.23764	0.8206 ± 0.06568	0.57792

Tabla 2: Precisión y pérdida de los conjuntos de entrenamiento, validación y test de los diferentes modelos entrenados.

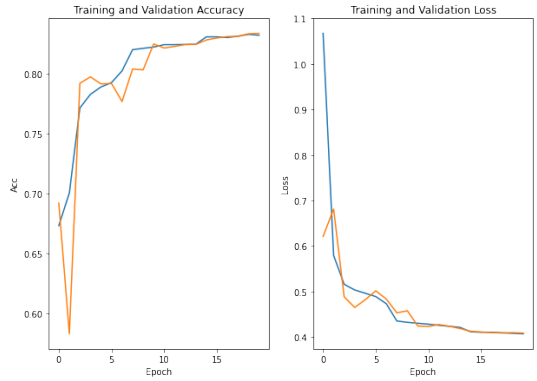
4.3.2. Matriz de confusión y métricas derivadas

Lo siguiente que se va a observar es la matriz de confusión de cada modelo. Para ello se ha calculado la matriz de confusión media de las distintas iteraciones por las que ha pasado el modelo en el entrenamiento. En ellas se puede observar el rendimiento de la red en la clasificación de imágenes, mostrando cuánto de bien o de mal clasifica cada categoría (benigno y maligno).

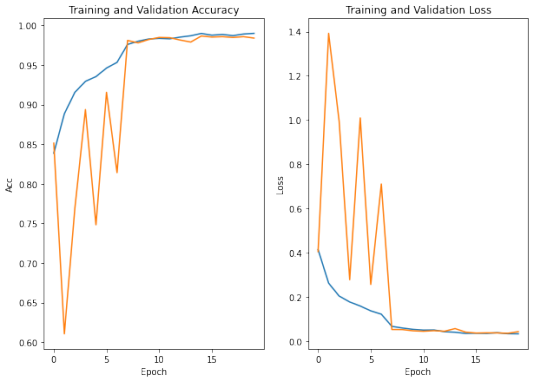
Como se puede observar en la Tabla 3, el modelo tiene tendencia a clasificar mejor las imágenes con tumores malignos, esto se puede deber a que existe un desbalanceo de las clases.

Por otro lado, en la Figura 19 se ha construido la matriz de confusión de la última iteración de la red ResNet, ya que ha tenido buenos resultados.

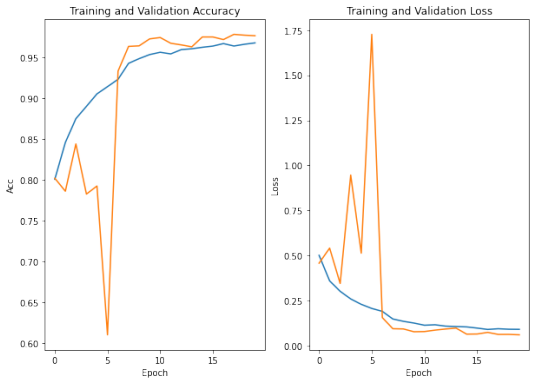
Finalmente, también se han obtenido los valores de sensibilidad y especificidad de los modelos (Tabla 4). Gracias a estas medidas podemos saber que el modelo tiene mayor precisión a la hora de calcular casos positivos.



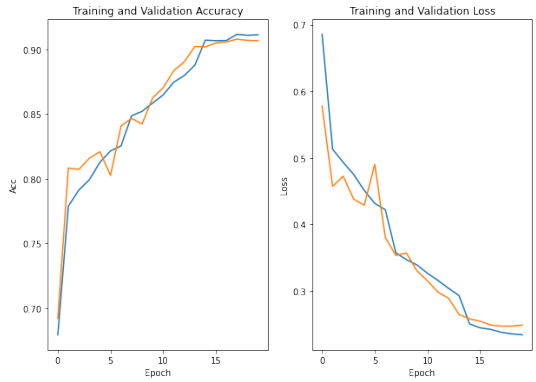
(a) AlexNet



(b) ResNet 18



(c) ShuffleNet V2



(d) SqueezeNet

Figura 18: Curvas de aprendizaje de cada modelo. En azul se observan los resultados de entrenamiento y en naranja los de validación. En cada par de gráficas podemos ver la evolución de la precisión y de la pérdida de las redes.

4.3.3. Tiempo de ejecución

Por último, también es interesante comparar los tiempos de ejecución de los modelos (Tabla 5). En general no hay mucha diferencia entre la duración del entrenamiento de cada red. La red ShuffleNet V2 es la que tiene un mayor tiempo de ejecución, esto puede deberse a que esta arquitectura es la que cuenta con un mayor número de capas (50 capas), a diferencia de las demás que tienen entre 15-20 capas. Por otro lado, SqueezeNet es la red que ha tardado menos en realizar el entrenamiento del modelo, lo cual es lógico sabiendo que el objetivo de esta red es reducir al máximo el coste computacional de la misma.

Red neuronal	TN	FP	FN	TP
AlexNet	1166.75	708.25	278.0	3832.0
ResNet 18	1261.25	613.75	261.0	3849.0
ShuffleNet V2	1243.0	632.0	345.5	3764.5
SqueezeNet	1294.0	581.0	544.25	3764.5

Tabla 3: Media de valores de la matriz de confusión resultante de cada iteración de la validación cruzada de cada modelo.

Red neuronal	Sensibilidad	Especificidad
AlexNet	0.9324	0.6223
ResNet 18	0.9365	0.6727
ShuffleNet V2	0.9159	0.6629
SqueezeNet	0.8676	0.6901

Tabla 4: Sensibilidad y especificidad media de los modelos entrenados.

4.4. Comparativa

Tras el análisis de los resultados obtenidos podemos concluir que las cuatro redes neuronales han ofrecido resultados similares, con pequeñas diferencias y matices según la red elegida.

Sin embargo, una de las redes ha destacado por encima de las demás. La red ResNet 18 ha obtenido una precisión final del 85 %, por lo que tiene mayor capacidad de generalización que las demás.

Por ello, esta red será la que se use para clasificar las imágenes que lleguen a la aplicación web.

4.5. Función de clasificación

Para realizar la clasificación de una imagen, esta se divide en 15 patches iguales que, tras cargar el modelo, se evalúan obteniendo las 15 predicciones, que cada una de ellas se corresponderán con un valor de cero o uno. Tras esto, se ha tenido que realizar un estudio para elegir

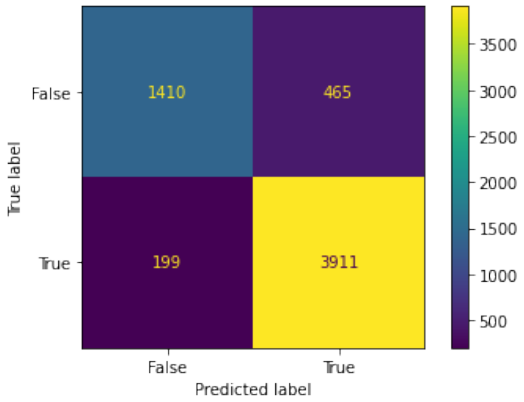


Figura 19: Matriz de confusión de los resultados de la iteración 4 de validación cruzada de la red ResNet 18.

Red neuronal	Tiempo ejecución (segundos)
AlexNet	1634
ResNet 18	1611
ShuffleNet V2	1768
SqueezeNet	1556

Tabla 5: Tiempos de ejecución totales del entrenamiento de los distintos modelos.

que estrategia calculaba mejor la predicción final de la imagen en función de la predicción de cada uno de los patches.

Este estudio ha consistido en calcular la proporción de patches malignos para cada imagen. Si esta proporción supera un umbral, consideramos que la imagen es maligna, si no, es benigna. Hemos probado a realizar este proceso con varios umbrales para ver cual se ajusta mejor a nuestros datos. Los resultados se han obtenido mediante los valores de la matriz de confusión y posteriormente, se ha calculado la precisión obtenida para cada clase (Tabla 6).

En los resultados de la tabla podemos ver que el umbral que alcanza una precisión más justa en las dos clases es 0.90, por lo que es el que utilizaremos para el cálculo del resultado final.

Umbral	TN	FP	FN	TP	Precisión positivos	Precisión negativos
0.10	30	95	1	273	0.99	0.24
0.25	40	85	2	272	0.99	0.32
0.50	64	61	7	267	0.97	0.51
0.75	86	39	15	259	0.95	0.69
0.90	93	32	25	249	0.91	0.74

Tabla 6: Resultados de aplicar diferentes umbrales a la clasificación de las imágenes de test.

5

Aplicación Web

En este capítulo se explicará como se ha desarrollado la aplicación web de recogida de datos y el resultado de esta. Como se ha comentado anteriormente, se va a utilizar la red neuronal que ha obtenido mejores resultados en el proceso de entrenamiento que se ha llevado a cabo. Además explicaremos las herramientas que nos han ayudado a crear la aplicación y la estrategia elegida para calcular los resultados finales.

Para poder realizar estas pruebas se han usado las imágenes de test del entrenamiento, que suman un total de 399 imágenes.

5.1. Herramientas utilizadas

Para la creación de la app, se han utilizado una serie de herramientas que vamos a explicar una a una:

- **Visual Studio Code**: Esta herramienta es un editor de código fuente donde hemos desarrollado la interfaz, los ficheros HTML y el fichero de estilos CSS.

  La **interfaz de programación de aplicaciones o API de REST** conecta el algoritmo de clasificación de imágenes con la interfaz de usuario. Esta programada en *python* y alberga la lógica entre los dos ficheros HTML.

  Los **ficheros HTML** son los que implementan las páginas de la aplicación web. Tenemos un HTML para la página principal y otro para la página de resultados.

  El **fichero de estilos CSS** define la apariencia de los dos ficheros HTML. Mediante este podemos personalizar (colores, formas, tamaños, posiciones...) como se va a visualizar cada elemento de la aplicación web.

- **Flask**: Es un framework que permite crear aplicaciones web de forma sencilla, a través de los métodos que proporciona. Este se carga mediante un paquete de *python*.
- **XAMPP**: Por último, Xampp es un paquete de software libre que permite probar los desarrollos web en una máquina local. Este proporciona un sistema de gestión de bases de datos MySQL, un servidor Apache e intérpretes para algunos lenguajes de script. En nuestro caso, solo hemos hecho uso del servidor, para poder visualizar la aplicación.

5.2. Resultados

El primer paso para la implementación de la interfaz ha sido desarrollar la función que cargue el modelo ResNet 18, para su uso en la clasificación de las imágenes que se introduzcan en la aplicación. Tras ello, se realiza la división de la imagen de entrada en patches. Posteriormente se obtiene la predicción de cada uno de los patches y, finalmente, se obtiene la predicción de la imagen de entrada. Todo ello se ha completado mediante una diversos ficheros HTML y CSS que sirven de interfaz, mediante la cual un usuario puede ejecutar la aplicación y visualizarla en su navegador habitual.

En la Figura 20 podemos ver cómo quedó la página principal de la aplicación, en la que tenemos un formulario para cargar la imagen histopatológica y un botón para ejecutar la función de clasificación sobre esa imagen.

En la Figura 21 se puede observar cómo se carga la imagen, desde un directorio de carpetas local.

Por último, si se pincha en el botón (Figura 22), aparece el resultado obtenido sobre la imagen, que puede ser positivo o negativo en cáncer.

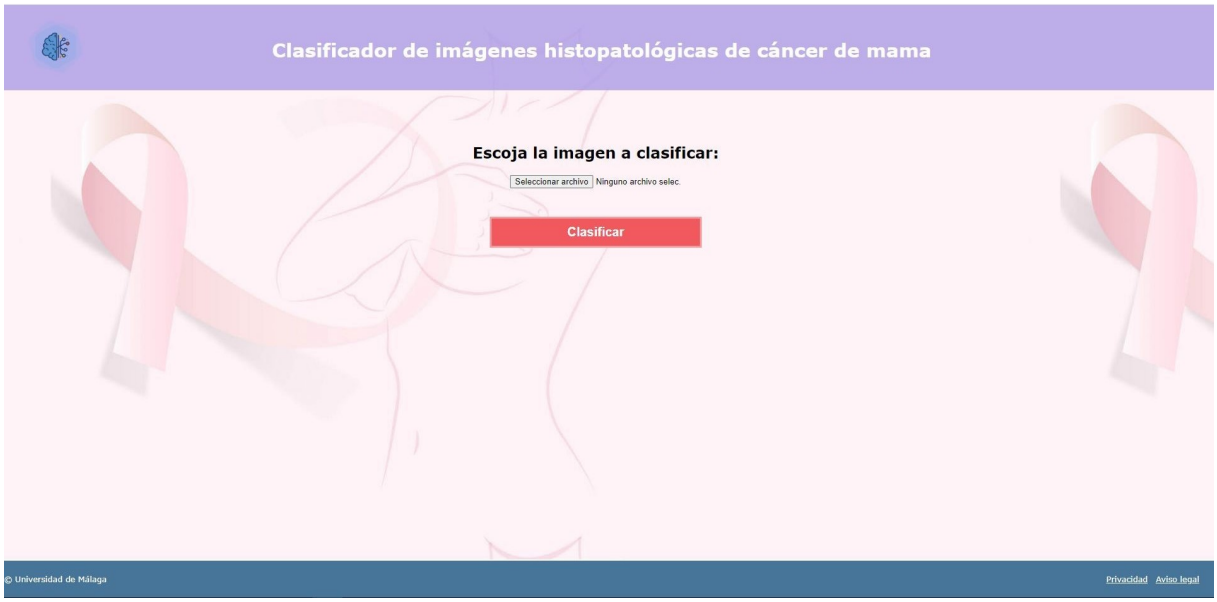


Figura 20: Interfaz aplicación web.

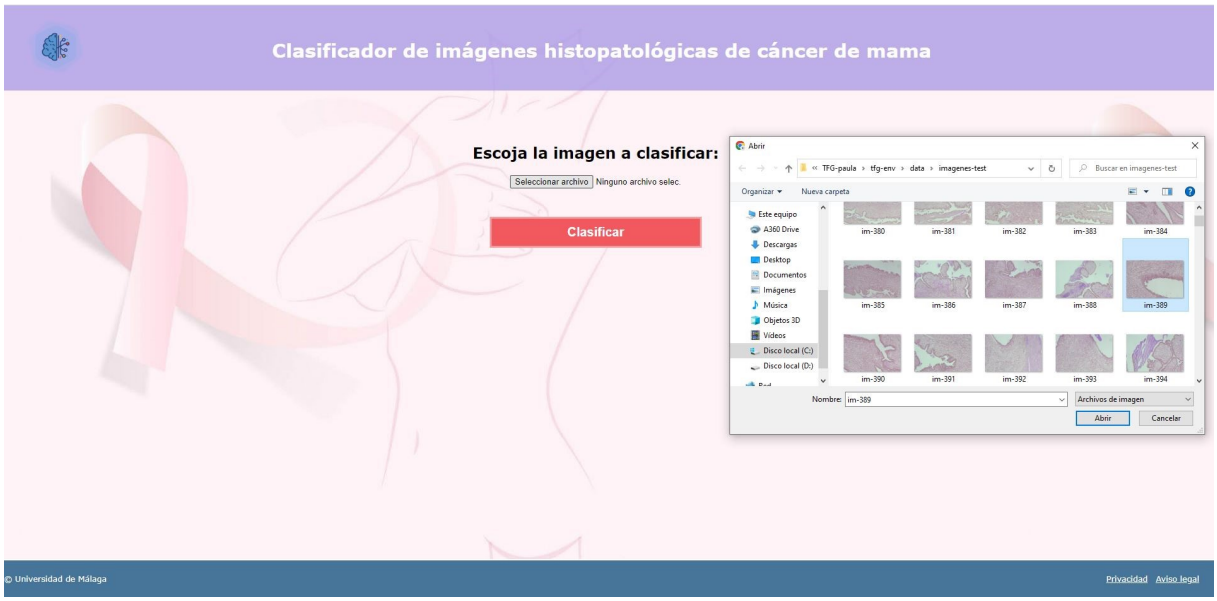


Figura 21: Interfaz aplicación web. Se escoge una imagen para clasificar.

Figura 22: Interfaz aplicación web. Resultados.

6

Conclusiones y líneas futuras

6.1. Conclusiones

El uso del aprendizaje profundo y en concreto de las redes neuronales convolucionales es un gran avance para el diagnóstico clínico que está evolucionando a gran escala en estos últimos años. Las aplicaciones que se podrían ofrecer en este campo son infinitas y facilitarían mucho el trabajo humano, consiguiendo resultados más certeros y de forma más rápida.

El objetivo de este proyecto, como se mencionó en el capitulo 1.2, ha sido estudiar diferentes metodologías de aprendizaje computacional para construir un modelo de redes neuronales convolucionales lo más preciso posible, teniendo en cuenta los recursos disponibles.

Para ello, se utilizó el conjunto de datos *BreakHis Dataset* formado por imágenes histopatológicas de tejido tumoral mamario. Estas imágenes fueron preprocesadas y divididas en *patches* antes de ser incorporadas a la red.

Para elegir los parámetros más óptimos para el entrenamiento del algoritmo, se realizó un ajuste de hiperparámetros, que consiste en entrenar la red varias veces con distintos parámetros y comparar los resultados obtenidos.

El entrenamiento del modelo se ha realizado con cuatro tipos de red neuronal diferentes: AlexNet, Resnet 18, ShuffleNet V2 y SqueezeNet. Las resultados han desvelados que las cuatro redes obtienen rendimientos similares pero ResNet18 sobresale ligeramente con una precisión del 85 %.

Para realizar los diferentes experimentos expuestos en esta memoria, se ha seguido el mé-

todo científico. Por ello podemos afirmar que estos son rigurosos y demostrables. Gracias a haber hecho uso de este método hemos podido realizar una planificación de cada etapa de este estudio, se han valorado diferentes estrategias en base a los objetivos que queríamos alcanzar y finalmente se ha obtenido un resultado fiable que podemos divulgar a la comunidad científica.

Finalmente, me gustaría añadir que este resultado no es suficiente como para poder diagnosticar cáncer de mama ya que una enfermedad de esta envergadura necesita algo más de precisión que un 85 % de probabilidad. Sin embargo, sí podría ser de gran ayuda o soporte para los profesionales sanitarios.

6.2. Líneas futuras

Respecto a posibles mejoras y extensiones de este proyecto se proponen los siguientes puntos:

- Probar a entrenar el modelo con otras redes más complejas o con un mayor número de capas, que requieren un mayor coste computacional y más tiempo, estas podrían ser VGG, ResNet50 o ResNet152. Con este tipo de redes podríamos obtener resultados más precisos, ya que con un mayor número de capas, el entrenamiento es más profundo. Sin embargo, el problema es que necesitaríamos máquinas más potentes, preparadas para poder ejecutar estas redes.

- Aumentar el conjunto de datos con más imágenes histopatológicas de cáncer de mama, por ejemplo, se podrían incorporar al entrenamiento imágenes con distintos aumentos. Si obtenemos imágenes de diferentes fuentes de datos, el problema estaría en que necesitaríamos realizar un preprocesamiento de las imágenes mucho más riguroso para que variables como el ruido, el color o la intensidad no supongan un problema en el entrenamiento. También supondría más coste y tiempo computacional.

- Abordar la clasificación desde un enfoque multiclase, que permita al algoritmo clasificar las imágenes en función del tipo de cáncer de mama. Esto sería bastante útil ya que a parte de conocer si una imagen es cancerígena, podríamos saber que tipo de cáncer tiene. Para ello, habría que realizar algunas modificaciones en el algoritmo actual que permita este tipo de clasificación.

- Realizar un estudio más exhaustivo de optimización de hiperparámetros, probando con distintas combinaciones de ellos para obtener el mejor rendimiento del modelo. Al igual que el primer punto, esto podría mejorar el resultado del modelo pero también hay que tener en cuenta que sería más costoso computacionalmente.
- Por último, también se podrían aplicar diferentes técnicas de preprocesamiento como *data augmentation, transfer learning o feature extraction.* Estas técnicas se podrían probar para observar que efectos tienen sobre nuestros datos e incluirlas en el estudio.

Referencias

[1] Molina-Cabello, M. A., Rodriguez-Rodriguez, J. A., Thurnhofer-Hemsi, K., & Lopez-Rubio, E. (2021). Histopathological image analysis for breast cancer diagnosis by ensembles of convolutional neural networks and genetic algorithms. *Proceedings of the International Joint Conference on Neural Networks, 2021-July.*

[2] Spanhol, F. A., Oliveira, L. S., Petitjean, C., & Heutte, L. (2016). Breast cancer histopathological image classification using Convolutional Neural Networks. *Proceedings of the International Joint Conference on Neural Networks, 2016-October,* 2560–2567.

[3] Litjens, G., Sánchez, C. I., Timofeeva, N., Hermsen, M., Nagtegaal, I., Kovacs, I., Hulsbergen -Van De Kaa, C., Bult, P., van Ginneken, B., & van der Laak, J. (2016). *Deep learning as a tool for increased accuracy and efficiency of histopathological diagnosis.*

[4] Wang, Z., Dong, N., Dai, W., Rosario, S. D., & Xing, E. P. (2018). Classification of Breast Cancer Histopathological Images using Convolutional Neural Networks with Hierarchical Loss and Global Pooling. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 10882 LNCS,* 745–753.

[5] Rashmi, R., Prasad, K., & Udupa, C. B. K. (2022). Breast histopathological image analysis using image processing techniques for diagnostic puposes: A methodological review. In *Journal of Medical Systems* (Vol. 46, Issue 1). Springer.

[6] Pérez Lorenzo Tutor, C., & Carlos San Miguel Avedillo, J. (2019). *UNIVERSIDAD AUTÓNOMA DE MADRID ESCUELA POLITÉCNICA SUPERIOR.*

[7] García Rojo Patólogo Cádiz Gloria Bueno, M. (n.d.). *"Deep Learning". Aplicaciones en Patología.*

[8] Min, S., Lee, B., & Yoon, S. (2017). Deep learning in bioinformatics. *Briefings in Bioinformatics, 18*(5), 851–869. https://doi.org/10.1093/BIB/BBW068

[9] *¿Qué es machine learning? - España | IBM.* (n.d.). Retrieved May 6, 2022, from https://www.ibm.com/es-es/cloud/learn/machine-learning

[10] Lecun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature 2015 521:7553, 521*(7553), 436–444. https://doi.org/10.1038/nature14539

[11] *Aprendizaje Supervisado y No Supervisado - Fernando Sancho Caparrini.* (n.d.). Retrieved August 19, 2022, from http://www.cs.us.es/ fsancho/?e=77

[12] *Introducción al Machine Learning: Una Guía Desde Cero | by Victor Roman | Ciencia y Datos | Medium.* (n.d.). Retrieved August 19, 2022

[13] *Overfitting In Machine Learning – Perpetual Enigma.* (n.d.). Retrieved August 22, 2022, from https://prateekvjoshi.com/2013/06/09/overfitting-in-machine-learning/

[14] *Changes in the loss function vs. the epoch by the learning rate [40]. | Download Scientific Diagram.* (n.d.). Retrieved August 22, 2022, from https://www.researchgate.net/figure/Changes-in-the-loss-function-vs-the-epoch-by-the-learning-rate-40_fig2_341609757

[15] *Redes Residuales: ResNet • Un artículo de La Máquina Oráculo.* (n.d.). Retrieved August 25, 2022, from https://lamaquinaoraculo.com/computacion/redes-residuales/

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *in Advances in neural information processing systems,* 2012.

[17] X. Han, Y. Zhong, L. Cao, and L. Zhang, "Pre-trained alexnet architecture with pyramid pooling and supervision for high spatial resolution remote sensing image scene classification," in *Remote Sensing*, 2017.

[18] *Definición de entropía cruzada..* Retrieved August 29, 2022, from https://www.lokad.com/es/definicion-de-entropia-cruzada

[19] *The SqueezeNet architecture and the considered subsets of layers used... | Download Scientific Diagram.* Retrieved September 2, 2022, from https://www.researchgate.net/figure/The-SqueezeNet-architecture-and-the-considered-subsets-of-layers-used-to-build_fig2_331783730

[20] *SqueezeNet Fire Module [24]. | Download Scientific Diagram.* Retrieved September 2, 2022, from https://www.researchgate.net/figure/SqueezeNet-Fire-Module-24_fig3_345161797

[21] *ShuffleNet V2 Block Explained | Papers With Code.* Retrieved September 4, 2022, from https://paperswithcode.com/method/shufflenet-v2-block

[22] *Original ResNet-18 Architecture | Download Scientific Diagram.* Retrieved September 4, 2022, from https://www.researchgate.net/figure/Original-ResNet-18-Architecture_fig1_336642248

[23] *Build Your Own Convolution Neural Network in 5 mins | by Rohith Gandhi | Towards Data Science.* Retrieved September 4, 2022, from https://towardsdatascience.com/build-your-own-convolution-neural-network-in-5-mins-4217c2cf964f

Apéndice A
Manual de instalación

Para poder hacer uso de la aplicación, se necesita cumplir unos requisitos necesarios:

- Disponer de la carpeta con el proyecto de la aplicación web: dentro de esta se encuentran los ficheros **algoritmo.py, tfgapp.py**, la carpeta **static** con las imágenes que se utilizan en la aplicación y el archivo de estilos, la carpeta **templates** que contiene los ficheros HTML y las **imágenes de prueba** que se encuentran en el directorio */data/imagenes-test*. Todo el código del proyecto se puede encontrar en GitHub en la siguiente dirección: `https://github.com/Paulandujar/CNN-for-breast-cancer-clasificatio git`.
- Tener instalado Visual Studio Code para poder ejecutar el proyecto. La versión que se ha utilizado es la 1.71.1 en Windows 11. La url de instalación es la siguiente: `https://code.visualstudio.com/download`.
- Tener instalado XAMPP para alojar la aplicación en un servidor.La versión con la que se ha trabajado es la 7.4.29, para poder descargarla: `https://www.apachefriends.org/download.html`.
- Instalar mediante la consola las librerías necesarias para ejecutar los archivos de python.

Una vez cumplidos estos requisitos, la ejecución de la aplicación es sencilla.

Primero, en VS Code y debemos abrir la carpeta del proyecto, para poder trabajar con ella (Figura 23). Para ello pinchamos en 'Archivo' ->'Abrir Carpeta...' y seleccionamos la carpeta del proyecto.

Segundo, debemos arrancar el servidor Apache de XAMPP para alojar la aplicación web. Para esto abrimos el panel de control de XAMPP e iniciamos el modulo Apache pinchando en 'Start' (Figura 24).

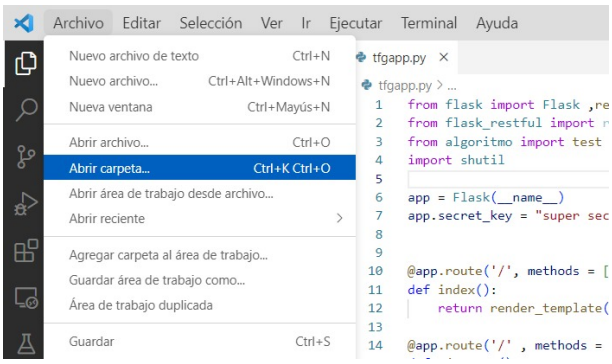


Figura 23: Abrir Carpeta en Visual Studio Code.

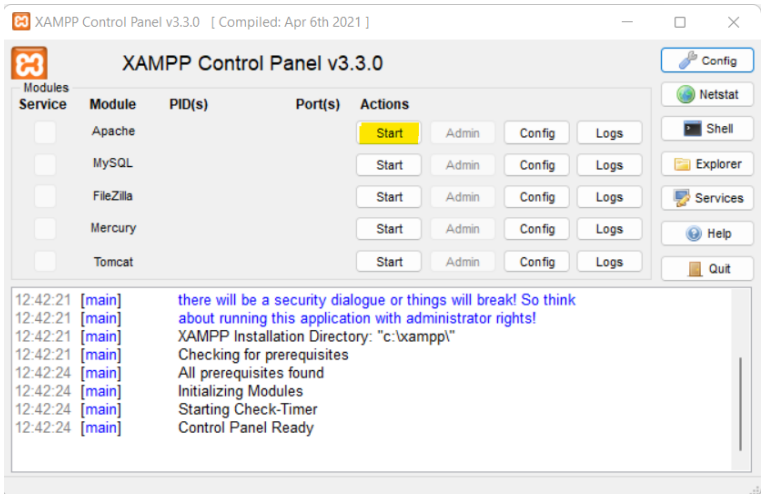


Figura 24: Iniciar servidor Apache de Xampp, pinchamos en botón amarillo.

Una vez hecho esto ya podemos ejecutar la aplicación en VS Code. El archivo que hay que ejecutar es tfgapp.py, pinchando en 'Ejecutar' ->'Ejecutar sin depuración' (Figura 25).

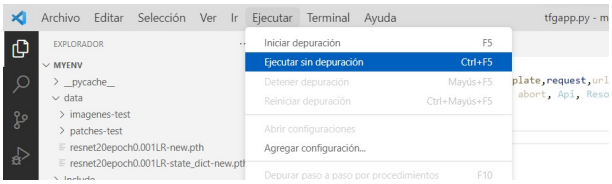


Figura 25: Ejecutar proyecto en Visual Studio Code.

Cuando ya tengamos el proyecto en ejecución abrimos una página nueva en el navegador y pegamos esto: 'http://127.0.0.1:5000/'. Si no funciona, puede que el puerto por defecto del localhost sea otro, esto se puede mirar en la consola de VS Code una vez que hemos ejecutado el proyecto (Figura 26).

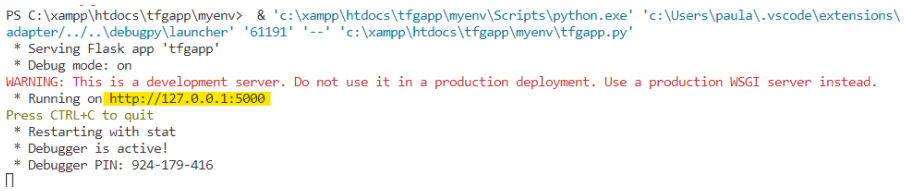


Figura 26: IP donde se encuentra alojada la aplicación. Subrayada de amarillo.

Finalmente, si hemos seguido todos los pasos de forma correcta, ya tendríamos la aplicación puesta en marcha en nuestro navegador.

Un aspecto importante a tener en cuenta es que la imagen que vayamos a cargar en la app debe estar en la ruta */data/imagenes-test*, dentro de la carpeta del proyecto.