Universidad Internacional de La Rioja
Escuela Superior de Ingeniería y Tecnología

Grado en Ingeniería Informática

Desarrollo de un Sistema de Detección de Cáncer de Mama Basado en Redes Neuronales y Métodos de Optimización

Trabajo fin de estudio presentado por:	Ernesto González Pradas
Director/a:	José Alberto Benítez Andrades
Fecha:	04/12/2023
Repositorio del código fuente:	https://github.com/ErnestoGPradas/TrabajoFinalEstudios

Resumen

El presente proyecto se enfoca en el desarrollo de un sistema basado en inteligencia artificial para la detección de cáncer de mama a partir de mamografías. Tras realizar una exhaustiva revisión del estado del arte y del contexto médico relacionado con esta problemática, establecimos los objetivos y la metodología de trabajo necesarios para abordar el problema de manera eficaz. Utilizando herramientas y tecnologías avanzadas, diseñamos una solución basada en redes neuronales convolucionales (CNN) que permitió el análisis automático de las imágenes de mamografías.

El proceso de diseño de la solución incluyó la preparación del entorno de desarrollo, la descarga y procesamiento de imágenes DICOM, así como la implementación y optimización de un modelo CNN. Exploramos diversas estrategias de mejora, como la configuración de Tensorflow para utilizar la GPU del ordenador, la expansión de la arquitectura CNN y la aplicación de técnicas de transfer learning con modelos pre-entrenados.

Los resultados obtenidos reflejaron un desafío importante en cuanto al sobreajuste de los modelos, a pesar de nuestros esfuerzos por mejorar el rendimiento mediante la optimización de hiperparámetros y el uso de técnicas de regularización. Esta limitación nos lleva a reflexionar sobre la adecuación de la ingeniería de datos realizada, especialmente en la fase de confección de los CSV de datos.

Hemos realizado diversas tareas abarcando distintos campos laborales. En la ingeniería de datos, recopilamos imágenes DICOM y metadatos creando archivos CSV para entrenar redes neuronales. En el desarrollo de software, creamos scripts y herramientas para procesar y analizar datos, y configurar modelos de IA. En ingeniería artificial, investigamos y recopilamos arquitecturas de redes neuronales, explorando su aplicación en la detección de cáncer de mama. Este enfoque multidisciplinario nos permitió abordar el problema desde varias perspectivas y lograr avances significativos en la construcción de un sistema de detección basado en IA.

Palabras clave: Ingeniería de datos, redes neuronales convolucionales, detección de cáncer de mama, desarrollo de software, inteligencia artificial.

Abstract

The present project focuses on the development of a system based on artificial intelligence for the detection of breast cancer from mammograms. After an exhaustive review of the state of the art and the medical context related to this problem, we established the objectives and methodology necessary to address the problem effectively. Using advanced tools and technologies, we designed a solution based on convolutional neural networks (CNN) that enabled the automatic analysis of mammography images.

The solution design process included preparation of the development environment, downloading and processing of DICOM images, as well as implementation and optimization of a CNN model. We explored various enhancement strategies, such as configuring Tensorflow to use the computer GPU, expanding the CNN architecture, and applying transfer learning techniques with pre-trained models.

The results obtained reflected a significant challenge in terms of model over-fitting, despite our efforts to improve performance through hyperparameter optimization and the use of regularization techniques. This limitation leads us to reflect on the adequacy of the data engineering performed, especially in the data CSV confection phase.

We have performed several tasks covering different fields of work. In data engineering, we collected DICOM images and metadata by creating CSV files to train neural networks. In software development, we created scripts and tools to process and analyze data, and set up AI models. In artificial engineering, we researched and compiled neural network architectures, exploring their application in breast cancer detection. This multidisciplinary approach allowed us to approach the problem from various perspectives and make significant progress in building an AI-based detection system.

Keywords: Data engineering, convolutional neural networks, breast cancer detection, software development, artificial intelligence.

Índice de contenidos

1. Introducción 1

1.1. Motivación 2

1.2. Planteamiento del trabajo 4

1.3. Estructura del trabajo 4

2. Contexto y Estado del Arte 6

2.1. Visión Global 7

2.2. Población vulnerable 8

2.3. Manifestaciones patológicas 8

2.4. Tratamiento médico 10

2.5. Optimizando la Identificación del Cáncer de Mama con IA 11

3. Objetivos y metodología de trabajo 15

3.1. Objetivo general 15

3.2. Objetivos específicos 16

3.3. Metodología de trabajo 17

4. Entorno de desarrollo y tecnologías 19

4.1. Entorno de desarrollo 19

4.2. Banco de imágenes 21

4.3. Tecnologías 22

5. Diseño de la solución 24

5.1. Preparación del entorno, descarga de las imágenes y ciencia de datos 24

5.1.1. Normalizado de rutas para poder realizar el estudio 25

5.1.2. Relación de csv de entrenamiento y test con sus correspondientes imágenes DICOM 28

5.2. Generación de un modelo CNN 37

5.2.1. Definición y descripción de modelos CNN 38

5.2.2. Implementación de nuestro modelo CNN 44

6. Estrategias de mejora en nuestro modelo CNN 49

6.1. Configuración de Tensorflow para que utilice la GPU del ordenador 49

6.2. Mejora de la Arquitectura CNN a través de Capas Convolutivas y Técnicas de Regularización 50

6.2.1. Expansión de la profundidad y complejidad de la red CNN 50

6.2.2. Ajuste del optimizador y la tasa de aprendizaje 52

6.2.3. Optimización de los datos de entrada con aumento de datos 53

6.2.4. Entrenamiento mejorado con datos aumentados 54

6.3. Transfer Learning y otras arquitecturas de redes neuronales 56

6.3.1. ¿Qué es MobileNetV2? 56

6.3.2. Codificamos MobileNetV2 58

6.3.3. ¿Qué es MobileNetV3? 60

6.3.4. Codificamos MobileNetV3 61

6.3.5. ¿Qué es ResNet? 63

6.3.6. Codificamos ResNet 63

6.3.7. ¿Qué es Inception GoogLeNet? 65

6.3.8. Codificamos Inception GoogLeNet 65

6.3.9. ¿Qué es DenseNet? 67

6.3.10. Codificamos DenseNet 67

6.3.11. ¿Qué es VGG (Visual Geometry Group)? 69

6.3.12. Codificamos VGG 69

6.3.13. ¿Qué es EfficientNet? 71

6.3.14. Codificamos EfficientNet 71

6.3.15. Analizamos los resultados....73

7. Conclusiones y trabajo futuro....79

7.1. Conclusiones del trabajo....79

7.2. Líneas de trabajo futuro....79

Referencias bibliográficas....81

Anexo A. Script para normalizado de ruta de las imágenes....84

Anexo B. Estructura completa de las tablas de test y training....86

Anexo C. Configuración del entorno para poder utilizar la GPU....86

Índice de figuras

Figura 1. Comparativa de los resultados obtenidos por la IA y por un equipo de radiólogos...2

Figura 2. IA del MIT detectando cáncer de mama 4 años antes de desarrollarse.....3

Figura 3. Número de muertes por cáncer registrado en mujeres en España en 2021, por tipo6

Figura 4. Anatomía de la mama femenina 7

Figura 5. Posibles síntomas del cáncer de mama 9

Figura 6. Estadios del cáncer de mama.....10

Figura 7. Descripción general del diseño del cUSBr-Patch.....12

Figura 8. Estudio en vivo sobre el tejido mamario.....13

Figura 9. Modelo de proceso incremental 18

Figura 10. Mamografía en formato DICOM 22

Figura 11. Conjunto de imágenes DICOM para descargar 24

Figura 12. Descarga de imágenes DICOM utilizando NBIA Data Retriever.....24

Figura 13. Estructura del directorio de imágenes DICOM descargada 25

Figura 14. Nombre real de los archivos DICOM 30

Figura 15. Ruta relativa dentro del proyecto donde se encuentran los conjuntos de entrenamiento y pruebas en formato csv.....31

Figura 16. Diagrama de las distintas capas que existen dentro de las ciences de la computación 37

Figura 17. Estructura general de una neurona.....39

Figura 18. Neurona vs Perceptrón 39

Figura 19. Primera convolución de una red neuronal convolucional (podemos tener una o varias convoluciones) 41

Figura 20. Esquema del conjunto de las distintas capas CNN.....41

Figura 21. Esquema del modelo matemático Backpropagation 42

Figura 22. Entrenamiento del modelo y resultados 47

Figura 23. Matriz que se genera sobre la imagen de entrada realizando operaciones de multiplicación 51

Figura 24. Añadiendo dropout a las capas 52

Figura 25. Entrenamiento del nuevo modelo y resultados 55

Figura 26. Visualización de los mapas de características intermedias en la capa residual invertida 57

Figura 27. Incorporación de MobileNetV2 en un Esquema de Red Neuronal Convolucional . 58

Figura 28. Estructura de bloques de MobileNetV3 61

Figura 29. Esquema general de Inception GoogLeNet V4 65

Figura 30. Arquitectura DenseNet-121 con imagen histopatológica de cáncer de mama 67

Figura 31. Arquitectura VGG 69

Figura 32. Arquitectura de capas de VGG-16 69

Figura 33. Diagramas del entrenamiento y validación para la pérdida y precisión de MobileNetV2 74

Figura 34. Diagramas del entrenamiento y validación para la pérdida y precisión de MobileNetV3 74

Figura 35. Diagramas del entrenamiento y validación para la pérdida y precisión de ResNet75

Figura 36. Diagramas del entrenamiento y validación para la pérdida y precisión de Inception Google 76

Figura 37. Diagramas del entrenamiento y validación para la pérdida y precisión de DenseNet 76

Figura 38. Diagramas del entrenamiento y validación para la pérdida y precisión de VGG 77

Figura 39. Diagramas del entrenamiento y validación para la pérdida y precisión de EfficieNet 78

Figura 40. Imagen copiada a la ruta principal 85

Figura 41. Ejemplo de salida por consola al ejecutar el script 85

Figura 42. Variables del sistema.....87

Figura 43. Copiamos los binarios de cuDNN a la ruta del sistema de CUDA 87

Figura 44. Versión correcta de Tensorflow 88

Figura 45. GPU configurada y lista para usar 88

Índice de tablas

Tabla 1. Comparativa de IDEs de desarrollo 20

Tabla 2. Campos relevantes en mass_case_description_train_set.csv 29

Tabla 3. Comparativa entre una red neuronal "tradicional" y una red neuronal convolucional 43

Tabla 4. Comparativa resultados distintas arquitecturas CNN 73

1. Introducción

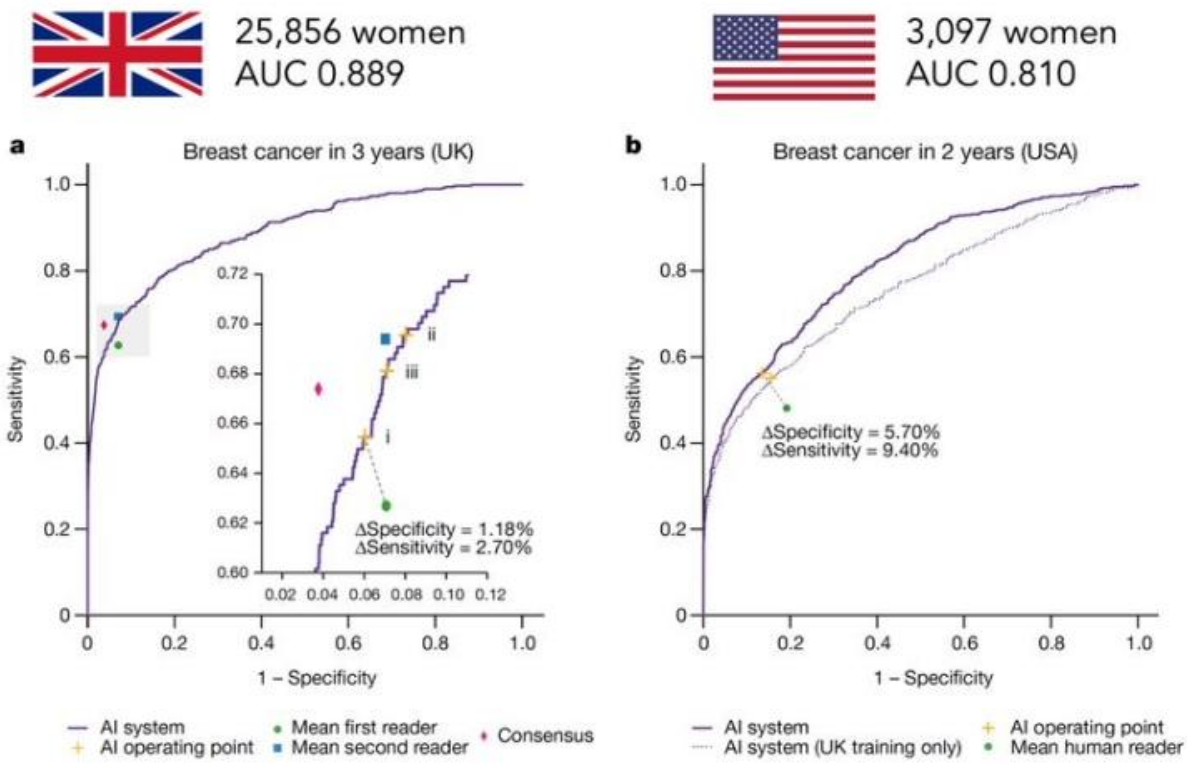
El presente Trabajo de Fin de Grado, titulado " Desarrollo de un Sistema de Detección de Cáncer de Mama Basado en Redes Neuronales y Métodos de Optimización", aborda una temática de suma importancia en el ámbito de la prevención de la salud, donde la aplicación de tecnologías avanzadas se presenta como una herramienta prometedora para la detección temprana de enfermedades. En específico, este trabajo se centra en el desarrollo e implementación de una red neuronal utilizando Python, con el objetivo de analizar mamografías y predecir la presencia o ausencia de cáncer de manera eficiente y precisa.

La detección temprana del cáncer de mama es crucial para mejorar las tasas de supervivencia y reducir el impacto de esta enfermedad en la sociedad. En este contexto, la **inteligencia artificial**, y en particular las **redes neuronales**, se posicionan como una herramienta valiosa al ofrecer la capacidad de aprendizaje a partir de conjuntos de datos extensos y variados. La implementación de este enfoque se ha llevado a cabo mediante un script en Python, el cual se entrena con mamografías previamente etiquetadas y luego se evalúa en la predicción de nuevas imágenes, brindando una herramienta potencialmente útil para los profesionales de la salud.

A lo largo de este apartado, se explorarán los fundamentos teóricos de las redes neuronales aplicadas a la detección de cáncer en mamografías. Se destacarán los conceptos clave, la metodología utilizada en el desarrollo del modelo y la estructura del script implementado. Además, se proporcionarán ejemplos prácticos para ilustrar el proceso de entrenamiento y evaluación de la red neuronal.

Este apartado sienta las bases para comprender el trabajo realizado, destacando la importancia de la aplicación de la inteligencia artificial en la detección precoz de enfermedades, especialmente en el caso del cáncer de mama. La combinación de la experiencia clínica y la capacidad predictiva de las redes neuronales ofrece un enfoque alentador para mejorar los protocolos de diagnóstico y, en última instancia, contribuir al avance de la medicina preventiva.

Figura 1. Comparativa de los resultados obtenidos por la IA y por un equipo de radiólogos

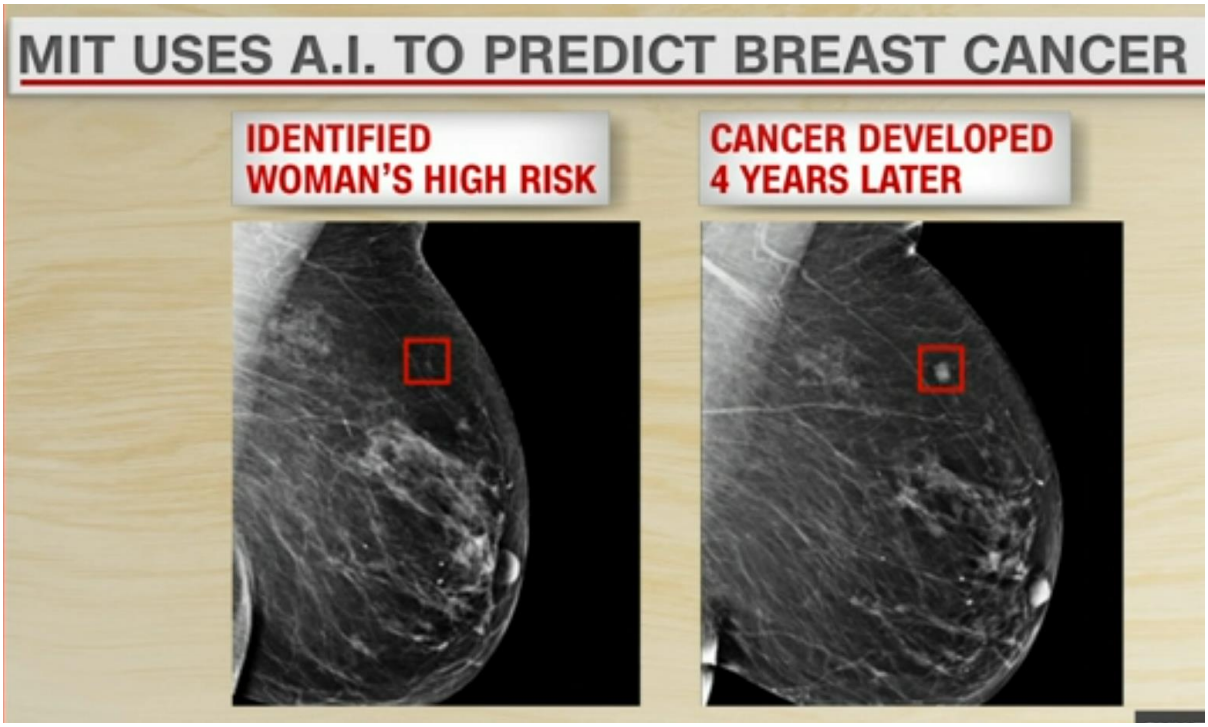


Fuente: https://www.xataka.com/inteligencia-artificial/google-desarrolla-inteligencia-artificial-para-detectar-cancer-mama-capaz-superar-a-expertos-humanos

1.1. Motivación

El principal beneficio de desarrollar sistemas predictivos para el cáncer de mama es una mejora en la calidad de vida de las personas y poder contribuir a la mejora de la atención médica. La motivación primordial radica en la posibilidad de crear una herramienta innovadora que apoye a los profesionales sanitarios especializados en la detección temprana del cáncer de mama y con ello a la mejora de la salud de las pacientes que puedan padecer esta patología. Inspirado por el trabajo pionero realizado por el Instituto Tecnológico de Massachusetts (MIT) en el campo del aprendizaje profundo (deep learning), en este trabajo se propone aprovechar estas tecnologías para desarrollar una solución, siempre teniendo presente el tiempo del que se dispone para realizar este TFG, que pueda integrarse en la práctica clínica, y así poder potenciar la eficacia del diagnóstico tan necesario en este tipo de enfermedades.

Figura 2. IA del MIT detectando cáncer de mama 4 años antes de desarrollarse



Fuente: https://edition.cnn.com/videos/health/2023/03/07/artificial-intelligence-breast-cancer-detection-mammogram-cnntm-vpx.cnn

La herramienta de deep learning desarrollada por el MIT se presenta como un punto de partida crucial para este proyecto, sirviendo como base tecnológica robusta y avanzada. Esta motivación técnica se combina con una perspectiva humanista, alimentada por la conciencia de la importancia crítica de la detección temprana del cáncer de mama. Observar cómo la inteligencia artificial puede ser aplicada para mejorar la capacidad predictiva en el diagnóstico, y así colaborar con los esfuerzos de los profesionales de la salud, ha sido un estímulo constante en mi búsqueda por contribuir al avance de la medicina en cuanto a tecnología y desarrollo se refiere y la atención médica personalizada.

En paralelo con lo mencionado en los párrafos anteriores, una fuente adicional de motivación reside en el firme deseo de aprender y profundizar en las tecnologías fundamentales que impulsan este proyecto, como Python, TensorFlow y NumPy, que constituyen el tejido tecnológico esencial de mi investigación.

La elección de sumergirme en el desarrollo de algoritmos de aprendizaje profundo no solo se fundamenta en la utilidad práctica de estas tecnologías en el ámbito médico, sino también en el reconocimiento de la importancia de adquirir habilidades técnicas avanzadas. El entorno de

programación en Python, conocido por su versatilidad y legibilidad, proporciona un espacio sobre el cual se desarrolla el código necesario para llevar a cabo este TFG de manera eficiente. La inclusión de TensorFlow, especializado en la construcción de modelos de aprendizaje automático, y NumPy, potenciando la manipulación eficaz de datos, representa un desafío estimulante que me impulsa a expandir mis conocimientos y habilidades.

1.2. Planteamiento del trabajo

El objetivo de este trabajo radica en desarrollar una herramienta basada en una red neuronal para la predicción del cáncer de mama a partir de mamografías. Utilizando tecnologías como Python o TensorFlow, el planteamiento se centra en crear un modelo de aprendizaje profundo que, al ser entrenado con mamografías previamente etiquetadas, pueda predecir la presencia de cáncer de manera eficiente. El **proceso de carga de imágenes DICOM**, su preprocesamiento y la construcción de un **modelo de red neuronal convolucional** (CNN) son pasos fundamentales.

Este trabajo incluye la evaluación de la eficacia del modelo en la predicción de cáncer de mama, así como la identificación de áreas de mejora o ajuste. Además, se busca demostrar la utilidad de la herramienta propuesta en la detección temprana de la enfermedad, contribuyendo así a la mejora de la atención médica en el ámbito de la medicina preventiva. La combinación de la implementación de algoritmos específicos y el análisis detallado de los resultados permitirá destacar la relevancia de la inteligencia artificial en la detección de enfermedades críticas como el cáncer de mama.

Las tecnologías utilizadas, como Python para la programación eficiente y legible, TensorFlow como herramienta central en el desarrollo de modelos de aprendizaje automático, y NumPy para la manipulación eficaz de datos, se presentan como pilares fundamentales en la construcción de la solución propuesta. La implementación de una CNN, delineada en el código que se muestra en capítulos futuros, refleja la aplicación concreta de estas tecnologías para la creación de un modelo predictivo, destacando así el enfoque tecnológico de este trabajo.

1.3. Estructura del trabajo

En el Capítulo 2 se sitúa el contexto y Estado del Arte en el que está enmarcado el desarrollo de este TFG. Analizaremos los distintos tipos de redes neuronales y asentaremos las bases de las diferentes tecnologías utilizadas en el código de nuestra aplicación y que nos permitirán

tener entender los objetivos de este TFG. También destacaremos algunas herramientas que ya aportan soluciones similares a las de este trabajo.

En el Capítulo 3 se detallan los objetivos tanto generales como específicos del presente trabajo, así como la metodología de este.

En el Capítulo 4 describiremos el entorno de desarrollo, así como las tecnologías y recursos utilizados.

En el Capítulo 5 expondremos el desarrollo el diseño de la solución, explicando cómo preparar el entorno y generando nuestro primer modelo de Red Neuronal Convolucional (CNN).

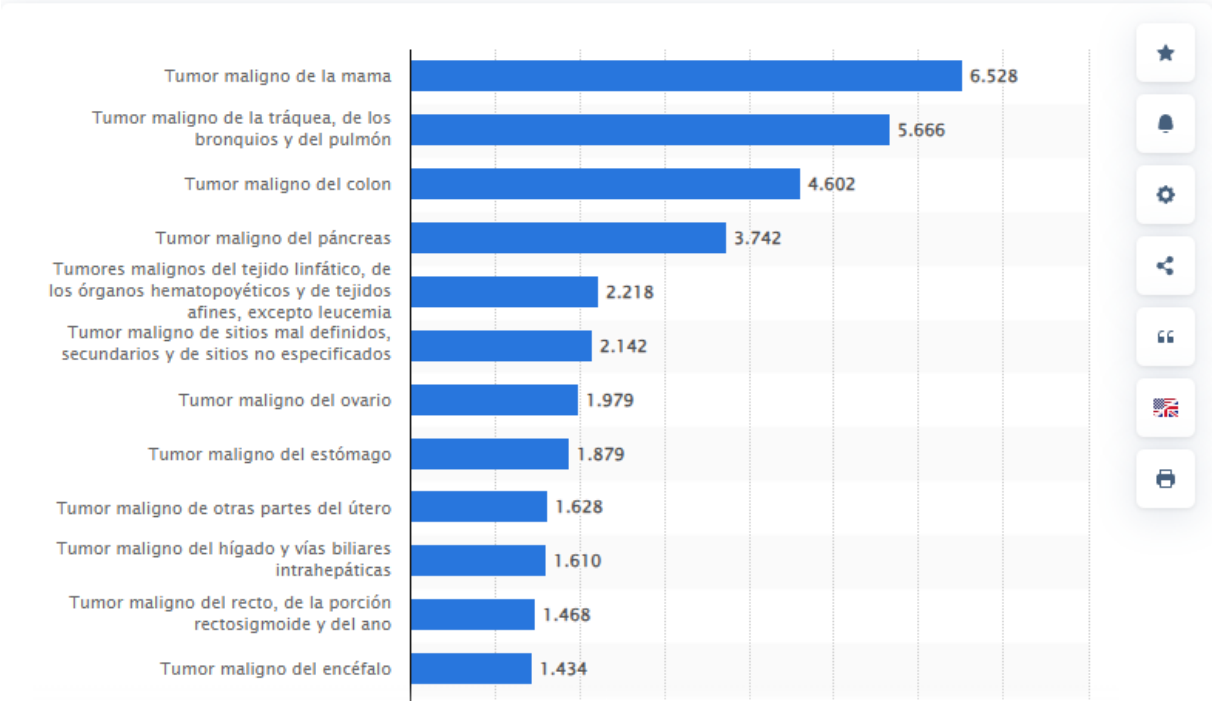
En el Capítulo 6 veremos las diferentes estrategias que podemos seguir a la hora de mejorar nuestro primer modelo de CNN.

En el Capítulo 7 destacaremos las conclusiones del trabajo, así como las mejoras del mismo y las líneas de trabajo futuro.

2. Contexto y Estado del Arte

El cáncer de mama es una de las principales causas de muerte en mujeres a nivel mundial y la detección temprana es crucial para mejorar las tasas de supervivencia y reducir la morbilidad. Sin embargo, los métodos tradicionales de detección pueden ser costosos, invasivos y a veces subjetivos, siendo, el análisis de mamografías mediante técnicas de inteligencia artificial, como las redes neuronales, una oportunidad para mejorar la eficiencia y precisión en la detección temprana.

Figura 3. Número de muertes por cáncer registrado en mujeres en España en 2021, por tipo



Fuente:

https://es.statista.com/estadisticas/677111/muertes-por-cancer-en-mujeres-por-tipo-en-espana/

La magnitud del problema radica en la alta prevalencia del cáncer de mama y la necesidad de un diagnóstico preciso y oportuno. Las mamografías son una herramienta clave en la detección, pero el análisis manual puede ser propenso a errores y está sujeto a la variabilidad del observador. La implementación de un sistema automatizado como el propuesto en este trabajo, puede ayudar a superar estos desafíos, permitiendo una detección más rápida y precisa.

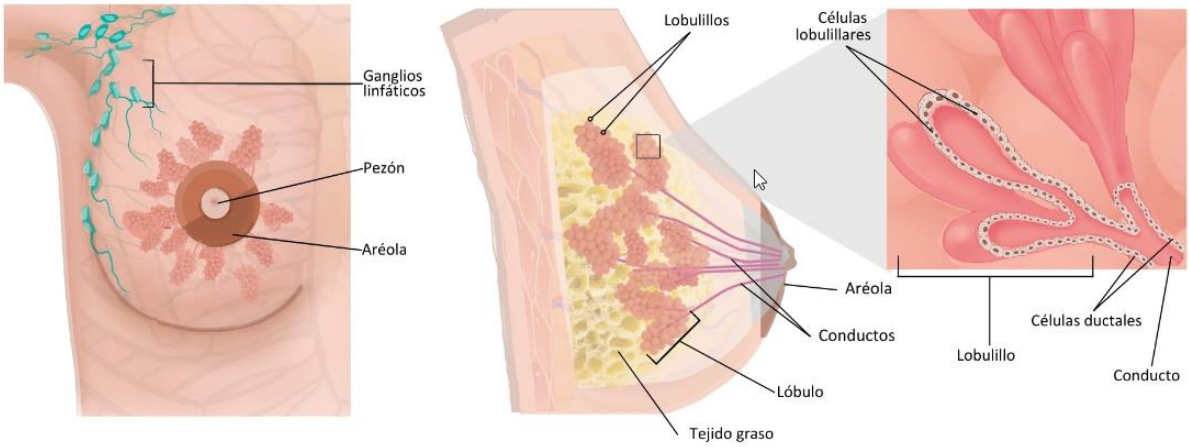
Las limitaciones en los métodos de detección tradicionales, la carga de trabajo en los profesionales de la salud y la necesidad de mejorar la eficiencia en el proceso de diagnóstico son algunas de las causas que motivan la búsqueda de soluciones basadas en inteligencia artificial.

2.1. Visión Global

El cáncer de mama se caracteriza por la proliferación descontrolada de células mamarias alteradas, dando lugar a la formación de tumores que, de no recibir tratamiento, pueden diseminarse por todo el organismo y llegar a provocar la muerte.

La iniciación del proceso canceroso tiene lugar en los **conductos galactóforos** o en los **lobulillos** encargados de la producción de leche en la mama. En su estadio inicial, conocido como cáncer in situ, la enfermedad no presenta riesgo de mortalidad. Sin embargo, a medida que las células cancerosas se propagan hacia el tejido mamario circundante, se forman nódulos o áreas de engrosamiento.

Figura 4. Anatomía de la mama femenina



Fuente: https://www.ruberinternacional.es/en/patient/pathologyes/cancer-mama

Los cánceres invasivos tienen la capacidad de extenderse a los **ganglios linfáticos** cercanos o a otros órganos, dando lugar a **metástasis**, un fenómeno potencialmente letal. La efectividad del tratamiento varía según las características individuales de la paciente, el tipo de cáncer y su grado de diseminación. En términos generales, el tratamiento implica una combinación de intervenciones quirúrgicas, radioterapia y medicación.

2.2. Población vulnerable

El cáncer de mama representa una amenaza significativa para la salud, y al analizar la población vulnerable a esta enfermedad, surgen diversos puntos clave. En primer lugar, el género femenino se identifica como el principal factor de riesgo, aunque se debe tener en cuenta un pequeño porcentaje de casos residuales que afectan a hombres (entre un 0.5% - 1%). Es importante destacar que el tratamiento es similar para ambos sexos, a pesar de las diferencias en la incidencia.

Además del género, varios factores aumentan la probabilidad de desarrollar cáncer de mama. Entre estos factores se incluyen el envejecimiento, la obesidad, el consumo de alcohol y tabaco, antecedentes familiares, historial de exposición a radiación y el tratamiento hormonal postmenopáusico. Estos elementos contribuyen de manera significativa al riesgo de la enfermedad, subrayando la complejidad de sus determinantes.

En un análisis más detallado, se observa que alrededor de la mitad de los casos de cáncer de mama afectan a mujeres sin factores de riesgo identificables, a excepción de su género y una edad superior a los 40 años. Esta observación subraya la importancia de la detección temprana y el monitoreo regular incluso en ausencia de factores de riesgo evidentes.

Adicionalmente, ciertas mutaciones genéticas hereditarias, como las relacionadas con los genes **BRCA1**, **BRCA2** y **PALB2**, incrementan significativamente el riesgo de cáncer de mama. A pesar de ello, es crucial señalar que la mayoría de las mujeres diagnosticadas no tienen antecedentes familiares conocidos, lo que destaca la necesidad de una comprensión más amplia y detallada de los factores que contribuyen a esta enfermedad. En algunos casos, la consideración de estrategias de reducción del riesgo, como la extirpación quirúrgica de ambos senos, se plantea como una opción para quienes presentan mutaciones genéticas de alto riesgo.

2.3. Manifestaciones patológicas

El cáncer de mama puede manifestarse de diversas maneras, y los síntomas pueden variar, especialmente en etapas avanzadas. En las fases tempranas, la mayoría de las personas pueden **no experimentar síntomas**.

Los posibles síntomas del cáncer de mama incluyen la presencia de un nódulo o engrosamiento en el seno, cambios en tamaño, forma o aspecto del seno, aparición de hoyuelos, enrojecimiento, grietas en la piel, alteraciones en el pezón o la areola, y secreción anómala o sanguinolenta por el pezón.

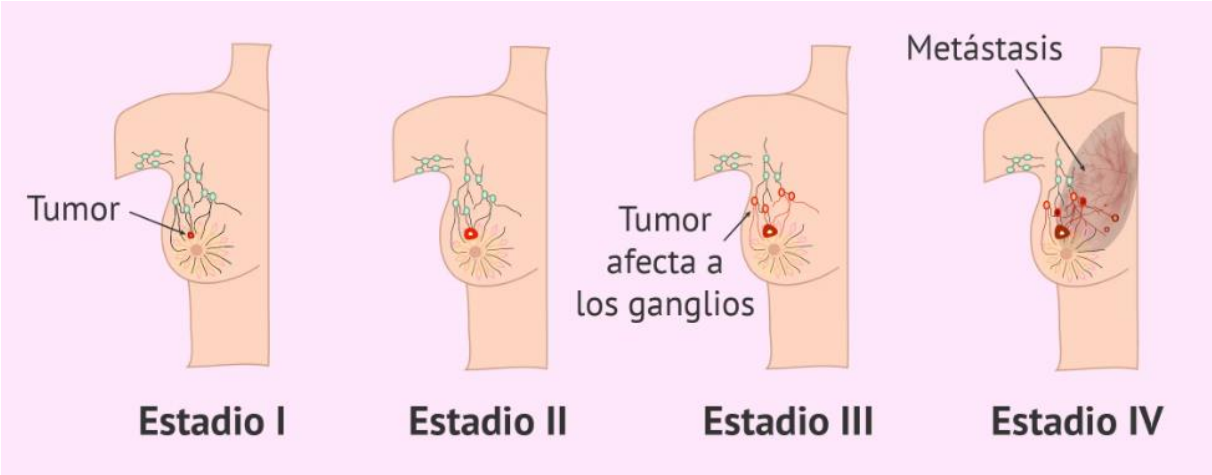
Figura 5. Posibles síntomas del cáncer de mama

Fuente: https://www.cirugiasdelamama.com/cancer-de-mama?lightbox=dataItem-it4rqwkj

Si se detecta un nódulo anómalo es esencial buscar atención médica, incluso si no causa dolor. Cabe destacar que la mayoría de los nódulos en los senos no son cancerosos. La efectividad del tratamiento es mayor cuando los nódulos son pequeños y no se han propagado a los ganglios linfáticos cercanos.

El cáncer de mama puede extenderse, siendo los **ganglios linfáticos de la axila** el lugar común de detección inicial (estadios II o III). Sin embargo, la presencia de ganglios linfáticos cancerosos puede no ser detectable. Con el tiempo, las células cancerosas pueden diseminarse a otras partes del cuerpo, como pulmones, hígado, cerebro y huesos, provocando nuevos síntomas relacionados con el cáncer, como dolor óseo o cefaleas (estadio IV).

Figura 6. Estadios del cáncer de mama



Fuente: https://www.reproduccionasistida.org/cancer-de-mama/estadios-cancer-mama/

2.4. Tratamiento médico

El tratamiento del cáncer de mama es altamente especializado y depende del subtipo específico de la enfermedad, así como de la extensión de su propagación. Los enfoques multidisciplinarios son esenciales para abordar cada caso de manera integral y prevenir posibles recidivas[1]. Este proceso combina diversas modalidades, entre ellas la cirugía, la radioterapia y la administración de medicamentos.

La **cirugía** desempeña un papel crucial y puede implicar desde la extirpación del tejido canceroso hasta la mastectomía total[2]. La actualidad favorece la biopsia del ganglio centinela en lugar de la disección axilar completa, minimizando complicaciones y mejorando los resultados a largo plazo. En casos avanzados, la radioterapia se emplea para abordar tumores residuales y reducir el riesgo de recurrencia, incluso después de la mastectomía.

Los medicamentos seleccionados se basan en pruebas de marcadores tumorales. Los tratamientos hormonales, como el **tamoxifeno**, son fundamentales para cánceres que expresan receptores hormonales, y se administran a lo largo de varios años. Para cánceres

[1] Recidiva, repetición de una enfermedad poco después de terminada la convalecencia.
[2] Cirugía para extirpar toda la mama, incluso el tejido mamario, el pezón, la areola (área de piel oscura alrededor del pezón) y la piel que cubre la mama. También es posible que se extirpen algunos ganglios linfáticos de la axila para determinar si hay cáncer. También se llama mastectomía simple.

que carecen de estos receptores, la quimioterapia desempeña un papel crucial, aunque se evita en casos de tumores muy pequeños.

En casos de cánceres que sobre expresan el oncogén HER2/neu[3], se recurre a fármacos biológicos dirigidos, como el trastuzumab, a menudo combinados con quimioterapia para maximizar la eficacia en la eliminación de células cancerosas. La **radioterapia** se presenta como una herramienta versátil, siendo beneficiosa tanto en estadios tempranos para evitar la mastectomía como en estadios avanzados para reducir el riesgo de mortalidad.

La efectividad de estos tratamientos está intrínsecamente ligada al cumplimiento del ciclo completo, subrayando la importancia de un enfoque integral y oportuno para optimizar los resultados en pacientes con cáncer de mama.

2.5. Optimizando la Identificación del Cáncer de Mama con IA

El estudio de Lång, K. et al., (2023) llevado a cabo con 80.000 mujeres, destaca el impacto positivo de la inteligencia artificial (IA) en la detección del cáncer de mama. Comparado con la lectura tradicional realizada por dos radiólogos, el cribado con IA demostró un **aumento del 20%** en la precisión al identificar casos de cáncer. Publicado en The Lancet Oncology (Lång et al., 2023), el ensayo aleatorizado evidencia el potencial de la IA para hacer más eficaces y precisos estos procedimientos.

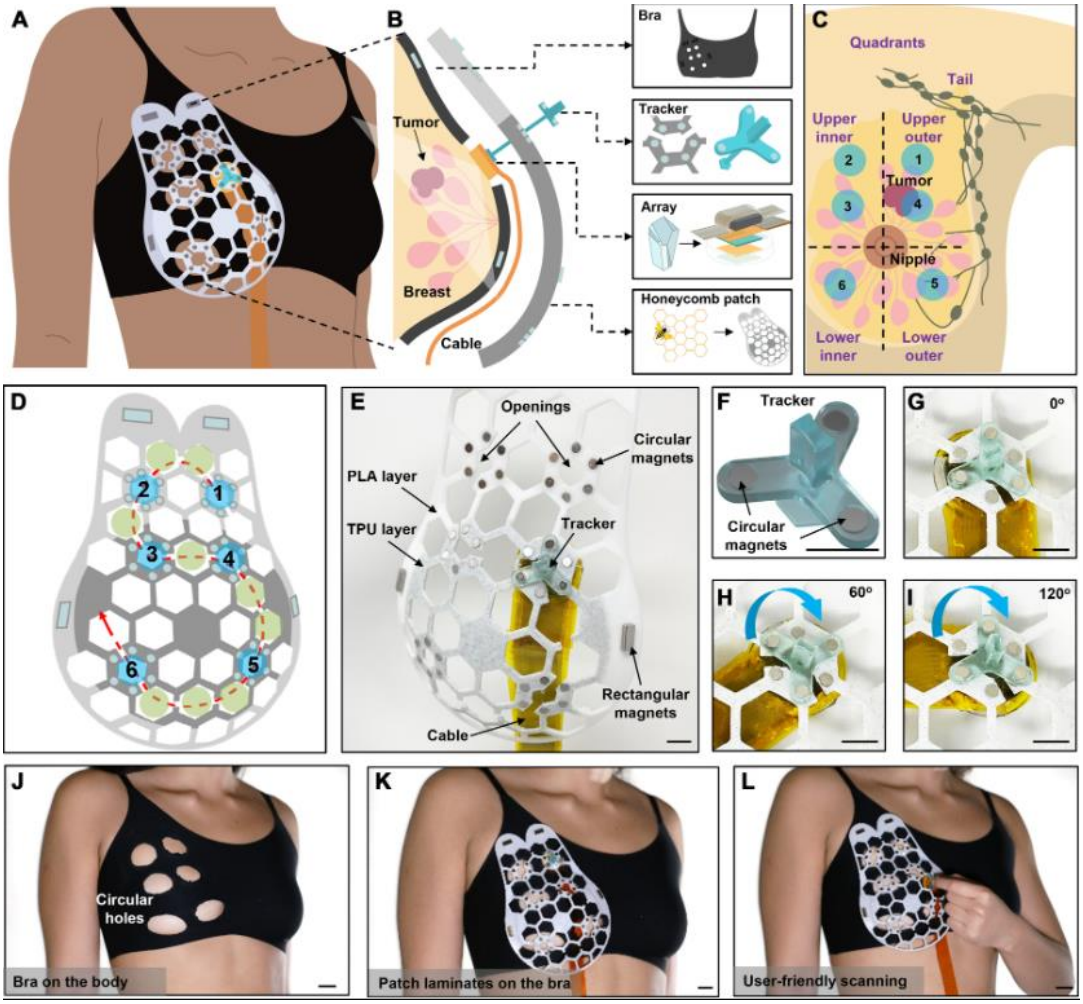
Las participantes se dividieron en dos grupos; mientras el primero seguía la evaluación convencional con dos radiólogos, el segundo se benefició de un cribado inicial con IA seguido de la revisión de un solo radiólogo. Los resultados indican que el grupo con asistencia de IA logró una detección un 20% más precisa, sin aumentar la tasa de falsos positivos. Además, este enfoque podría reducir a la mitad la carga de trabajo de los radiólogos, según los investigadores.

[3] Proteína que participa en el desarrollo normal de las células. Algunos tipos de células cancerosas, como las del cáncer de mama, ovario, vejiga, páncreas, estómago y esófago, producen cantidades anormales del HER2/neu. Es posible que esto haga que las células cancerosas se multipliquen más rápido y se diseminen a otras partes del cuerpo. Medir la cantidad de esta proteína en algunos tipos de células cancerosas sirve para planificar el tratamiento. También se llama c-erbB-2, HER2, receptor 2 del EGF humano y receptor 2 del factor de crecimiento epidérmico humano.

Aunque los resultados son provisionales, destacan la alentadora seguridad de utilizar la IA en el cribado mamográfico y estos resultados podrían respaldar nuevos ensayos y evaluaciones para abordar la escasez de radiólogos en muchos países.

Por otro lado, tenemos la herramienta de Deep Learning desarrollada por el MIT que mencionábamos en párrafos anteriores (Dagdeviren, A. et al., 2023). Este dispositivo, creado mediante impresión 3D, es portátil y flexible, lo que podría revolucionar la detección en mujeres con alto riesgo de cáncer de mama. Utilizando datos de análisis, el sistema examina cambios en el tejido mamario, evitando confusiones con factores hormonales y biológicos. Los resultados preliminares muestran una **reducción del 70% en la carga de trabajo** en comparación con la interpretación de radiólogos en mamografía digital, sugiriendo mejoras significativas en los procesos de detección.

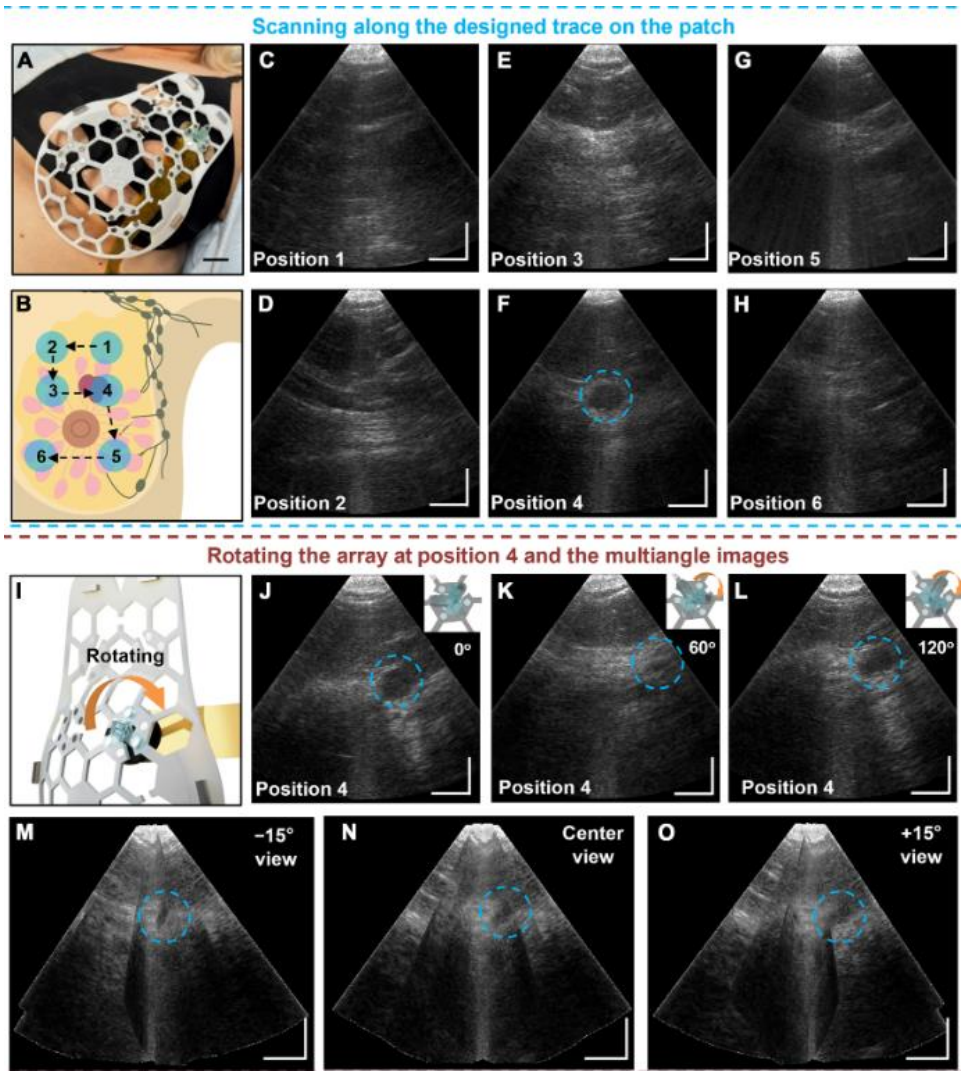
Figura 7. Descripción general del diseño del cUSBr-Patch



Fuente: https://www.science.org/doi/full/10.1126/sciadv.adh5325#sec-4

Aunque el dispositivo es un prototipo, promete permitir a las usuarias realizar controles rutinarios en cualquier momento y lugar, aumentando la frecuencia de los cribados y potencialmente salvando vidas. Liderado por la profesora Canan Dagdeviren (Dagdeviren et al., 2023) del MIT, el proyecto busca **recrear un ecógrafo miniaturizado** utilizando tecnología avanzada de ultrasonidos para obtener y analizar imágenes de tejido mamario de manera menos invasiva. Los ensayos con datos de mamografías del Hospital General de Massachusetts muestran que la IA pudo **predecir con éxito el desarrollo del cáncer en un 31%** de los casos de alto riesgo, superando las técnicas tradicionales (solo pudieron predecir el 18% de estos casos).

Figura 8. Estudio en vivo sobre el tejido mamario



Fuente: https://www.science.org/doi/full/10.1126/sciadv.adh5325#sec-4

Este proyecto busca mejorar tratamientos individuales y proporcionar una herramienta valiosa para especialistas en prevención y tratamiento temprano del cáncer de mama.

3. Objetivos y metodología de trabajo

3.1. Objetivo general

El propósito principal de este Trabajo de Fin de Grado (TFG) es diseñar y desarrollar un sistema de inteligencia artificial destinado a la detección temprana de cáncer de mama en mamografías. Este sistema no solo se centrará en proporcionar resultados precisos en la detección de posibles tumores, sino que también aspira a ofrecer una herramienta integral con aplicaciones más amplias en el ámbito médico.

Esta herramienta podría ser utilizada en contextos tales como:

- **Apoyo a la Toma de Decisiones Clínicas**: El sistema facilitará información valiosa a los profesionales de la salud, mejorando la toma de decisiones clínicas al proporcionar análisis detallados y contextualizados de las mamografías. Esto permitirá una atención médica más precisa y rápida.
- **Monitoreo y Seguimiento Personalizado**: La herramienta permitirá un seguimiento longitudinal de las lesiones detectadas, proporcionando información sobre su evolución temporal. Esto podría ser crucial para establecer estrategias de tratamiento personalizadas y evaluar la efectividad de las intervenciones médicas.
- **Colaboración e Investigación Médica**: La plataforma integrará funcionalidades que faciliten la colaboración entre profesionales de la salud y la investigación médica. Los datos recopilados podrían contribuir a la generación de conocimientos científicos y a la mejora continua de los algoritmos de detección.
- **Concientización y Educación**: La herramienta se utilizará como una plataforma educativa para concienciar a los pacientes sobre la importancia de las mamografías y el cáncer de mama. Además, podría proporcionar información educativa sobre los resultados de las pruebas y las opciones de tratamiento disponibles.
- **Eficiencia en la Asignación de Recursos**: Al facilitar una detección temprana y precisa, la herramienta contribuirá a una asignación más eficiente de recursos médicos al centrarse en casos prioritarios, lo que puede tener un impacto positivo en la calidad de la atención y en la reducción de costos asociados a diagnósticos tardíos.

3.2. Objetivos específicos

Una vez visto el objetivo general de este trabajo, desarrollamos los posibles objetivos específicos del mismo:

- **Optimización de la precisión y pérdida del modelo**: Refinar y ajustar los hiperparámetros del modelo de inteligencia artificial con el objetivo de mejorar la precisión de las predicciones y reducir la pérdida asociada. Se explorarán técnicas de optimización, tales como la selección cuidadosa de tasas de aprendizaje y la adaptación de arquitecturas de red.
- **Investigación de atributos adicionales para profesionales de la salud**: Explorar y evaluar la incorporación de atributos adicionales a las salidas del modelo, más allá de la precisión y la pérdida, que puedan ser de utilidad para los profesionales de la salud. Estos podrían incluir métricas de sensibilidad, especificidad y otras medidas relevantes en el contexto de la detección de cáncer de mama.
- **Implementación de técnicas de procesamiento paralelo con GPU**: Investigar y aplicar técnicas para aprovechar la capacidad de procesamiento de las GPU (Unidades de Procesamiento Gráfico) con el fin de acelerar el entrenamiento y evaluación del modelo. Se buscará mejorar la eficiencia computacional para manejar volúmenes de datos más grandes y complejos.
- **Utilización de recursos en Google Colab para rendimiento mejorado**: Explotar los recursos proporcionados por Google Colab para mejorar el rendimiento del modelo. Se explorará el uso de recursos adicionales, como memoria RAM extendida y unidades de procesamiento tensorial (TPU), para incrementar la capacidad de procesamiento y la velocidad de entrenamiento.
- **Modificaciones dinámicas en las capas del modelo durante la evaluación**: Implementar estrategias de ajuste dinámico en las capas del modelo durante la fase de evaluación. Esto podría incluir técnicas de ajuste fino adaptativas, en las que se modifiquen los pesos de las capas en función del rendimiento del modelo en conjuntos de validación específicos.
- **Reutilización de modelos preentrenados para mejora del rendimiento**: Investigar y aplicar la reutilización de modelos preentrenados, especialmente aquellos entrenados en conjuntos de datos médicos relacionados. Esta técnica permitirá aprovechar el

conocimiento previo de modelos exitosos y adaptarlo a la detección de cáncer de mama, potencialmente mejorando el rendimiento y la generalización del modelo.

- **Evaluación y documentación detallada**: Realizar una evaluación exhaustiva y comparativa de todas las modificaciones implementadas en el modelo, documentando los resultados obtenidos. Se prestará especial atención a la interpretación clínica de las métricas y atributos adicionales incorporados.

3.3. Metodología de trabajo

Al definir los objetivos del proyecto, hemos optado por un enfoque incremental. La **metodología incremental** es un enfoque de desarrollo de proyectos que se caracteriza por abordar el trabajo de manera progresiva y evolutiva, dividiendo el proyecto en etapas más pequeñas y manejables. En lugar de intentar completar todo el proyecto de una sola vez, se divide en partes más pequeñas y manejables que se completan de forma iterativa.

Algunos aspectos clave de una metodología incremental:

División en etapas: El proyecto se divide en una serie de etapas o fases más pequeñas y manejables, cada una de las cuales aborda una parte específica del proyecto.

Desarrollo iterativo: Cada etapa implica un ciclo de desarrollo iterativo en el que se planifica, se implementa, se prueba y se evalúa una parte del proyecto antes de pasar a la siguiente etapa.

Retroalimentación continua: Se fomenta la retroalimentación continua de los usuarios o stakeholders a lo largo del proceso de desarrollo, lo que permite realizar ajustes y mejoras en cada iteración.

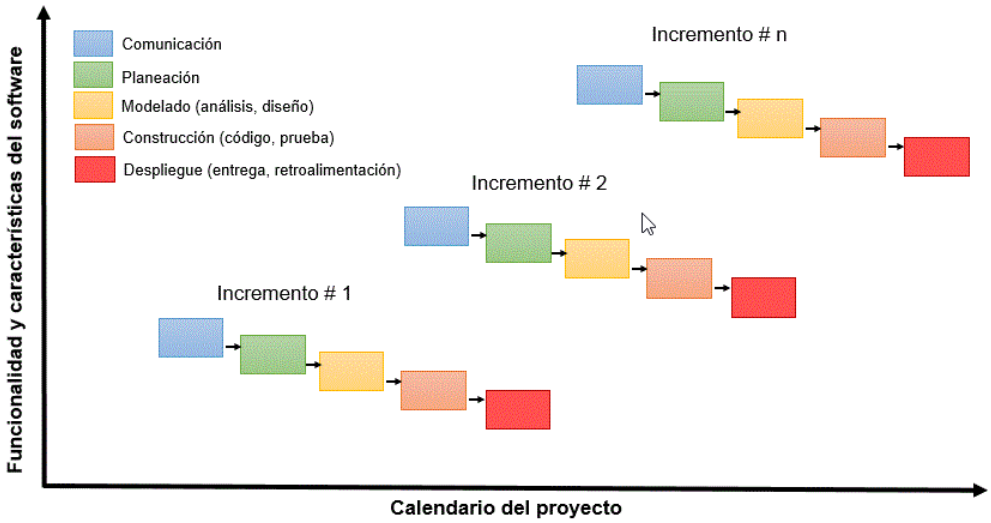
Entrega incremental de funcionalidades: En lugar de esperar hasta que todo el proyecto esté completo para entregar resultados, se entregan funcionalidades o partes del proyecto de forma incremental a medida que se completan.

Flexibilidad y adaptabilidad: La metodología incremental permite una mayor flexibilidad y adaptabilidad a medida que se desarrolla el proyecto, ya que permite responder a cambios en los requisitos o en las condiciones del proyecto de manera más efectiva.

Establecemos como objetivo general el desarrollo de un sistema de inteligencia artificial (IA) para la detección de cáncer de mama a partir de mamografías. Además, nos proponemos

mejorar la precisión y reducir la pérdida del modelo mediante ajustes de hiperparámetros y la aplicación de técnicas como el transfer learning.

Figura 9. Modelo de proceso incremental



Fuente: http://cidecame.uaeh.edu.mx/lcc/mapa/PROYECTO/libro17/12_etapas.html

Nuestro enfoque metodológico adoptado al ser incremental, comenzamos con un script inicial para la generación de un modelo de redes neuronales convolucionales (CNN). Iniciamos el proceso de recopilación de datos explorando diferentes bancos de imágenes DICOM y, posteriormente, procesamos y organizamos los datos en conjuntos de entrenamiento y prueba utilizando scripts en Python.

Para el desarrollo del proyecto, empleamos diversas herramientas y tecnologías, incluyendo Python 3.7, TensorFlow 2.1.0, Keras, CUDA 10.1, cuDNN 7.5, así como los entornos de desarrollo IntelliJ y Google Colab. Además, exploramos arquitecturas CNN pre-entrenadas como MobileNetV2, MobileNetV3, DenseNet e Inception, entre otras.

La recopilación de datos y preparación del conjunto de datos la llevamos a cabo mediante scripts en Python que permiten la descarga, procesamiento y generación de archivos CSV para el entrenamiento del modelo. Este enfoque incremental facilita un desarrollo progresivo del sistema, adaptándose a las necesidades emergentes a lo largo del proyecto.

La evaluación y validación de los modelos se realiza utilizando datos de entrenamiento y prueba generados internamente, complementados con la expansión de datos mediante la biblioteca ImageDataGenerator. A medida que avanzamos en el proyecto, realizamos ajustes y refinamientos en el modelo para mejorar su rendimiento y precisión.

4. Entorno de desarrollo y tecnologías

4.1. Entorno de desarrollo

Para la creación y desarrollo de este proyecto, hemos utilizado IntelliJ IDEA 2023.3.1 en su edición Ultimate como entorno de desarrollo integrado (IDE). La elección de esta herramienta no solo la hemos basado en su reputación como un IDE robusto y eficiente, sino también en la ventaja de acceder a la versión de pago como estudiante universitario. IntelliJ IDEA proporciona un entorno de desarrollo amigable, con una amplia gama de funciones que facilitan la escritura, depuración y optimización del código.

Otra ventaja que tiene es la integración sencilla de herramientas como depuradores, analizadores estáticos y sistemas de control de versiones en un único entorno mejorando significativamente la productividad, ya que en muchas ocasiones (como por ejemplo con eclipse), se necesita una curva de aprendizaje del propio IDE antes de empezar a desarrollar el proyecto. Además, su compatibilidad con una variedad de lenguajes de programación, como Python, Java, etc. y la posibilidad de personalización han hecho de IntelliJ IDEA la elección ideal.

A continuación, se presenta una tabla comparativa detallada con las ventajas y desventajas de cada uno:

Tabla 1.

Comparativa de IDEs de desarrollo

Características	IntelliJ IDEA	Eclipse	NetBeans	VSCode
Facilidad de Uso	Interfaz intuitiva y amigable, fácil de aprender.	Requiere configuración inicial, curva de aprendizaje.	Interfaz sencilla y fácil de navegar.	Ligero y rápido, fácil de configurar.
Desempeño	Excelente rendimiento y velocidad de compilación.	Rendimiento sólido, pero puede ser más lento que IntelliJ.	Buena velocidad, pero puede ralentizarse con proyectos grandes.	Ligero y rápido, especialmente en proyectos pequeños.
Soporte de Lenguajes	Amplio soporte para Java y otros lenguajes.	Inicialmente diseñado para Java, requiere extensiones para otros lenguajes.	Soporte para Java, pero menos extensible para otros lenguajes.	Compatible con numerosos lenguajes gracias a extensiones.
Funcionalidades Avanzadas	Funciones avanzadas y herramientas de refactorización.	Amplia gama de plugins y extensiones.	Herramientas avanzadas, pero menos extensibles.	Funciones básicas, extensible con extensiones.
Integración con Herramientas Externas	Integración completa con herramientas populares.	Requiere configuración para algunas herramientas.	Integración adecuada, pero menos que IntelliJ.	Extensible y compatible con varias herramientas.

Características	IntelliJ IDEA	Eclipse	NetBeans	VSCode
Licencia y Costo	Versión Ultimate de pago, pero gratuita para estudiantes.	Código abierto y gratuito.	Código abierto y gratuito.	Gratuito y de código abierto.

4.2. Banco de imágenes

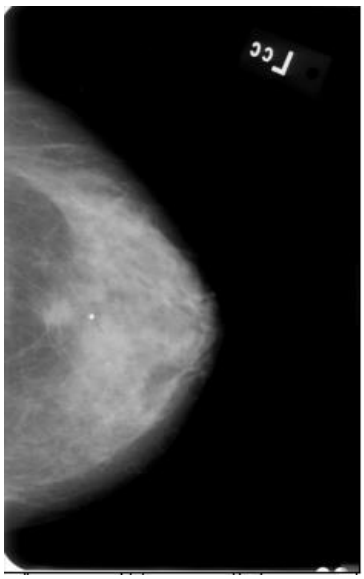
Existen numerosos repositorios en internet donde se pueden descargar imágenes mamográficas con formato DICOM. El formato DICOM (Digital Imaging and Communications in Medicine) es un estándar utilizado en el ámbito médico para el intercambio, almacenamiento y transmisión de imágenes médicas. Fue desarrollado por la American College of Radiology (ACR) y la National Electrical Manufacturers Association (NEMA) para estandarizar la comunicación y manejo de datos de imágenes médicas entre dispositivos de diferentes fabricantes.

Las imágenes médicas en formato DICOM pueden incluir radiografías, tomografías computarizadas (TC), resonancias magnéticas (RM), ecografías y otros tipos de imágenes utilizadas en el diagnóstico y tratamiento médico. Siendo estas imágenes el centro del TFG cabe mencionar algunas características fundamentales de su formato:

- **Estructura Jerárquica**: Las imágenes DICOM no solo contienen la imagen en sí, sino también información adicional relacionada con el paciente, la institución médica, el dispositivo de adquisición de la imagen, parámetros de adquisición, entre otros. La información se organiza de manera jerárquica en un conjunto de datos DICOM.
- **Metadatos Estándar**: Cada archivo DICOM incluye una serie de metadatos que describen la información asociada con la imagen. Esto permite que las imágenes sean fácilmente identificables y comprensibles por sistemas de visualización y análisis.
- **Compatibilidad Internacional**: El estándar DICOM es ampliamente utilizado a nivel internacional, lo que facilita el intercambio de información médica entre diferentes dispositivos y sistemas de salud.

- **Soporte para Diferentes Modalidades de Imágenes**: DICOM es compatible con diversas modalidades de imágenes médicas, lo que significa que puede utilizarse para almacenar y compartir imágenes provenientes de diferentes equipos médicos.
- **Seguridad y Privacidad**: DICOM incluye características para proteger la privacidad de los pacientes, como la capacidad de almacenar información de forma anónima y el soporte para funciones de seguridad.

Figura 10. Mamografía en formato DICOM



Fuente: Extraído del dataset utilizado en el TFG

Para poder realizar este trabajo, hemos utilizado las imágenes de cancerimagingarchive (https://wiki.cancerimagingarchive.net/pages/viewpage.action?pageId=22516629), donde se encuentran disponibles mamografías de alta calidad. Este recurso ofrece una amplia variedad de imágenes mamográficas de diversos casos clínicos, proporcionando un conjunto de datos representativo y valioso para poder entrenar la red neuronal del proyecto. La accesibilidad, diversidad y previo etiquetado de casos en este banco de imágenes han sido cruciales para garantizar la robustez y la generalización del modelo.

4.3. Tecnologías

El desarrollo del script que entrena la red neuronal con las imágenes vistas en el apartado anterior se ha llevado a cabo mediante la utilización de diversas tecnologías que han demostrado ser fundamentales en el ámbito de la investigación médica y el procesamiento de imágenes. En primer lugar, se ha empleado Python como lenguaje principal debido a su

versatilidad y amplia comunidad de desarrollo. Además, las bibliotecas como os, pandas, pydicom, y numpy han facilitado la manipulación y procesamiento eficiente de datose imágenes DICOM. La integración de scikit-image (skimage) ha sido esencial para la transformación y mejora de las imágenes mamográficas, mientras que scikit-learn ha proporcionado herramientas clave para la división de datos y la codificación de etiquetas.

Para la implementación del modelo de aprendizaje profundo, TensorFlow y Keras han sido herramientas cruciales, permitiendo la construcción eficiente de **redes neuronales convolucionales**. La manipulación y optimización eficientes de las imágenes se han logrado mediante el uso de las bibliotecas concurrent.futures y skimage. Además, se han utilizado herramientas como time y psutil para monitorear y gestionar eficazmente los recursos del sistema durante la ejecución del script. Las operaciones de lectura y escritura de archivos CSV se han simplificado gracias a las bibliotecas csv y codecs, mientras que pixel_data_handlers de pydicom ha mejorado el manejo de datos en formato DICOM. Y finalmente, se han utilizado ast y json para procesar y evaluar literales de cadena de manera segura.

5. Diseño de la solución

5.1. Preparación del entorno, descarga de las imágenes y ciencia de datos

Se necesita una cantidad de imágenes mamográficas considerable con la que poder entrenar nuestra red neuronal, como hemos visto en el apartado anterior. Para ello se ha decidido utilizar el repositorio de imágenes de cancerimagingarchive:

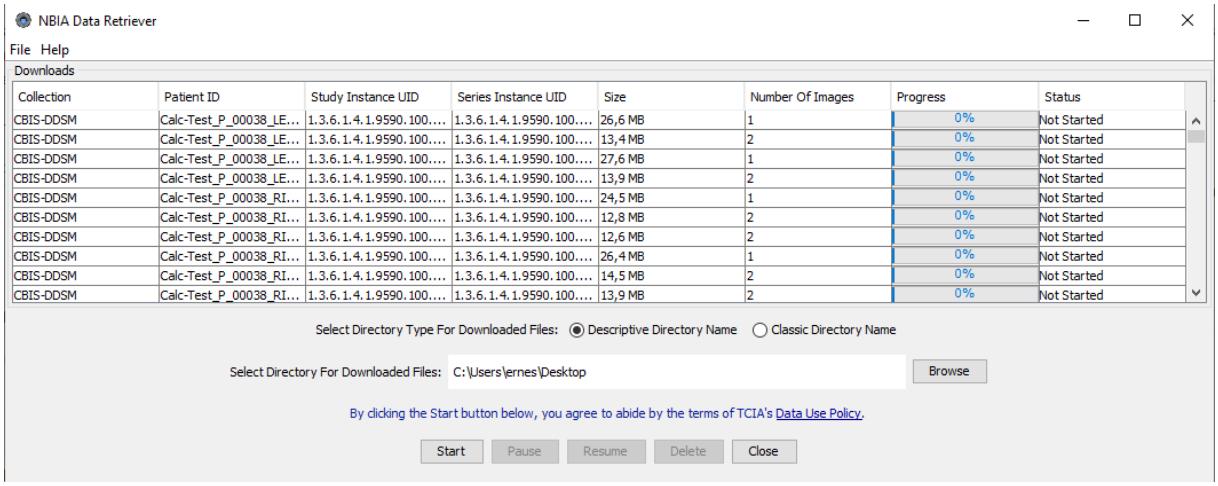
Figura 11. Conjunto de imágenes DICOM para descargar

Data Type	Download all or Query/Filter
Images (DICOM, 163.6GB)	Download Search (Requires the NBIA Data Retriever .)

Fuente: https://wiki.cancerimagingarchive.net/pages/viewpage.action?pageId=22516629#225166296af7157f6e4641c286eee03e498e9305

Necesitamos del software **NBIA Data Retriever** para poder descargar este conjunto de imágenes, una aplicación con interfaz gráfica que nos facilita la descarga de conjuntos de datos desde el repositorio NBIA. Con este software podemos buscar, seleccionar y descargar imágenes médicas y datos asociados almacenados:

Figura 12. Descarga de imágenes DICOM utilizando NBIA Data Retriever



Una vez que tenemos nuestro conjunto de imágenes descargado en la ruta de nuestro proyecto, vamos a proceder al simplificado de las rutas para poder trabajar con los datos de una manera más cómoda y eficiente.

5.1.1. Normalizado de rutas para poder realizar el estudio

Cuando se ha terminado de descargar el conjunto de imágenes DICOM lo encontramos con la siguiente estructura de carpetas:

Figura 13. Estructura del directorio de imágenes DICOM descargada

Como podemos observar en la imagen se nos crea una carpeta raíz llamada *CBIS-DDSM* y a partir de ahí tenemos todas las imágenes estructuradas por paciente con la siguiente nomenclatura, ejemplo: **Calc-Test_P_00038_LEFT_CC**

- **Calc**: Hallazgos que se encuentran en las mamografías y los podemos dividir en:
    - **Calc**: El hallazgo encontrado en la imagen DICOM es una calcificación, las cuales son depósitos de minerales en el tejido mamario que pueden ser benignas o malignas. Las malignas pueden estar asociadas al Cáncer mientras que las benignas no.
    - **Mass**: El hallazgo encontrado en la imagen DICOM es una masa, las cuales son áreas anormales de tejido que pueden ser sólidas o líquidas. La presencia de masas podrían ser tumores y la evaluación de estas es crucial para determinar si son benignas o malignas.
- **Test**: nos indica que es una imagen que se utiliza para realizar en la parte de test del modelo. Los posibles valores son: **Test** y **Trainning**.
- **P_00038**: Es el identificativo anonimizado del paciente.
- **LEFT**: Se refiere a la mama izquierda (valores posibles **LEFT**, **RIGHT**).
- **CC**: es el tipo de proyección o vista desde la que se toman las mamografías de los senos. A continuación, se describen todas las posibles proyecciones:
    - **Craniocaudal** (CC): esta proyección se realiza al tomar imágenes desde arriba hacia abajo, perpendicular al cuerpo. En una mamografía CC, el paciente se coloca de pie frente a la máquina de mamografía, y la imagen se toma proyectando el rayo hacia el seno desde la parte superior, siendo esta vista útil para evaluar la parte interna del seno.

- **Mediolateral Oblique** (MLO): Esta proyección implica una inclinación del tubo de rayos X en un ángulo oblicuo. El paciente está de pie o sentado de lado con el seno colocado en una posición oblicua, siendo esta imagen útil para evaluar el tejido mamario en el área del pezón y la parte externa del seno.
- **Lateral** (L): La proyección lateral se toma desde el costado del seno. El paciente puede estar de pie o sentado con el seno expuesto, y la imagen se toma desde el lateral, proporcionando una visualización detallada de la parte externa del seno, lo que es útil para evaluar lesiones o anomalías en esa área.
- **Axilar** (AX): La proyección axilar se centra en el área de la axila. El paciente puede estar de pie o sentado, y la imagen se toma para visualizar los ganglios linfáticos axilares, siendo útil para evaluar la presencia de ganglios linfáticos agrandados o cualquier otra anormalidad en la región axilar.
- **Proyección de Comprensión Magnificada** (Mag CC o Mag MLO): Esta proyección se toma con compresión, similar a la CC o MLO estándar, pero se enfoca en áreas específica proporcionando una visión ampliada de dichas áreas, lo que ayuda a evaluar con mayor detalle lesiones sospechosas o anormalidades.
- **Proyección de Perfil** (ML o Lateral Medial): Se toma desde el perfil lateral del seno. El paciente puede estar de pie o sentado, y la imagen se toma desde un ángulo lateral, siendo útil para evaluar el tejido mamario en el perfil lateral, proporcionando información adicional sobre posibles lesiones.

Aunque aquí se han descrito todas las posibles variantes de proyecciones en el conjunto de datos solo vamos a trabajar con Craniocaudal (**CC**) y Mediolateral Oblique (**MLO**).

A continuación, y mediante el siguiente script realizado en Python, vamos a buscar todas las imágenes DICOM que se encuentren por debajo de la carpeta principal que se genera una vez descargado el repositorio (*CBIS-DDSM*):

```
1.  import os
2.  import shutil
3.  import glob
4.
5.  ruta_origen = "D:\\Documentos\\0.-Universidad Ingeniería
    Informática\\TFG\\proyecto\\src\\resources\\CBIS-DDSM"
6.  rutas_archivos = glob.glob(ruta_origen + "\\*")
7.
8.  def buscar_y_copiar_imagenes(ruta):
9.      # Lista para almacenar las rutas de las imágenes .dcm encontradas
10.     imagenes = []
11.
12.     # Buscamos archivos de imagen (dcm) en la ruta de origen y subdirectorios
13.     for root, dirs, files in os.walk(ruta):
14.         for file in files:
15.             if file.endswith('.dcm'):
16.                 imagenes.append(os.path.join(root, file))
17.
18.     print("Imágenes encontradas:")
19.     for imagen in imagenes:
20.         print(imagen)
21.         # Copiar la imagen a la ruta destino
22.         shutil.copy(imagen, ruta)
23.         print("Imagen copiada a:", ruta)
24.
25.
26. # Iteramos sobre todas las rutas encontradas
27. for ruta in rutas_archivos:
28.     buscar_y_copiar_imagenes(ruta)
```

La descripción detallada de cómo funciona este código se encuentra en el Anexo A.

Una vez que tenemos normalizadas las rutas el siguiente paso es realizar la carga de los datos de cáncer de masa y de calcificación expresados en los csv que vienen en el repositorio descargado (llamados mass_case_description_train_set.csv y mass_case_description_test_set.csv para la masa y calc_case_description_train_set.csv y calc_case_description_test_set.csv para la calcificación) y relacionarlos con las imágenes DICOM correspondientes.

5.1.2. Relación de csv de entrenamiento y test con sus correspondientes imágenes DICOM

Ahora, vamos a relacionar las imágenes DICOM que hemos estructurado en el directorio de carpetas con sus valores principales que nos van a servir para entrenar nuestro modelo CNN. Estos valores principales pueden ser:

- **MALIGNANT** (Maligno): Esta etiqueta indica que la lesión o hallazgo en la imagen se ha identificado como maligno, lo que significa que se considera potencialmente canceroso.
- **BENIGN** (Benigno): Indica que la lesión o hallazgo en la imagen se ha identificado como benigno, lo que significa que no se considera canceroso y generalmente no representa una amenaza para la salud.
- **BENIGN WITHOUT CALLBACK** (Benigno sin seguimiento): Esta etiqueta sugiere que la lesión o hallazgo en la imagen se ha identificado como benigno, y no se requiere un seguimiento adicional ya que no se considera una preocupación médica.
- **MALIGNANT WITHOUT CALLBACK** (Maligno sin seguimiento): Indica que la lesión o hallazgo en la imagen se ha identificado como maligno, pero no se considera necesario realizar un seguimiento adicional. Esto puede deberse a que la lesión ya ha sido tratada o evaluada de manera exhaustiva.

Vamos a utilizar los csv proporcionados por la propia página de cancerimagingarchive para llevar esto a cabo, donde encontramos los atributos de las imágenes de cada paciente. Podemos observar la siguiente estructura dentro de estos csv utilizando la librería de pandas:

Tabla 2.

Campos relevantes en mass_case_description_train_set.csv

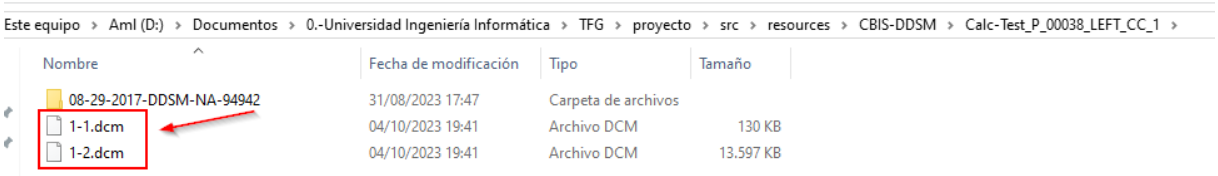
patient_id	pathology	image file path
P_00001	MALIGNANT	Mass-Training_P_00001_LEFT_CC/1.3.6.1.4.1.9590.100.1.2.422112722213189649807611434612228974994/1.3.6.1.4.1.9590.100.1.2.342386194811267636608694132590482924515/000000.dcm
P_00001	MALIGNANT	Mass-Training_P_00001_LEFT_MLO/1.3.6.1.4.1.9590.100.1.2.319478999311971442426185353560182990988/1.3.6.1.4.1.9590.100.1.2.359308329312397897125630708681441180834/000000.dcm
P_00004	BENIGN	Mass-Training_P_00004_LEFT_CC/1.3.6.1.4.1.9590.100.1.2.347107867812656628709864319310977895697/1.3.6.1.4.1.9590.100.1.2.891800462110225318343526314836693465 40/000000.dcm

Nota. Para ver el número de columnas completo consultar Anexo B

La estructura de los otros tres csv (mass_case_description_test_set.csv, calc_case_description_train_set.csv y calc_case_description_test_set.csv) es igual que la mencionada en la tabla ya que la única diferencia es que los datos están separados entre entrenamiento y test de masa y calcificación.

Llegados a este punto, observamos que en la columna "**image file path**" todas las rutas acaban con el nombre de la imagen 000000.dcm (como hemos podido observar en la parte de normalización de rutas, los archivos DICOM, tienen nombres como 1-1.dcm o 1-2.dcm), por lo que tenemos que relacionar los datos de los csv con el nombre de la imagen real para luego poder entrenar nuestro modelo y realizar la parte de pruebas:

Figura 14. Nombre real de los archivos DICOM



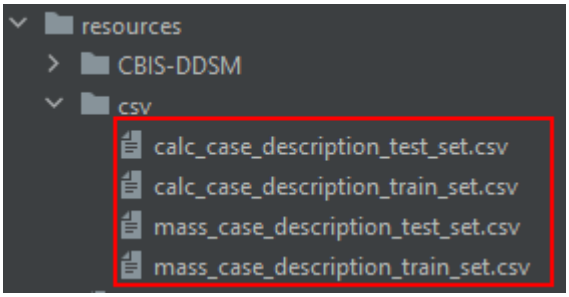
Desarrollamos el siguiente código en Python para poder hacer esta relación entre los datos de los CSV y sus imágenes reales:

```
# Definir la ruta base donde se encuentran las imágenes
base_image_path = "D:/Documentos/0.-Universidad Ingeniería Informática/TFG/proyecto/src/resources/CBIS-DDSM"

# Cargar datos de cáncer de masa
df_mass_train = pd.read_csv('./resources/csv/mass_case_description_train_set.csv')
df_mass_test = pd.read_csv('./resources/csv/mass_case_description_test_set.csv')

# Cargar datos de cáncer de calcificación
df_calc_train = pd.read_csv('./resources/csv/calc_case_description_train_set.csv')
    df_calc_test = pd.read_csv('./resources/csv/calc_case_description_test_set.csv')

    # Combinar los datos de entrenamiento y prueba
    df_train = pd.concat([df_mass_train, df_calc_train], ignore_index=True)
    df_test = pd.concat([df_mass_test, df_calc_test], ignore_index=True)
```

Lo primero que realizamos es definir la ruta base donde se encuentran las imágenes, que como hemos visto en apartados anteriores se encuentra en CBIS_DDSM.

Acto seguido, cargamos los datos de cáncer relacionados tanto con masa como con calcificaciones, divididos además en conjuntos de entrenamiento y prueba. Para ello utilizamos la librería de **pandas** y la función **read_csv** pasándole como parámetro la ruta relativa que se encuentra dentro del proyecto y donde podemos encontrar dichos csv que representan los conjuntos descritos.

Figura 15. Ruta relativa dentro del proyecto donde se encuentran los conjuntos de entrenamiento y pruebas en formato csv



Posteriormente, utilizamos la función **pd.concat** de pandas para combinar los conjuntos de datos de entrenamiento y prueba. Los argumentos son:

- **df_mass_train** y **df_calc_train**: DataFrames relacionados con el cáncer de mama de tipo masa y calcificación para el conjunto de entrenamiento, respectivamente.
- **df_mass_test** y **df_calc_test**: DataFrames relacionados con el cáncer de mama de tipo masa y calcificación para el conjunto de prueba, respectivamente.
- **ignore_index=True**: Este parámetro restablece los índices del DataFrame resultante para evitar problemas con índices duplicados.

Debido a la elevada carga computacional asociada con la carga individual de cada imagen DICOM en el proyecto, se ha tomado la decisión de consolidar los datos correspondientes a estas imágenes junto con sus respectivas clasificaciones (maligno, benigno, etc.) en dos archivos CSV (resultadosXytrain.csv y resultadosXytest.csv). Estos archivos contendrán la información completa de las 5824 imágenes. Esta estrategia nos permitirá simplificar futuras fases del proyecto, ya que podremos cargar directamente estos archivos CSV en lugar de procesar nuevamente cada imagen al realizar pruebas con distintos modelos de redes neuronales o sistemas de inteligencia artificial. De esta manera, optimizamos el rendimiento del proyecto y agilizamos el proceso de desarrollo, evitando la carga computacional innecesaria al acceder a los datos requeridos de manera más eficiente.

Continuando con el código, procedemos a desarrollar dicha parte:

```
# Cargar imágenes y etiquetas de entrenamiento y prueba y guardarlas en csv
X_train, y_train, paths_train = load_images_and_labels(df_train, base_image_path)
csv_filenameTrain = 'resultadosXytrain.csv'
save_to_csv(X_train, y_train, paths_train, csv_filenameTrain)

X_test, y_test, paths_test = load_images_and_labels(df_test, base_image_path)
csv_filenameTest = 'resultadosXytest.csv'
save_to_csv(X_test, y_test, paths_test, csv_filenameTest)
```

Cargamos las imágenes y etiquetas de entrenamiento y prueba:

Utilizamos la función ***load_images_and_labels*** para cargar las imágenes y etiquetas de entrenamiento (esta función se describirá más adelante). *df_train* y *df_test* son nuestros DataFrame los cuales contienen información sobre las imágenes de entrenamiento y prueba, y *base_image_path* es la ruta base donde se encuentran las imágenes.

La función devuelve tres matrices: *X_train* e *y_*train que contienen las imágenes y las etiquetas respectivamente para el conjunto de entrenamiento, *X_test* e *y_*test para el conjunto de pruebas y *paths_train* y *paths_*test que contienen las rutas de las imágenes.

Guardar los datos de entrenamiento en archivos CSV:

Empleamos la función ***save_to_csv*** para guardar las imágenes, etiquetas y rutas de entrenamiento en un archivo CSV. Esta función toma como argumentos las matrices *X_train*, *y_train* y *paths_train*, junto con el nombre del archivo CSV (csv_filenameTrain).

Consecuentemente con esta parte, pasamos a describir el desarrollo de la función ***load_images_and_labels***:

```
def load_images_and_labels(df, base_path, batch_size=100):
    images_path = []
    labels = []
    paths = []  # Agrega una lista para almacenar las rutas completas

    with ThreadPoolExecutor(max_workers=5) as executor:
        for _, row in df.iterrows():
            path_paciente = row['image file path']
            image_path = os.path.join(base_path, path_paciente.split('/')[0])
                image_path = image_path.replace("/", "\\")
                images_path.extend(search_images(image_path))

          # Cargar imágenes en paralelo
          loaded_images = list(executor.map(load_image, images_path))

```

```
16.             # Filtrar las imágenes que no pudieron ser cargadas
17.             valid_indices = [i for i, img in enumerate(loaded_image
    s) if img is not None]
18.             loaded_images = [img for i, img in enumerate(loaded_ima
    ges) if i in valid_indices]
19.             paths = [path for i, path in enumerate(images_path) if
    i in valid_indices]
20.
21.             # Obtener las etiquetas de las imágenes válidas
22.             labels = df['pathology'].apply(lambda x: 1 if 'MALIGNAN
    T' in x else (2 if 'BENIGN' in x else (3 if 'BENIGN WITHOUT
    CALLBACK' in x else (4 if 'MALIGNANT WITHOUT
    CALLBACK' in x else 0))))
23.             labels = labels.values.astype(np.float32)
24.             labels = labels[valid_indices]
25.
26.             # Nos aseguramos de que todas las imágenes tengan la
    forma adecuada
27.             loaded_images = [img.reshape(128, 128, 1) for img in lo
    aded_images]
28.
29.         return np.array(loaded_images), labels, paths
```

Inicializamos tres listas vacías: *images_path* para almacenar las rutas de las imágenes, *labels* para almacenar las etiquetas, y *paths* para almacenar las rutas completas de las imágenes.

Seguidamente, en el bucle *while* utilizamos ***ThreadPoolExecutor*** para ejecutar operaciones en paralelo. En este caso, utilizando el valor *max_workers=5* cargaremos hasta cinco imágenes a la vez utilizando distintos hilos, pero sin sobrecargar la RAM de la CPU. Dicho bucle, itera sobre las filas del *DataFrame df* y obtiene la ruta del paciente (path_paciente) de la columna *'image file path*'. Construimos la ruta completa de la imagen y la convertimos a un formato compatible con Windows (*replace("/", "\\")*). Extendemos la lista *images_path* con las rutas de las imágenes obtenidas llamando a la función ***search_images***, con la que buscamos imágenes DICOM en la ruta proporcionada.

Aplicamos ***executor.map*** para cargar las imágenes en paralelo utilizando la función ***load_image*** (esta función se describirá más adelante) en la lista de rutas de imágenes *images_path*.

Adicionalmente, creamos una lista de índices válidos (*valid_indices*) que corresponden a las imágenes que se cargaron correctamente y filtramos las listas *loaded_images* y *images_path* para mantener solo las imágenes y rutas correspondientes a los índices válidos. Este paso es esencial debido a que algunas de las imágenes de los conjuntos de datos descargados contenían algún tipo de error en sus datos de origen y no fue posible su lectura.

Hicimos uso de la columna '*pathology*' del *df* descrito en la anterior tabla, para asignar etiquetas numéricas basadas en la presencia de ciertas palabras clave como 'MALIGNANT', 'BENIGN', 'BENIGN WITHOUT CALLBACK'y 'MALIGNANT WITHOUT CALLBACK'. Convertimos las etiquetas a un array NumPy de tipo float32 y filtramos las mismas para mantener solo aquellas correspondientes a las imágenes válidas.

Finalmente, nos aseguramos de que todas las imágenes tengan una forma específica: 128x128x1 (este ajuste de tamaño se ha realizado en un paso previo dentro de la función **load_image**) y devolvemos en el *return* un array NumPy de imágenes (*loaded_images*), las etiquetas correspondientes (*labels*), y las rutas completas de las imágenes (*paths*).

Ahora vamos con la función ***search_images***:

```
1. def search_images(ruta):
2.     imagenes = []
3.
4.     # Busca archivos de imagen (jpg, dcm, jpeg, png, gif) en la
   ruta de origen y subdirectorios
5.     for root, dirs, files in os.walk(ruta):
6.
7.         # Este if lo hacemos por cómo está estructurada la
   arquitectura de carpetas y dcm, si no está en la raíz que no lo
   coja.
8.         # Previamente ya se ha realizado un copiado de todas
   las imagenes .dcm a la carpeta que nos interesa con un script
9.         if(len(root.split("\\")) <= 9):
10.                 for file in files:
11.                     if file.endswith('.dcm'):
12.                         imagenes.append(os.path.join(root, fi
   le))
13.
14.          return imagenes
```

Inicializamos una lista vacía llamada *imagenes* para almacenar las rutas de las imágenes encontradas.

Utilizamos ***os.walk*** para recorrer la ruta y sus subdirectorios buscando archivos de tipo imagen. Con esta función devolvemos un generador que produce tuplas de la forma (*dirpath*, *dirnames*, *filenames*) para cada directorio encontrado durante el recorrido.

A continuación, verificamos si la longitud de la lista resultante de dividir la ruta por "\" es menor o igual a 9. Con este *if* cogemos solo las imágenes que fueron copiadas a la raíz de la carpeta de cada paciente. Si el número de niveles en la estructura de carpetas es menor o igual a 9, se procede a buscar imágenes DICOM.

Por último, iteramos sobre los archivos en el directorio actual (*files*) y agregamos las rutas completas de aquellos archivos que terminan con la extensión '.dcm' a la lista *imágenes*, devolviendo esta lista con las imágenes DICOM encontradas en la ruta y sus subdirectorios.

Procedemos ahora a la descripción de la función desarrollada ***load_image***:

```
1. def load_image(image_path):
2.
3.     # Para ver la matriz total declaramos estas opciones
4.     pd.set_option('display.max_rows', None)
5.     pd.set_option('display.max_columns', None)
6.     np.set_printoptions(threshold=np.inf)
7.     print("Cargando imagen", image_path)
8.
9.     try:
10.             ds = pydicom.dcmread(image_path)
11.
12.             # Configurar PyDicom para que utilice el
    manejador pillow
13.             pydicom.config.image_handlers = [pillow_handler]
14.
15.             image = ds.pixel_array.astype(np.float32)
16.
17.             # Normalizar los valores entre 0 y 1
18.             image = (image -
    np.min(image)) / (np.max(image) - np.min(image))
19.
20.             image = exposure.equalize_adapthist(image)
21.             image_resized = resize(image, (128, 128), mode='r
    eflect', anti_aliasing=True)
22.
23.             return image_resized
24.         except Exception as e:
25.             print(f"Error al procesar la imagen {image_path}:
    {e}")
26.             return None  # Retorna None para indicar que ha
    ocurrido un error
```

Previamente a realizar la lógica del código, establecemos las opciones de visualización para **pandas** y **NumPy**. En este caso, configuramos la salida para mostrar todas las filas y columnas sin truncar y pintamos un mensaje en la consola indicando que se está cargando la imagen cuya ruta es *image_path*.

A continuación, utilizamos la biblioteca **PyDicom** para leer el archivo DICOM especificado por *image_path* y lo configuramos para que utilice el manejador *pillow_handler*, pudiendo así trabajar con imágenes.

Extraemos la matriz de píxeles de la imagen DICOM y la convertimos a tipo ***float32*** y normalizamos los valores de píxeles entre 0 y 1.

Como en los siguientes pasos vamos a redimensionar las imágenes DICOM, aplicamos una mejora adaptativa del histograma utilizando la función ***equalize_adapthist*** de la biblioteca **scikit-image**.

Finalmente, redimensionamos la imagen a un tamaño específico de 128x128 píxeles utilizando la función ***resize*** de **scikit-image** y devolvemos en el *return* la imagen procesada y redimensionada.

Si ocurre alguna excepción durante el procesamiento de la imagen, imprimimos un mensaje de error en la consola y devolvemos un *None* para indicar que ha ocurrido un error. Esto nos será útil para identificar imágenes que no pudieron ser procesadas correctamente.

Una vez descritas todas estas funciones para buscar y cargar imágenes, solo nos queda explicar la función ***save_to_csv*** que nos permite guardar en un csv todos los datos de las imágenes junto con su etiqueta y su path:

```
1. def save_to_csv(images, labels, paths, csv_filename):
2.     # Concatenar las imágenes en una matriz tridimensional
3.     images_array = np.stack(images, axis=0)
4.
5.     # Crear un DataFrame con las imágenes y etiquetas
6.     df = pd.DataFrame({'Image': images_array.tolist(), 'Label':
   labels, 'Path': paths})
7.
8.     # Guardar el DataFrame en un archivo CSV
9.     df.to_csv(csv_filename, index=False, sep=';')
```

Empezamos utilizando la función ***np.stack*** de **NumPy** para concatenar las imágenes almacenadas en la lista *images* en una matriz tridimensional (*images_array*). El argumento *axis=0* indica que la concatenación se realiza a lo largo del eje 0, creando así una nueva dimensión.

Creamos un DataFrame de pandas con tres columnas: '**Image**', '**Label**' y '**Path**'.

- La columna '**Image**' contiene las matrices tridimensionales de las imágenes convertidas a listas para que puedan ser almacenadas en el DataFrame.
- La columna '**Label**' contiene las etiquetas (MALIGNANT, BENIGN, etc) asociadas a las imágenes.
- La columna '**Path**' contiene las rutas completas de las imágenes.

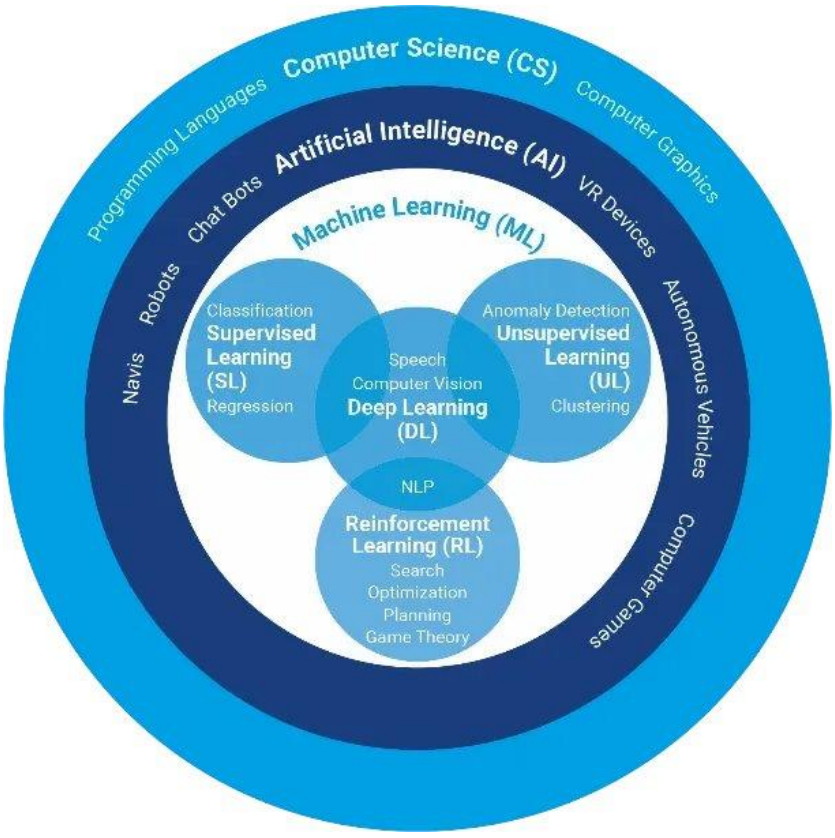
Implementamos la función ***to_csv*** de pandas para guardar el DataFrame en un archivo CSV especificado por *csv_filename*.

- El argumento *index=False* indica que no se debe incluir la columna de índices en el archivo CSV.
- El argumento sep=';' especifica que el separador de campos en el archivo CSV será el punto y coma (';'). **Este argumento es imprescindible y muy importante**, ya que, si no, por defecto guardará la separación de campos con ‘,’ y este carácter es el que se utiliza para separar los valores de las matrices tridimensionales que definen cada imagen provocando así errores en la compilación del modelo CNN.

5.2. Generación de un modelo CNN

Vamos a contextualizar en que parte de las ciencias de la computación nos encontramos antes de comenzar a desarrollar nuestro modelo de CNN y que conjunto de algoritmos vamos a utilizar dentro de la Inteligencia Artificial, conocidos como **Deep Learning** o Aprendizaje profundo.

Figura 16. Diagrama de las distintas capas que existen dentro de las ciences de la computación



Fuente: https://juandomingofarnos.wordpress.com/2023/05/08/aprendizaje-profundo-y-redes-neuronales-de-juan-domingo-farnos-en-procesos-algoritmicos-de-aprendizaje-con-eduacion-disruptiva-inteligencia-artificial/

Si seguimos el diagrama de la Figura 15 de forma descendente, podemos observar que:

Las **ciencias de la computación** es el campo amplio que estudia los fundamentos teóricos y prácticos de la información y la computación. Incluye disciplinas como algoritmos, estructuras de datos, teoría de la computación y sistemas operativos. La **inteligencia artificial (IA)** es una rama de la ciencia de la computación que se centra en desarrollar sistemas que pueden realizar tareas que, de otro modo, requerirían inteligencia humana. **Machine learning (aprendizaje automático)** es un subconjunto de la IA que se enfoca en el desarrollo de algoritmos que permiten a las computadoras aprender patrones y realizar tareas sin ser programadas explícitamente. **Deep learning (aprendizaje profundo)** es una técnica específica dentro del aprendizaje automático que utiliza redes neuronales profundas con múltiples capas para modelar y procesar datos de manera más compleja, logrando representaciones jerárquicas de características.

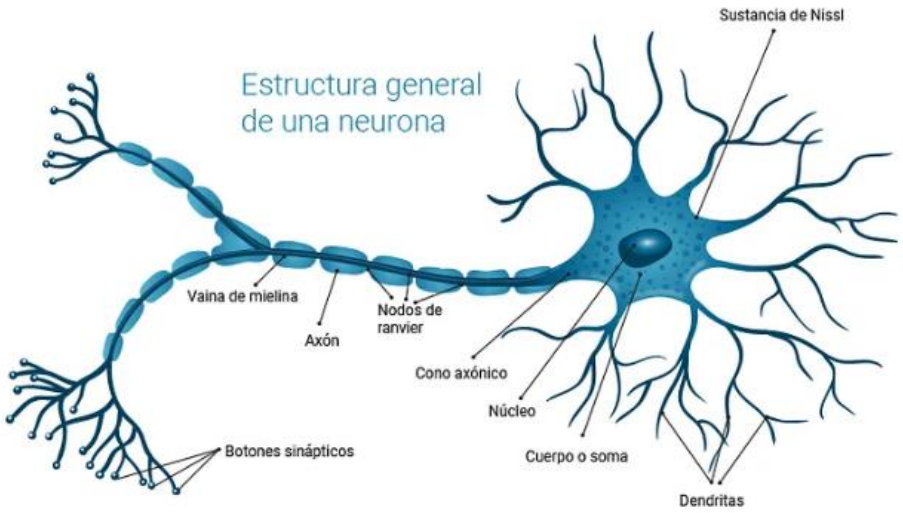
Las **similitudes** entre estas disciplinas radican en que todas están vinculadas a la ciencia de la computación y comparten fundamentos teóricos, como algoritmos y estructuras de datos. La inteligencia artificial, el aprendizaje automático y el aprendizaje profundo están interrelacionados, como hemos podido observar en el párrafo anterior. Sin embargo, las **diferencias** radican en su enfoque y complejidad. Mientras que la inteligencia artificial es un campo más amplio, el aprendizaje automático se centra en el desarrollo de algoritmos que pueden aprender de datos, y el aprendizaje profundo utiliza redes neuronales profundas para realizar tareas más complejas y abstractas, como el reconocimiento de patrones en datos complejos como imágenes, voz o texto.

5.2.1. Definición y descripción de modelos CNN

Las **Redes Neuronales Convolucionales (CNN)** son un tipo de arquitectura de redes neuronales diseñadas específicamente para procesar datos bidimensionales, como imágenes.

En términos de semejanza con el cerebro humano, las CNN se inspiran en la organización cortical visual, donde las neuronas especializadas en regiones locales responden a características específicas.

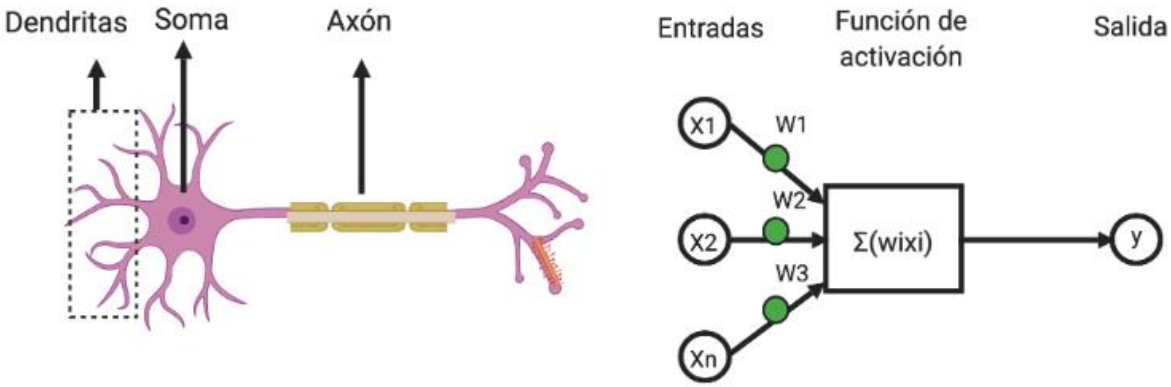
Figura 17. Estructura general de una neurona



Fuente: https://medium.com/soldai/inspiraci%C3%B3n-biol%C3%B3gica-de-las-redes-neuronales-artificiales-9af7d7b906a

Al igual que el cerebro, las CNN son capaces de aprender representaciones jerárquicas de características, imitando la manera en que el sistema visual humano procesa información. Sin embargo, es importante destacar que las CNN son simplificaciones y no replican completamente la complejidad y plasticidad del cerebro humano.

Figura 18. Neurona vs Perceptrón



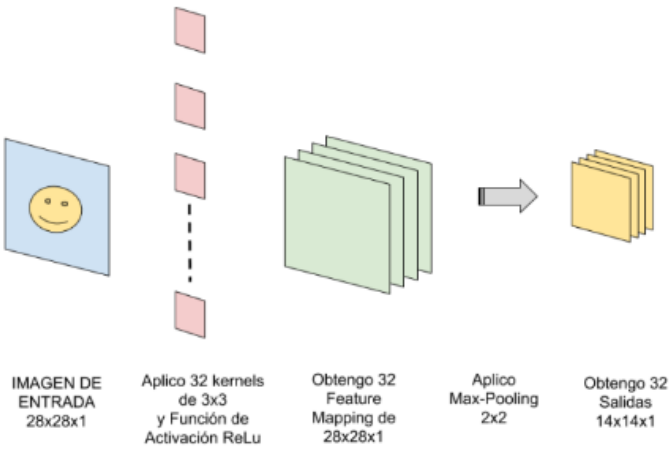
Fuente:
https://futurelab.mx/redes%20neuronales/inteligencia%20artificial/2019/06/25/intro-a-redes-neuronales-pt-1/

Las CNN están compuestas por varias capas, cada una desempeñando un papel específico en la extracción y procesamiento de característica. Estas capas las clasificamos en:

- **Capa de Convolución**: Esta capa es fundamental y utiliza filtros (o kernels) para escanear la entrada y realizar operaciones de convolución. Cada filtro detecta patrones locales, como bordes o texturas, generando mapas de características convolucionales.
- **Capa de Activación (ReLU)**: Después de la convolución, se aplica una función de activación, comúnmente ReLU (Rectified Linear Unit), que introduce no linealidades al transformar los valores negativos a cero, es decir, la función ReLU transforma los valores de entrada de la siguiente forma:
    - **Si el valor de entrada es positivo, lo deja sin cambios**
    - **Si el valor de entrada es negativo, lo transforma a cero**

  Al transformar los valores negativos a cero, la función ReLU permite que solo las activaciones positivas se propaguen a través de la red, activando las neuronas asociadas. Esta no linealidad es importante para que la red pueda aprender patrones y relaciones más complejas en los datos, ya que introduce la capacidad de "activarse" solo cuando ciertos criterios se cumplen, mejorando así la capacidad de la red para representar y aprender características más sofisticadas.
- **Capa de Agrupación (Pooling)**: La capa de agrupación reduce la dimensionalidad espacial de los mapas de características, disminuyendo el número de parámetros y, por ende, la carga computacional. La operación más común es el max-pooling, que toma el valor máximo de un conjunto de valores vecinos.
- **Capa Totalmente Conectada**: Al final de la red, se suelen incluir capas totalmente conectadas que reciben las características extraídas y las utilizan para realizar la tarea específica, como la clasificación. Cada neurona en estas capas está conectada a todas las neuronas de la capa anterior, permitiendo la combinación de características aprendidas.

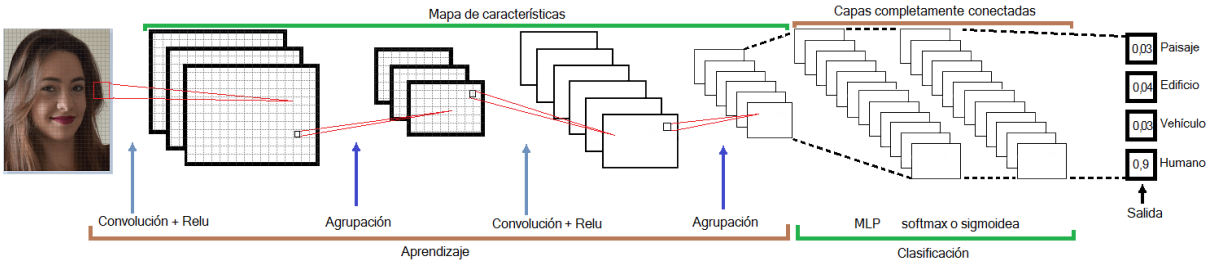
Figura 19. Primera convolución de una red neuronal convolucional (podemos tener una o varias convoluciones)



Fuente: https://www.juanbarrios.com/redes-neurales-convolucionales/

Una vez entendidas las capas que conforman una CNN, tenemos que realizar una pequeña matización en la capa **totalmente conectada** ya que esta capa no forma parte de lo que es el aprendizaje de la red si no que es parte de la clasificación del modelo. Esta capa juega un papel clave en la interpretación y clasificación de esas características para realizar la tarea final del modelo, ya sea clasificación de imágenes, detección de objetos u otra tarea específica. En el contexto de la clasificación, estas capas suelen tener una función de activación final, como la función **softmax**, que asigna probabilidades a las diferentes clases. Este matiz nos puede quedar más claro con la Figura 19:

Figura 20. Esquema del conjunto de las distintas capas CNN



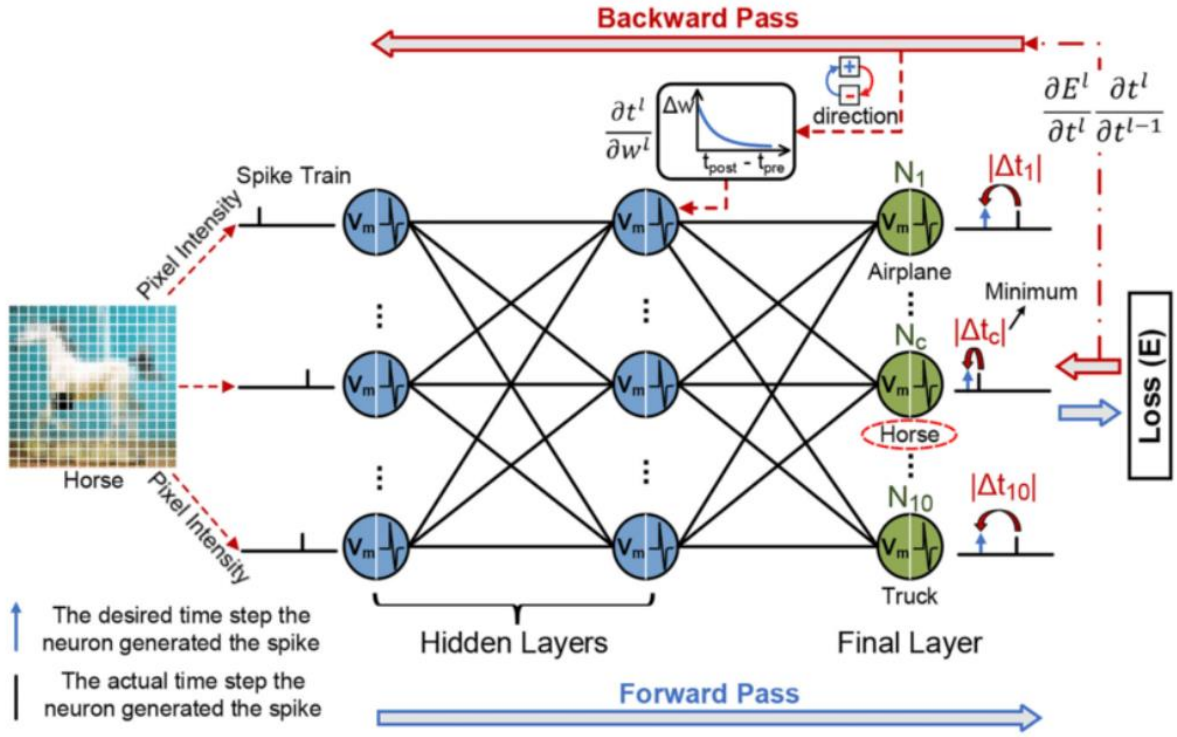
Fuente: https://numerentur.org/convolucionales/

En el proceso de entrenamiento, el **objetivo** es ajustar los pesos de las conexiones entre las neuronas de la red de manera que las predicciones del modelo se acerquen lo más posible a

las etiquetas reales de los datos de entrenamiento. El modelo matemático de **Backpropagation** utiliza el método del descenso del gradiente para lograr este ajuste.

Primero, durante la fase de propagación hacia adelante, se realizan predicciones y se calcula la diferencia entre estas predicciones y las etiquetas reales, conocida como la función de pérdida.

Figura 21. Esquema del modelo matemático Backpropagation



Fuente: https://www.researchgate.net/figure/The-illustration-of-the-forward-process-and-error-backpropagation-in-the-SSTDP-method_fig4_355905487

En la fase de **retropropagación (backpropagation)**, se calcula el gradiente de la función de pérdida con respecto a los pesos de la red. Este gradiente indica la dirección y la magnitud en la que los pesos deben ajustarse para reducir la pérdida. Luego, se ajustan los pesos en la dirección opuesta al gradiente, lo que significa que se actualizan para minimizar la pérdida. Este proceso iterativo se repite a lo largo de múltiples iteraciones (o épocas) de entrenamiento hasta que el modelo converge, es decir, hasta que los pesos alcanzan una configuración que minimiza la pérdida y maximiza el rendimiento del modelo en la tarea específica que se está abordando, como la clasificación de imágenes en el caso de las CNN que vamos a utilizar en este proyecto.

Tabla 3.

Comparativa entre una red neuronal "tradicional" y una red neuronal convolucional

Características	Red “tradicional” multicapa	Red Neuronal Convolucional CNN
Datos de entrada en la Capa Inicial	Las características que analizamos, ej: ancho, alto, grosor, etc.	Píxeles de una imagen. Si es ca color, serán 3 capas para rojo, verde y azul.
Capas ocultas	Elegimos una cantidad de neuronas para las capas ocultas.	Tenemos dos tipos: Convolución y Subsampling.
Capa de Salida	La cantidad de neuronas que queremos clasificar, ej: para “maligno” ó “benigno” serán 2 neuronas.	Debemos “aplanar” la última convolución con una o más capas de neuronas ocultas y hacer una salida mediante SoftMax a la capa de salida que clasifica “maligno” o “benigno” serán 2 neuronas.
Aprendizaje	Supervisado	Supervisado
Interconexiones	Entre capas, todas las neuronas de una capa con la siguiente	Son muchas menos conexiones necesarias, pues los pesos que ajustamos serán en los de los filtros/kernels que usamos.
Significado de la cantidad de capas ocultas	Realmente es algo desconocido y no representa algo en sí mismo.	Las capas ocultas son mapas de detección de características de la imagen y tienen jerarquía: primeras capas detectan líneas, luego curvas y formas cada vez más elaboradas.

Características	Red “tradicional” multicapa	Red Neuronal Convolucional CNN
Backpropagation	Se utiliza para ajustar los pesos de todas las interconexiones de las capas.	Se utiliza para ajustar los pesos de los kernels.

Nota. Fuente: https://www.juanbarrios.com/redes-neurales-convolucionales/

5.2.2. Implementación de nuestro modelo CNN

Una vez visto el contexto y cómo funcionan las CNN, es el momento de implementar la nuestra propia. A continuación, mostraremos el código y lo explicaremos después:

```
# Para ello primero cargamos en dataFrames los csv generados con la carga de los datos de las imágenes y sus etiquetas

# Configurar para mostrar todas las columnas y filas
# Para ver la matriz total declaramos estas opciones
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
np.set_printoptions(threshold=np.inf)

# Cargar los CSV de entrenamiento y prueba
    train_df = pd.read_csv('resultadosXytrain.csv', delimiter=';')
    test_df = pd.read_csv('resultadosXytest.csv', delimiter=';')

    # Convertir la columna 'Image' de lista a matriz numpy
    train_df['Image'] = train_df['Image'].apply(eval)
    test_df['Image'] = test_df['Image'].apply(eval)

    X_train = np.array(train_df['Image'].tolist())
    X_test = np.array(test_df['Image'].tolist())

    # Convertir las etiquetas a números usando LabelEncoder
    label_encoder = LabelEncoder()
    train_df['Label'] = label_encoder.fit_transform(train_df['Label'])
    test_df['Label'] = label_encoder.transform(test_df['Label'])

    y_train = train_df['Label']
    y_test = test_df['Label']

```

Hacemos uso de las funciones de **Pandas** y **NumPy** para configurar la visualización de datos en el entorno de trabajo. Como hemos visto en otras partes del código, ***pd.set_option*** se utiliza para mostrar todas las filas y columnas en la consola, y ***np.set_printoptions*** se configura para mostrar matrices NumPy sin restricciones, lo que nos es muy útil para ver matrices grandes.

Cargamos los datos de entrenamiento y prueba desde los dos archivos CSV llamados 'resultadosXytrain.csv' y 'resultadosXytest.csv' generados la función ***save_to_csv*** descrita con anterioridad. Especificamos en la carga que estos archivos contienen información sobre las imágenes y sus etiquetas utilizando el delimitador ';' (*delimiter=';'*).

Una vez cargados los datos realizamos dos conversiones, una para las imágenes del csv y otra para las etiquetas de este:

- Las columnas '*Image*' en los conjuntos de entrenamiento y prueba contienen listas de datos que representan las imágenes. La función ***apply(eval)*** se utiliza para evaluar estas listas y convertirlas en matrices **NumPy**. Como resultado, las columnas '*Image*' ahora contienen matrices NumPy que representan las imágenes.
- Las etiquetas en la columna '*Label*' se convierten de texto a números utilizando ***LabelEncoder*** de **scikit-learn**. Esta transformación es necesaria para que el modelo pueda interpretar las etiquetas en formato numérico.

Una vez que tenemos tanto la carga de los csv como las conversiones necesarias hechas, creamos las matrices de entrada *X_train* y *X_test* a partir de las columnas '*Image*', que ahora contienen matrices NumPy. Las etiquetas también se almacenan en *y_train* y *y_test* después de la transformación numérica.

En este punto, ya tenemos todos los datos preparados para crear nuestro modelo, compilarlo, entrenarlo y evaluarlo. Vamos a ver paso a paso que hace el resto del código:

Construcción del modelo:

```
1. # Construir el modelo
2. model = keras.Sequential([
3.     layers.Conv2D(32, (3, 3), activation='relu', input_shape=(1
   28, 128, 1)),
4.     layers.MaxPooling2D((2, 2)),
5.     layers.Flatten(),
6.     layers.Dense(64, activation='relu'),
7.     layers.Dense(4, activation='softmax')  # 4 clases: maligno,
   benigno, benigno sin callback, maligno sin callback
```

```
8. ])
```

Creamos un modelo secuencial de Keras (Sequential), que permite apilar capas de manera secuencial. Las capas las distribuiremos de la siguiente forma:

- La primera capa es una **capa convolucional** (*Conv2D*) con 32 filtros de 3x3, función de activación **ReLU**, y la forma de entrada de (128, 128, 1) que representa imágenes de 128x128 píxeles en escala de grises.
- Luego, aplicamos una capa de **max pooling** (MaxPooling2D) de 2x2 para reducir la dimensionalidad espacial.
- A continuación, utilizamos una **capa de aplanado** (Flatten) para convertir las salidas de la capa anterior en un vector unidimensional.
- Añadimos una **capa densa** (Dense) con 64 unidades y función de activación **ReLU**. Esta capa es esencial para que el modelo pueda aprender y comprender patrones más complejos y globales en los datos, lo que mejora la capacidad de clasificación y generalización del modelo.
- Finalmente, agregamos una **capa de salida** densa con 4 unidades (para las 4 clases) y función de activación **softmax** para la clasificación multiclase.

Compilación del modelo:

```
1. # Compilar el modelo
2. model.compile(optimizer='adam', loss='sparse_categorical_crosse
   ntropy', metrics=['accuracy'])
```

Compilamos el modelo utilizando el optimizador '**adam**', la función de pérdida '**sparse_categorical_crossentropy**' (adecuada para clasificación multiclase con etiquetas numéricas), y se monitoriza la métrica de precisión ('***accuracy***'). Seleccionamos un optimizador *Adam* debido a su eficiencia y capacidad para adaptar las tasas de aprendizaje, mientras que la función de pérdida *sparse_categorical_crossentropy* la elegimos para problemas de clasificación multiclase con etiquetas enteras, evitando la necesidad de codificación *one-hot* (one-of-K o dummy encoding, es una técnica utilizada para representar categorías o etiquetas de manera binaria).

Entrenamiento del modelo:

```
# Entrenar el modelo
model.fit(X_train, y_train, epochs=10, validation_data=(X_test,
    y_test))
```

Entrenamos el modelo utilizando el conjunto de datos de entrenamiento (*X_train* e *y_train*) durante 10 épocas. Además, utilizamos el conjunto de prueba (*X_test* e *y_test*) como conjunto de validación durante el entrenamiento.

Evaluación del modelo:

```
# Evaluar el modelo en el conjunto de prueba
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f'\nPrecisión en el conjunto de prueba: {test_acc}')
print(f'\nPérdida en el conjunto de prueba: {test_loss}')
```

Evaluamos el modelo con el conjunto de prueba, obteniendo la pérdida y la precisión.
El resultado obtenido de nuestro primer modelo de Red Neuronal Convolucional (CNN) es el siguiente:

Figura 22. Entrenamiento del modelo y resultados

```
Epoch 1/10
1/1 [==============================] - 1s 1s/step - loss: 1.4202 - accuracy: 0.0000e+00 - val_loss: 2.8542 - val_accuracy: 0.1918
Epoch 2/10
1/1 [==============================] - 1s 580ms/step - loss: 0.8331 - accuracy: 0.6667 - val_loss: 3.8444 - val_accuracy: 0.5028
Epoch 3/10
1/1 [==============================] - 1s 573ms/step - loss: 0.0108 - accuracy: 1.0000 - val_loss: 5.5240 - val_accuracy: 0.5043
Epoch 4/10
1/1 [==============================] - 1s 576ms/step - loss: 9.5809e-04 - accuracy: 1.0000 - val_loss: 6.8105 - val_accuracy: 0.5043
Epoch 5/10
1/1 [==============================] - 1s 573ms/step - loss: 1.6348e-04 - accuracy: 1.0000 - val_loss: 7.8621 - val_accuracy: 0.5043
Epoch 6/10
1/1 [==============================] - 1s 567ms/step - loss: 3.6754e-05 - accuracy: 1.0000 - val_loss: 8.7642 - val_accuracy: 0.5043
Epoch 7/10
1/1 [==============================] - 1s 575ms/step - loss: 1.0292e-05 - accuracy: 1.0000 - val_loss: 9.5507 - val_accuracy: 0.5043
Epoch 8/10
1/1 [==============================] - 1s 569ms/step - loss: 3.4968e-06 - accuracy: 1.0000 - val_loss: 10.2433 - val_accuracy: 0.5043
Epoch 9/10
1/1 [==============================] - 1s 584ms/step - loss: 1.3510e-06 - accuracy: 1.0000 - val_loss: 10.8573 - val_accuracy: 0.5043
Epoch 10/10
1/1 [==============================] - 1s 579ms/step - loss: 5.9605e-07 - accuracy: 1.0000 - val_loss: 11.4047 - val_accuracy: 0.5043
22/22 [==============================] - 1s 22ms/step - loss: 11.4047 - accuracy: 0.5043

Precisión en el conjunto de prueba: 0.5042613744735718

Pérdida en el conjunto de prueba: 11.404702186584473
```

Estos resultados indican que el modelo tiene **una precisión del 50.4%**, lo que significa que acierta aproximadamente la mitad de las veces en el conjunto de prueba. Sin embargo, **la**

pérdida es relativamente alta (11.4), y esto nos sugiere que hay margen de mejora para reducir la discrepancia entre las predicciones y las etiquetas reales.

6. Estrategias de mejora en nuestro modelo CNN

6.1. Configuración de Tensorflow para que utilice la GPU del ordenador

Una vez que tenemos nuestro modelo CNN desarrollado, es momento de estudiar los distintos métodos y técnicas que podemos utilizar para mejorar la precisión y reducir la pérdida de nuestra red neuronal.

Un paso muy importante para poder comenzar a desarrollar estás técnicas, es poder utilizar la capacidad de cómputo de nuestra tarjeta gráfica (GPU), ya que esto nos va a permitir realizar cálculos y operaciones de una forma más rápida y eficiente. Para ello configuramos nuestra librería Tensorflow de tal forma que pueda detectar nuestra GPU y así utilizarla. Este paso es complejo y se detalla en el Anexo C.

Una vez que tenemos todas las librerías en la versión correcta, utilizamos el siguiente código para utilizar la GPU con Tensorflow y habilita el crecimiento dinámico de la memoria en la misma evitando su agotamiento:

```
# Configurar para utilizar GPU si está disponible
if tf.config.list_physical_devices('GPU'):
    physical_devices = tf.config.list_physical_devices('GPU')
    tf.config.experimental.set_memory_growth(physical_devices[0], enable=True)
    print("GPU encontrada y configurada para el crecimiento dinámico de la memoria.")
else:
    print("No se encontraron GPUs disponibles.")
```

Comprobamos si hay dispositivos físicos de tipo GPU disponibles utilizando la función **tf.config.list_physical_devices('GPU')**. Si esta lista no está vacía (es decir, si hay al menos un dispositivo GPU disponible, obtenemos una lista de todos los dispositivos físicos de tipo GPU disponibles y lo guardamos en **physical_devices**.

Configuramos el crecimiento dinámico de la memoria para el primer dispositivo GPU encontrado con la función **tf.config.experimental.set_memory_growth(physical_devices[0], enable=True)**. Esto significa que TensorFlow asignará memoria a medida que sea necesaria en lugar de asignar toda la memoria de una vez, lo que puede ayudar a evitar problemas de falta de memoria.

Finalmente imprimimos un mensaje indicando que se encontró una GPU y se configuró para el crecimiento dinámico de la memoria, o no se encontraron GPUs disponibles.

6.2. Mejora de la Arquitectura CNN a través de Capas Convolutivas y Técnicas de Regularización

Dentro de las mejoras que vamos a realizar en nuestro modelo CNN ya creado, vamos a explicar por tipo de mejora, agrupándolas en las siguientes subcategorías: Expansión de la profundidad y complejidad de la red CNN, ajuste del optimizador y la tasa de aprendizaje, optimización de los datos de entrada con aumento de datos y entrenamiento mejorado con datos aumentados.

6.2.1. Expansión de la profundidad y complejidad de la red CNN

Procedemos a mejorar en esta parte del código, la arquitectura CNN añadiendo capas convolucionales, pooling y dropout. En el siguiente fragmento, se agregan múltiples capas para extraer características de las imágenes, regularizando así el modelo y reduciendo el sobreajuste.

```
# Mejoramos los resultados Agregando capas convolucionales y pooling y dropout
model = keras.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 1)),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(256, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(256, (3, 3), activation='relu'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),

    layers.Dropout(0.5),  # Ajusta el valor de dropout
    layers.Dense(128, activation='relu'),
    layers.BatchNormalization(),

    layers.Dropout(0.5),  # Ajusta el valor de dropout
```

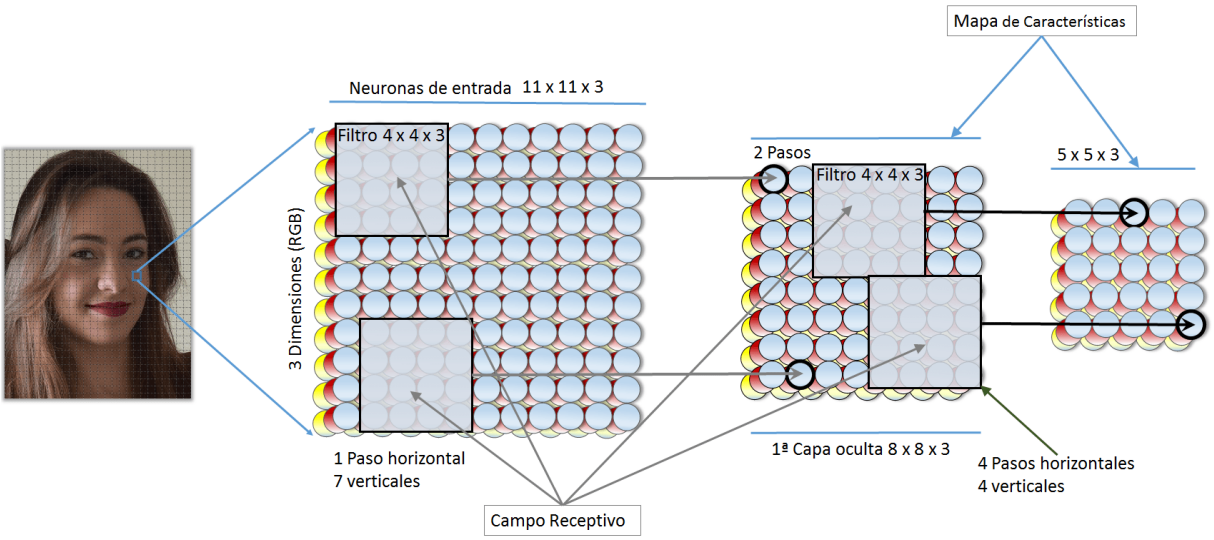
```
26.      layers.Dense(4, activation='softmax')
27.      ])
28.
29.
```

Añadimos capas de convolución y pooling adicionales:

Aumentamos la profundidad y la complejidad del modelo al añadir más capas de **convolución** y **pooling**. Esto permite que la red neuronal aprenda características de mayor nivel de abstracción en los datos de entrada. Las capas de convolución adicionales permiten detectar patrones más complejos en las imágenes, mientras que las capas de pooling reducen la dimensionalidad de las representaciones, manteniendo las características más relevantes.

Esto puede mejorar la precisión del modelo al capturar y representar de manera más efectiva las relaciones espaciales entre los píxeles de la imagen, lo que resulta en una mejor capacidad de generalización a nuevas muestras.

Figura 23. Matriz que se genera sobre la imagen de entrada realizando operaciones de multiplicación



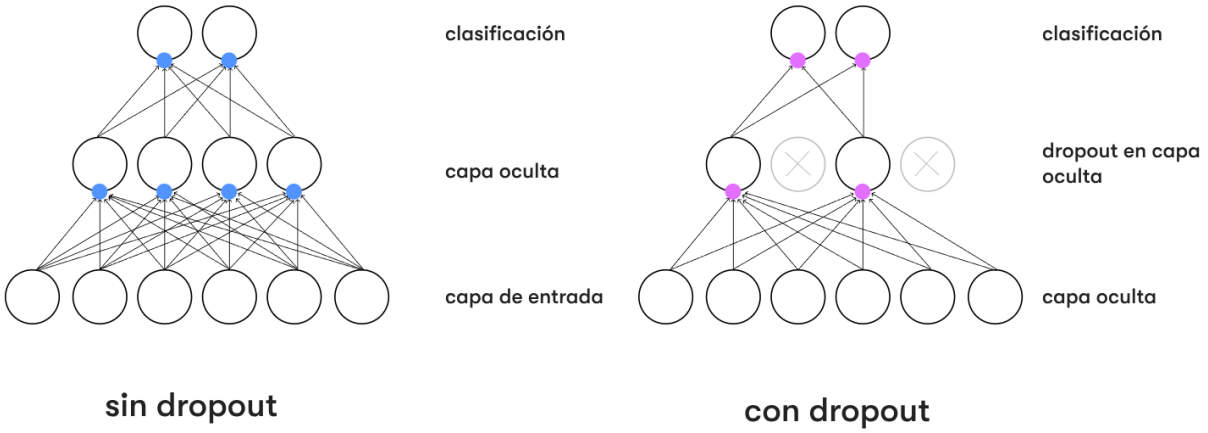
Fuente: https://numerentur.org/wp-content/uploads/2018/09/capa-convolucional-con-ester.png

Incluimos capas de dropout:

Con la adición de capas de dropout al modelo, ayudamos a regularizar el mismo, lo que significa que con estas capas ayudamos a prevenir el sobreajuste al reducir la coadaptación

entre las neuronas. Durante el entrenamiento, las capas de dropout desactivan aleatoriamente un porcentaje de las neuronas, lo que evita que el modelo dependa demasiado de un subconjunto particular de características.

Figura 24. Añadiendo dropout a las capas



Fuente: https://anywhere.epam.com/es/blog/preguntas-entrevista-aprendizaje-automatico

Esto puede mejorar la precisión del modelo al reducir el sobreajuste, lo que permite que el modelo generalice mejor a datos invisibles, resultando en una mayor precisión en el conjunto de datos de prueba.

Añadimos capas densas y normalización por lotes:

Permitimos que el modelo aprenda relaciones no lineales más complejas entre las características extraídas de las imágenes, cuando incluimos capas densas adicionales después de las capas de convolución y aplanado.

La normalización por lotes estandariza las activaciones de cada capa, lo que acelera el entrenamiento y mejora la estabilidad del modelo.

Estos cambios pueden mejorar la precisión del modelo al permitir una mejor representación de las características de las imágenes y una optimización más efectiva durante el entrenamiento.

6.2.2. Ajuste del optimizador y la tasa de aprendizaje

En el siguiente fragmento del código compilamos el modelo CNN configurando el optimizador para ser Adam con una tasa de aprendizaje de 0.0001. La función de pérdida se establece como '**sparse_categorical_crossentropy**' y se mide la precisión durante el entrenamiento:

```
# Ajusta la tasa de aprendizaje según sea necesario
model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.0001), loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

La tasa de aprendizaje es un parámetro crítico en el entrenamiento de nuestra red neuronal ya que determina el tamaño de los pasos que se toman para actualizar los pesos durante la optimización. Una tasa de aprendizaje demasiado alta puede causar oscilaciones o divergencia, mientras que una tasa de aprendizaje demasiado baja puede ralentizar el proceso de entrenamiento o quedar atrapada en mínimos locales.

En el código actual, personalizamos la tasa de aprendizaje del optimizador Adam, estableciéndola en 0.0001. Esta personalización permite ajustar finamente la velocidad de convergencia del modelo durante el entrenamiento.

6.2.3. Optimización de los datos de entrada con aumento de datos

En el código que sigue, configuramos el generador de aumento de datos utilizando la clase **ImageDataGenerator** de **Keras**. Especificamos diferentes transformaciones de imagen como rotación, desplazamiento horizontal y vertical, cambio de inclinación, zoom y volteo horizontal para aumentar la diversidad del conjunto de datos de entrenamiento.

```
# Configurar el generador de aumento de datos
datagen = ImageDataGenerator(
    rotation_range=30,
    width_shift_range=0.3,
    height_shift_range=0.3,
    shear_range=0.3,
    zoom_range=0.3,
    horizontal_flip=True,
    fill_mode='nearest'
     )

     # Aplicar el aumento de datos a las imágenes de entrenamiento
     datagen.fit(X_train)
```

Con **datagen.fit** aplicamos las transformaciones de aumento de datos y preprocesamiento al conjunto de datos de entrenamiento **(X_train)**. Esto significa que estas transformaciones se calculan estadísticamente utilizando solo el conjunto de datos de entrenamiento.

Esto es importante para evitar fugas de información entre el conjunto de entrenamiento y el conjunto de validación o prueba, lo que garantiza una evaluación justa y realista del rendimiento del modelo.

6.2.4. Entrenamiento mejorado con datos aumentados

En este fragmento, entrenamos el modelo utilizando el generador de datos aumentados. Los datos de entrenamiento se generan dinámicamente durante el entrenamiento utilizando el generador **datagen**. Esto nos permite que el modelo se entrene con una mayor variabilidad de datos, lo que puede mejorar su capacidad para generalizar a nuevas muestras.

```
14.     # Entrenar el modelo con el generador de datos aumentados
15.     model.fit(datagen.flow(X_train, y_train, batch_size=batch
   _size), epochs=100, validation_data=(X_test, y_test))
```

También hemos aumentado el número de épocas de 10 a 100 con lo que conseguimos:

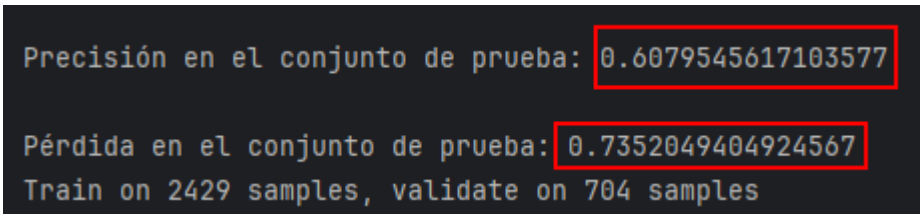
- **Mejora en la convergencia**: Muchas veces un número insuficiente de épocas puede no ser suficiente para que el modelo converja a una solución óptima. Cuando aumentamos el número de épocas, damos más tiempo al modelo para explorar el espacio de parámetros y encontrar una solución más precisa.
- **Reducción del sesgo de entrenamiento**: Al entrenar durante más épocas, el modelo tiene la oportunidad de ver más ejemplos de entrenamiento, lo que nos puede ayudar a reducir el sesgo de entrenamiento. Esto significa que el modelo puede aprender patrones más representativos de los datos y generalizar mejor a ejemplos no vistos.
- **Mejora en la generalización**: Con un mayor número de épocas podemos ayudar al modelo a aprender características más complejas y sutiles en los datos. Esto puede resultar en una mejora en su capacidad para realizar predicciones precisas en datos que no ha visto durante el entrenamiento.
- **Riesgo de sobreajuste**: Si bien es cierto que hasta ahora todo eran ventajas, es importante tener en cuenta que cuando aumentamos el número de épocas también aumentamos el riesgo de sobreajuste, especialmente si nuestro modelo es lo suficientemente complejo como para memorizar los datos de entrenamiento en lugar de aprender patrones generales. Por lo tanto, es importante monitoricemos el rendimiento del modelo en un conjunto de datos de validación para detectar signos

de sobreajuste y que podamos tomar medidas correctivas, como la introducción de regularización o la disminución del tamaño del modelo.

Aplicamos nuevamente la función **model.evaluate** y evaluamos la precisión y la pérdida obtenida una vez realizados estos cambios:

```
1. # Evaluar el modelo en el conjunto de prueba
2. test_loss, test_acc = model.evaluate(X_test, y_test)
3. print(f'\nPrecisión en el conjunto de prueba: {test_acc}')
4. print(f'\nPérdida en el conjunto de prueba: {test_loss}')
```

Figura 25. Entrenamiento del nuevo modelo y resultados



Nuestro modelo inicial presentaba una **precisión** del **50.43%** y una **pérdida** de **11.40** en el conjunto de datos de prueba, antes de que implementásemos las mejoras en la arquitectura de la red y en el proceso de entrenamiento. Estos valores nos indicaban que el modelo tenía dificultades para realizar predicciones precisas y eficientes sobre las cuatro clases de clasificación de nuestro modelo.

Una vez que hemos realizado las mejoras descritas en la arquitectura de la CNN y en el proceso de entrenamiento, hemos logrado mejoras significativas en el rendimiento del modelo. La **precisión** hemos conseguido aumentarla hasta un **60.79%**, mientras que la **pérdida** disminuyó a **0.7352**. Estos resultados nos indican una mejora sustancial en la capacidad del modelo para realizar predicciones precisas y generalizar a nuevas muestras de datos.

Las mejoras que hemos realizado en la arquitectura de la red aumentaron la profundidad y la complejidad del modelo, permitiendo una mejor extracción de características de las imágenes de entrada y una reducción del sobreajuste.

Además, implementamos el aumento de datos mediante la aplicación de transformaciones aleatorias a las imágenes durante el entrenamiento, lo que permitió al modelo aprender a generalizar mejor al exponerlo a una mayor variabilidad de datos de entrenamiento.

6.3. Transfer Learning y otras arquitecturas de redes neuronales

Buscamos mejorar la eficacia de nuestro modelo CNN mediante el empleo de transfer learning con otras arquitecturas de redes neuronales, en nuestro estudio. El **transfer learning** es una técnica poderosa que **permite aprovechar el conocimiento aprendido por modelos previamente entrenados en conjuntos de datos similares**. En este sentido, exploramos la posibilidad de utilizar arquitecturas pre-entrenadas, como mobilenetV2, mobilenetV3, VGG, ResNet o Inception, como punto de partida para nuestro modelo CNN. Esta estrategia nos brinda la ventaja de comenzar con pesos optimizados y características aprendidas de tareas relacionadas, lo que puede acelerar el proceso de entrenamiento y mejorar la generalización del modelo. Además, nos permite aprovechar la capacidad de estas arquitecturas para extraer características relevantes de las imágenes de entrada, adaptándolas a nuestro problema específico. El objetivo de este apartado es evaluar el rendimiento de estas diferentes arquitecturas pre-entrenadas en nuestra tarea particular, así como explorar la posibilidad de realizar ajustes finos (fine-tuning) para adaptar mejor el modelo a nuestro conjunto de datos específico. Para lograr esto, llevaremos a cabo experimentos rigurosos utilizando nuestro conjunto de datos y métricas de evaluación estándar para comparar el desempeño de cada enfoque. Analizaremos el impacto de diferentes estrategias de transfer learning en términos de rendimiento, tiempo de entrenamiento y capacidad de generalización del modelo final en el último capítulo antes de pasar a las conclusiones finales y líneas de trabajo futuro. En última instancia, esperamos que este enfoque nos permita desarrollar un modelo CNN altamente eficaz y generalizable para nuestra tarea específica.

Para realizar estas nuevas codificaciones, esta vez vamos a utilizar Google Colab. **Google Colab** es una plataforma en la nube que nos permite escribir y ejecutar código Python en notebooks interactivos. Nos ofrece acceso gratuito a recursos informáticos como GPUs y TPUs para proyectos de aprendizaje automático y ciencia de datos, facilitando la colaboración en tiempo real y la compartición de notebooks. Aprovecharemos su integración con Google Drive para el almacenamiento y la organización de proyectos.

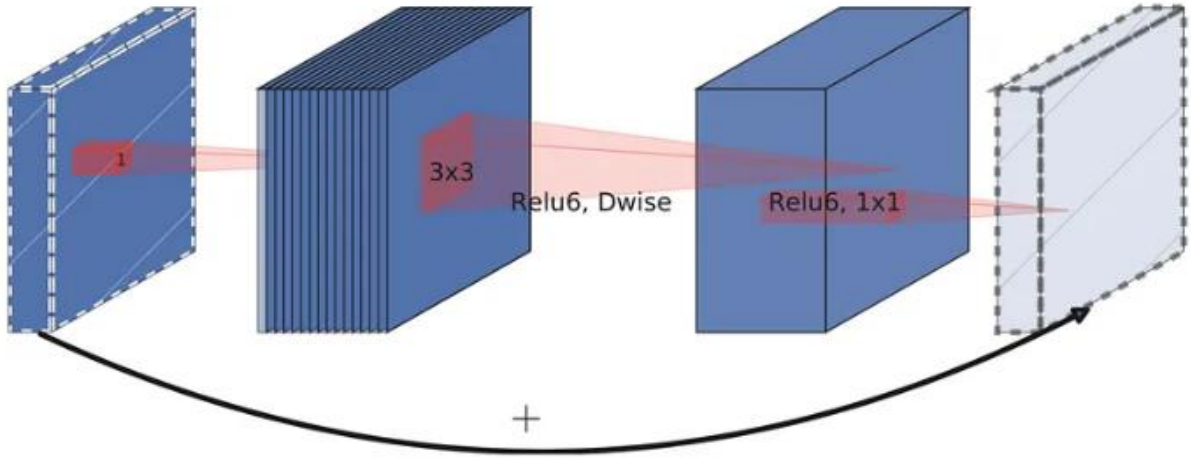
6.3.1. ¿Qué es MobileNetV2?

MobileNetV2 es una arquitectura de red neuronal convolucional (CNN) **diseñada específicamente para aplicaciones de visión por computadora** en dispositivos con recursos

computacionales limitados, como teléfonos móviles y dispositivos IoT (Internet de las cosas). Fue desarrollada por **Google** y es una evolución de la arquitectura original MobileNet, con mejoras significativas en términos de precisión y eficiencia.

La principal característica de MobileNetV2 es su estructura de bloques de construcción, conocidos como bloques "**Inverted Residuals**" (residuales invertidos), que permiten construir redes más profundas y eficientes en términos de uso de recursos computacionales. Estos bloques aprovechan la idea de los "residuales" introducidos en las redes residuales (ResNet), pero con una variación que reduce la cantidad de parámetros y operaciones necesarias, lo que los hace más adecuados para dispositivos con recursos limitados.

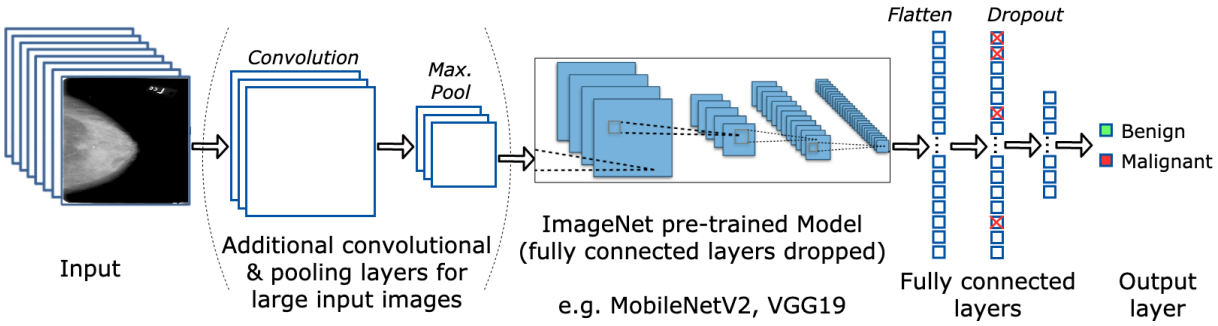
Figura 26. Visualización de los mapas de características intermedias en la capa residual invertida



Fuente: https://medium.com/@luis_gonzales/a-look-at-mobilenetv2-inverted-residuals-and-linear-bottlenecks-d49f85c12423

Asimismo, MobileNetV2 utiliza técnicas como la regularización por factor de escala y la expansión lineal de canales para mejorar la precisión y la capacidad de generalización de la red, al tiempo que mantiene una alta eficiencia computacional. Estas técnicas ayudan a reducir el riesgo de sobreajuste y a mejorar la capacidad del modelo para capturar y representar características relevantes en los datos de entrada.

Figura 27. Incorporación de MobileNetV2 en un Esquema de Red Neuronal Convolucional



Fuente: https://doi.org/10.1371/journal.pone.0280841

6.3.2. Codificamos MobileNetV2

Previamente a mostrar la codificación de esta arquitectura de red neuronal, vamos a codificar en nuestro notebook un fragmento con el que vamos a cargar los datos de los csv y estos van a servir para todos los modelos:

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder

# Montamos Google Drive en Google Colab
from google.colab import drive
drive.mount('/content/drive')

# Cargamos los datos desde los archivo CSV en Google Drive
    train_df = pd.read_csv('/content/drive/MyDrive/resultados Xytrain.csv', delimiter=';', nrows=700)
    test_df = pd.read_csv('/content/drive/MyDrive/resultadosXytest.csv', delimiter=';', nrows=700)

    # Aplicamos la función eval a la columna 'Image' para convertir las listas en arrays numpy
    train_df['Image'] = train_df['Image'].apply(eval)
    test_df['Image'] = test_df['Image'].apply(eval)

    # Convertimos la columna 'Label' a números usando LabelEncoder
    label_encoder = LabelEncoder()
    label_encoder.fit(train_df['Label'])

    train_df['Label'] = label_encoder.transform(train_df['Label'])
    test_df['Label'] = label_encoder.transform(test_df['Label'])


    # Convertim los datos a arrays numpy
    X_train = np.array(train_df['Image'].tolist())
```

```
    X_test = np.array(test_df['Image'].tolist())

    y_train = train_df['Label']
    y_test = test_df['Label']
```

Llevamos a cabo la carga y preprocesamiento de datos utilizando la biblioteca Pandas en el entorno de Google Colab. Primero, montamos Google Drive para acceder a los conjuntos de datos almacenados y cargamos los datos desde archivos CSV específicos ubicados en nuestro Google Drive personal, limitando la cantidad de filas a 700 para optimizar la eficiencia computacional. El problema de hacerlo con los ficheros csv completos es que caduca la sesión de Google Colab antes de que termine la ejecución de este fragmento. Finalmente, generamos conjuntos de entrenamiento y prueba tal y como se ha visto en capítulos anteriores de este trabajo.

```
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau
from sklearn.model_selection import train_test_split
import numpy as np


    # Cargar MobileNetV2 preentrenado en ImageNet sin la capa densa superior
    base_model = MobileNetV2(include_top=False, weights='imagenet', input_shape=(128, 128, 3))

    # Congelar todas las capas convolucionales de MobileNetV2
    for layer in base_model.layers:
        layer.trainable = False

    # Agregar capas adicionales para adaptar MobileNetV2 a tu conjunto de datos
    model = Sequential([
        base_model,
        GlobalAveragePooling2D(),
        Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(1, activation='sigmoid')
    ])

    # Compilar el modelo
    model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy'])

    # Reducir la tasa de aprendizaje cuando la métrica de validación se estanque
```

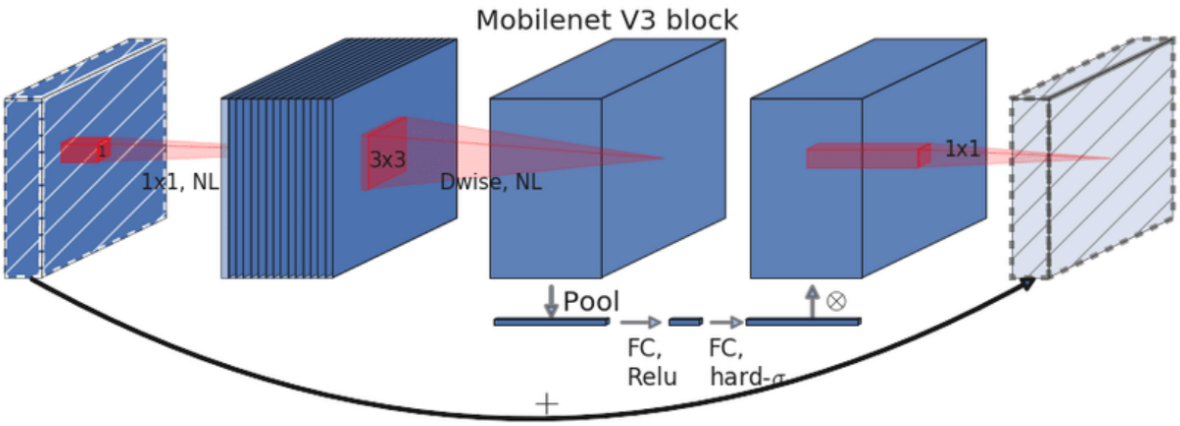
```
    reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=
  0.2, patience=5, min_lr=0.0001)

    # Entrenar el modelo
    history = model.fit(
        X_train, y_train,
        epochs=100,
        batch_size=32,
        validation_data=(X_test, y_test),
        callbacks=[reduce_lr]
    )

    # Evaluar el modelo en el conjunto de test
    test_loss, test_acc = model.evaluate(X_test, y_test)
    print(f'Precisión en el conjunto de prueba: {test_acc}')
    print(f'Pérdida en el conjunto de prueba: {test_loss}')
```

En este fragmento de código, primero, cargamos MobileNetV2 sin la capa densa superior, especificando la forma de entrada de nuestras imágenes. A continuación, **congelamos** todas las capas convolucionales de MobileNetV2 para evitar que se actualicen durante el entrenamiento y así mantener los conocimientos previamente aprendidos. Luego, **agregamos capas adicionales**, incluyendo una capa de agrupación global, capas densas con activación ReLU y dropout para regularización, y una capa densa de salida con activación sigmoide para clasificación binaria (por simplificar el modelo vamos a clasificar si tiene Cáncer o no). Además, incorporamos la función **ReduceLROnPlateau** como un callback para **reducir la tasa de aprendizaje** si la métrica de pérdida en el conjunto de validación no mejora durante un número específico de épocas. Finalmente, entrenamos el modelo utilizando los conjuntos de entrenamiento y validación, supervisando la métrica de pérdida en el conjunto de validación y utilizando el callback previamente definido. Luego evaluamos el modelo en el conjunto de prueba y mostramos la precisión y la pérdida obtenidas.

6.3.3. ¿Qué es MobileNetV3?

MobileNetV3 es una versión mejorada y optimizada de la arquitectura de redes neuronales MobileNet, diseñada para aplicaciones de visión por computadora en dispositivos con recursos limitados, como dispositivos móviles y sistemas embebidos. A diferencia de MobileNetV2, MobileNetV3 introduce varias mejoras significativas, incluyendo la incorporación de bloques residuales invertidos (**MBConv**), que permiten una mayor eficiencia en el uso de recursos computacionales y una mejora en la precisión de la red.

Figura 28. Estructura de bloques de MobileNetV3



Fuente: https://www.researchgate.net/figure/MobileNetV2-Layer-vs-MobileNetV3-Block_fig4_351564462

También, MobileNetV3 presenta una atención más cuidadosa en la selección de hiperparámetros y el diseño de la arquitectura, lo que resulta en modelos más ligeros y eficientes en términos de velocidad de inferencia y uso de memoria. Otra mejora notable es la introducción de mecanismos de búsqueda automática de arquitectura (NAS) para optimizar la estructura de la red, lo que conduce a un mejor rendimiento en una amplia gama de tareas de visión por computadora.

6.3.4. Codificamos MobileNetV3

Procedemos a codificar de la misma forma que con MobileNetV2:

```
1. from keras.utils import to_categorical
2. from tensorflow.keras.applications import MobileNetV3Small
3. from tensorflow.keras.models import Sequential
4. from tensorflow.keras.layers import GlobalAveragePooling2D, Den
   se, Dropout
5. from tensorflow.keras.callbacks import EarlyStopping, ReduceLRO
   nPlateau
6. from tensorflow.keras.optimizers import Adam
7. from tensorflow.keras import regularizers
8.
9. # Convertimos las etiquetas de entrenamiento a codificación en
   caliente
10.     y_train_encoded = to_categorical(y_train, num_classes=2)
11.
12.     # Convertimos las etiquetas de prueba a codificación en
   caliente
13.     y_test_encoded = to_categorical(y_test, num_classes=2)
14.
15.     # Cargar MobileNetV3 preentrenado en ImageNet sin la capa
   densa superior
```

```
    base_model = MobileNetV3Small(include_top=False, weights='imagenet', input_shape=(128, 128, 3))

    # Congelamos todas las capas convolucionales de MobileNetV3
    for layer in base_model.layers:
        layer.trainable = False

    # Agregamos capas adicionales para adaptar MobileNetV3 a tu conjunto de datos
    model = Sequential([
        base_model,
        GlobalAveragePooling2D(),
        Dense(256, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
        Dropout(0.5),
        Dense(128, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
        Dropout(0.5),
        Dense(2, activation='softmax')  # 2 neuronas para 2 salidas con activación softmax
    ])

    # Compilamos el modelo
    model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])

    # Reducimos la tasa de aprendizaje cuando la métrica de validación se estanque
    reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=0.0001)

    # Configuramos detención temprana (Early Stopping)
    early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

    # Entrenamos el modelo
    history = model.fit(
        X_train, y_train_encoded,
        epochs=100,
        batch_size=32,
        validation_data=(X_test, y_test_encoded),
        callbacks=[reduce_lr, early_stopping]
    )

    # Evaluamos el modelo en el conjunto de test
    test_loss, test_acc = model.evaluate(X_test, y_test_encoded)
    print(f'Precisión en el conjunto de prueba: {test_acc}')
    print(f'Pérdida en el conjunto de prueba: {test_loss}')
```

Primero, convertimos las etiquetas de entrenamiento y prueba a codificación en caliente (one-hot encoding) para facilitar su procesamiento en un problema de clasificación con dos clases. Posteriormente, cargamos MobileNetV3 preentrenado en ImageNet sin la capa densa

superior y congelamos todas las capas convolucionales para retener los conocimientos previos adquiridos durante el entrenamiento en ImageNet.

Enriquecemos el modelo con capas adicionales, que incluyen una capa de agrupación global seguida de dos capas densas con activación ReLU y regularización L2 para mitigar el sobreajuste. Además, implementamos capas de dropout para la regularización del modelo. La capa de salida emplea la activación softmax para clasificar las entradas en una de las dos clases posibles.

En este modelo agregamos la función **EarlyStopping** para detener el entrenamiento prematuramente si la métrica de pérdida en el conjunto de validación no mejora después de un número específico de épocas.

6.3.5. ¿Qué es ResNet?

ResNet, es una arquitectura de red neuronal profunda que introdujo el concepto de conexiones residuales. Estas conexiones permiten que las capas de la red aprendan diferencias entre las entradas y las salidas en lugar de aprender directamente las funciones de mapeo, lo que facilita el entrenamiento de redes más profundas. Esta innovación ayudó a abordar el problema de desvanecimiento del gradiente y permitió el desarrollo de redes neuronales mucho más profundas, lo que llevó a un rendimiento mejorado en una variedad de tareas de visión por computadora.

6.3.6. Codificamos ResNet

Vamos a utilizar la versión específica de '**ResNet50**' para la codificación del modelo ResNet. Esto quiere decir que nuestro modelo consta de 50 capas de profundidad ya que como se ha comentado antes, en Google Colab tenemos límite de computación en cuanto a que podemos perder la sesión antes de que termine la compilación. Existen otras versiones como **ResNet101** que es una versión más profunda de ResNet, con 101 capas.

```
1. mport pandas as pd
2. import numpy as np
3. from sklearn.preprocessing import LabelEncoder
4. from tensorflow.keras.applications import ResNet50
5. from tensorflow.keras.models import Sequential
6. from tensorflow.keras.layers import Dense, GlobalAveragePooling
   2D, Dropout
7. from tensorflow.keras.optimizers import Adam
```

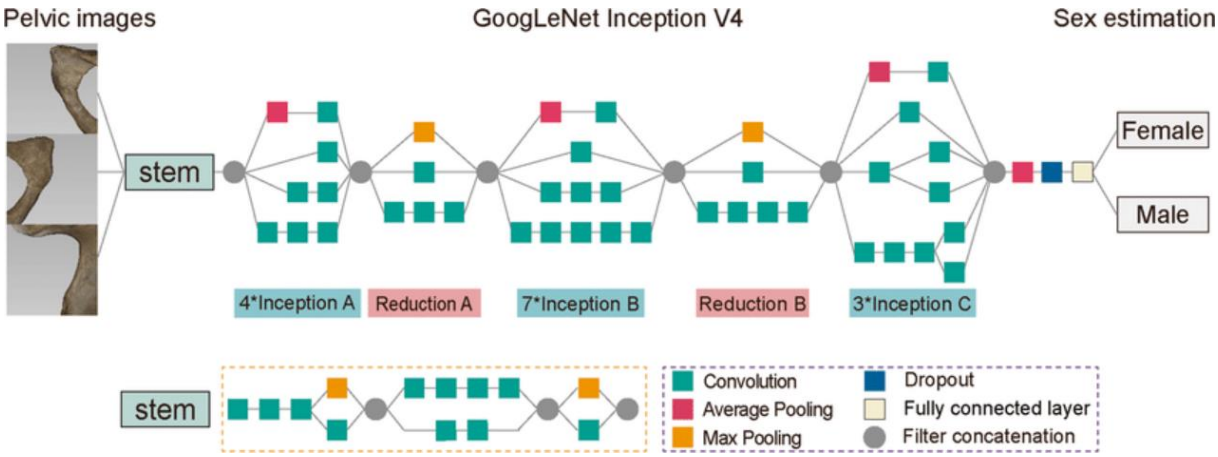
```
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau


    # Creamos el modelo de ResNet50 preentrenado
    base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(128, 128, 3))

    # Congelamos las capas convolucionales de ResNet50
    for layer in base_model.layers:
        layer.trainable = False

    # Añadimos capas adicionales al modelo
    model = Sequential([
        base_model,
        GlobalAveragePooling2D(),
        Dense(256, activation='relu'),
        Dropout(0.5),
        Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(1, activation='sigmoid')  # Solo necesitas una neurona para la clasificación binaria
    ])

    # Compilamos el modelo
    model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy'])

    # Reducimos la tasa de aprendizaje cuando la métrica de validación se estanca
    reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=0.0001)

    # Configuramos la detención temprana (Early Stopping)
    early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

    # Entrenamos el modelo
    history = model.fit(
        X_train, y_train,
        epochs=100,
        batch_size=32,
        validation_data=(X_test, y_test),
        callbacks=[reduce_lr, early_stopping]
    )

    # Evaluamos el modelo en el conjunto de test
    test_loss, test_acc = model.evaluate(X_test, y_test)
    print(f'Precisión en el conjunto de prueba: {test_acc}')
    print(f'Pérdida en el conjunto de prueba: {test_loss}')
```

La estructura del fragmento de código es muy parecida a la vista en los otros modelos. Las principales diferencias están en las capas agregadas de densidad, dropout y de salida, buscando que el modelo no sobreajuste y la regularización del propio modelo.

6.3.7. ¿Qué es Inception GoogLeNet?

Inception, también conocida como GoogleNet, representa una arquitectura de red neuronal convolucional (CNN) desarrollada por Google, la cual revolucionó el campo de la visión por computadora. Destaca su módulo de "Inception", esta arquitectura emplea múltiples filtros de diferentes tamaños en paralelo para capturar y procesar características a diferentes escalas espaciales. Este enfoque permite a la red aprender representaciones más ricas y complejas de las imágenes de entrada, optimizando así su capacidad de extracción de características.

Figura 29. Esquema general de Inception GoogLeNet V4



Fuente: https://www.researchgate.net/figure/The-overall-schema-of-GoogLeNet-Inception-V4-model-The-inception-and-reduction-models_fig2_354236457

Inception se ha diseñado específicamente para mejorar la eficiencia computacional y el rendimiento en diversas tareas de visión por computadora. Su efectividad ha sido ampliamente demostrada en aplicaciones que van desde la clasificación de imágenes hasta la detección de objetos, consolidándose como una herramienta fundamental en la investigación y desarrollo de sistemas de reconocimiento visual de vanguardia.

6.3.8. Codificamos Inception GoogLeNet

Seguidamente, se muestra la implementación del modelo Inception GoogLeNet:

```
1. import pandas as pd
2. import numpy as np
3. from sklearn.preprocessing import LabelEncoder
4. from tensorflow.keras.applications import InceptionV3
5. from tensorflow.keras.models import Sequential
6. from tensorflow.keras.layers import Dense, GlobalAveragePooling
   2D, Dropout
```

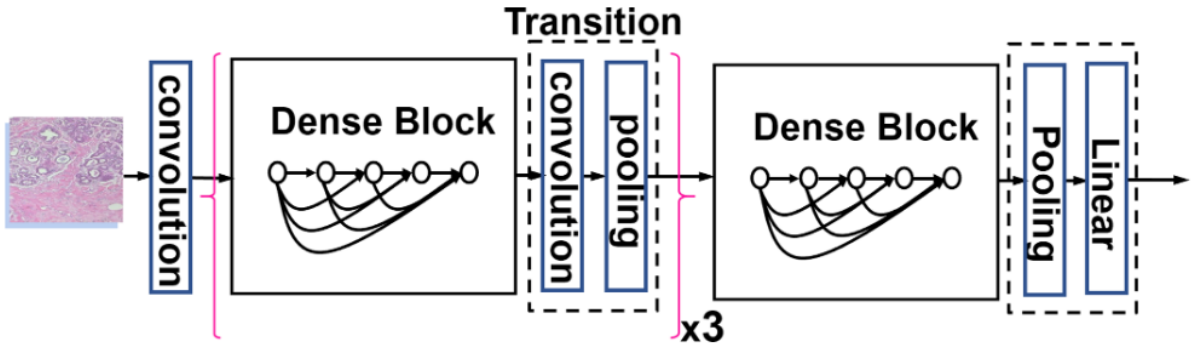
```
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

    # Creamos el modelo de InceptionV3 preentrenado
    base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(128, 128, 3))

    # Congelamos las capas convolucionales de InceptionV3
    for layer in base_model.layers:
        layer.trainable = False

    # Añadimos capas adicionales al modelo
    model = Sequential([
        base_model,
        GlobalAveragePooling2D(),
        Dense(256, activation='relu'),
        Dropout(0.5),
        Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(1, activation='sigmoid')  # Solo necesitas una neurona para la clasificación binaria
    ])

    # Compilamos el modelo
    model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy'])

    # Reducimos la tasa de aprendizaje cuando la métrica de validación se estanca
    reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=0.0001)

    # Configuramos la detención temprana (Early Stopping)
    early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

    # Entrenamos el modelo
    history = model.fit(
        X_train, y_train,
        epochs=100,
        batch_size=32,
        validation_data=(X_test, y_test),
        callbacks=[reduce_lr, early_stopping]
    )

    # Evaluamos el modelo en el conjunto de test
    test_loss, test_acc = model.evaluate(X_test, y_test)
    print(f'Precisión en el conjunto de prueba: {test_acc}')
    print(f'Pérdida en el conjunto de prueba: {test_loss}')
```

Añadimos, como en los otros modelos para el ajuste de hiperparámetros, capas adicionales al modelo, incluyendo capas de agrupación global, capas densas con activación ReLU y dropout para regularización.

6.3.9. ¿Qué es DenseNet?

DenseNet, específicamente DenseNet121 en nuestro caso, representa una arquitectura de red neuronal convolucional (CNN) reconocida por su estructura densamente conectada. En DenseNet, **cada capa está conectada directamente a todas las capas subsiguientes** en una disposición densa, lo que promueve la propagación de información y la reutilización eficiente de características en toda la red.

Figura 30. Arquitectura DenseNet-121 con imagen histopatológica de cáncer de mama



Fuente: https://doi.org/10.1371/journal.pone.0232127.g001

Esta característica de conectividad densa ha demostrado mejorar la eficiencia del entrenamiento, mitigar el problema de desvanecimiento del gradiente y promover un aprendizaje más profundo y representativo.

6.3.10. Codificamos DenseNet

Se ha elegido DenseNet121 debido a su equilibrio entre complejidad y rendimiento. Aunque existen variantes más profundas de DenseNet, como DenseNet169 o DenseNet201, DenseNet121 nos ofrece un rendimiento competitivo en una variedad de tareas de visión por computadora, mientras que es computacionalmente más eficiente y menos propenso al sobreajuste en comparación con sus contrapartes más profundas. Esta elección se alinea con nuestro objetivo de desarrollar un modelo eficaz y escalable para nuestra tarea específica, garantizando al mismo tiempo una implementación práctica y eficiente en términos de recursos computacionales:

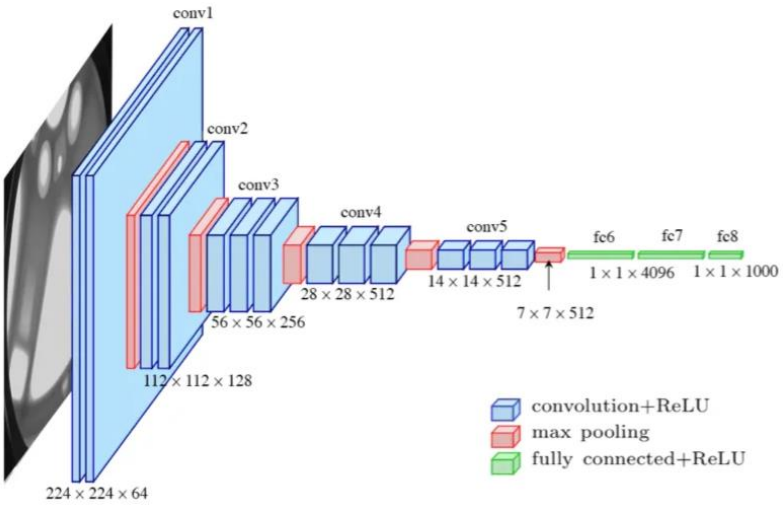
```
1. import pandas as pd
2. import numpy as np
3. from sklearn.preprocessing import LabelEncoder
4. from tensorflow.keras.applications import DenseNet121
5. from tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau


    # Creamos el modelo DenseNet121 preentrenado
    base_model = DenseNet121(weights='imagenet', include_top=False, input_shape=(128, 128, 3))

    # Congelamos las capas convolucionales de DenseNet121
    for layer in base_model.layers:
        layer.trainable = False

    # Añadimos capas adicionales al modelo
    model = Sequential([
        base_model,
        GlobalAveragePooling2D(),
        Dense(256, activation='relu'),
        Dropout(0.5),
        Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(1, activation='sigmoid')  # Solo necesitas una neurona para la clasificación binaria
    ])

    # Compilamos el modelo
    model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy'])

    # Reducimos la tasa de aprendizaje cuando la métrica de validación se estanca
    reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=0.0001)

    # Configuramos la detención temprana (Early Stopping)
    early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

    # Entrenamos el modelo
    history = model.fit(
        X_train, y_train,
        epochs=100,
        batch_size=32,
        validation_data=(X_test, y_test),
        callbacks=[reduce_lr, early_stopping]
    )

    # Evaluamos el modelo en el conjunto de test
    test_loss, test_acc = model.evaluate(X_test, y_test)
    print(f'Precisión en el conjunto de prueba: {test_acc}')
    print(f'Pérdida en el conjunto de prueba: {test_loss}')
```

6.3.11. ¿Qué es VGG (Visual Geometry Group)?

VGG, desarrollada por el grupo de investigación Visual Geometry Group, de la Universidad de Oxford, es una CNN ampliamente reconocida por su simplicidad y eficacia en la extracción de características visuales. En particular, VGG16 y VGG19 son dos variantes populares que se caracterizan por su estructura de capas convolucionales y su profundidad relativamente alta.

Figura 31. Arquitectura VGG



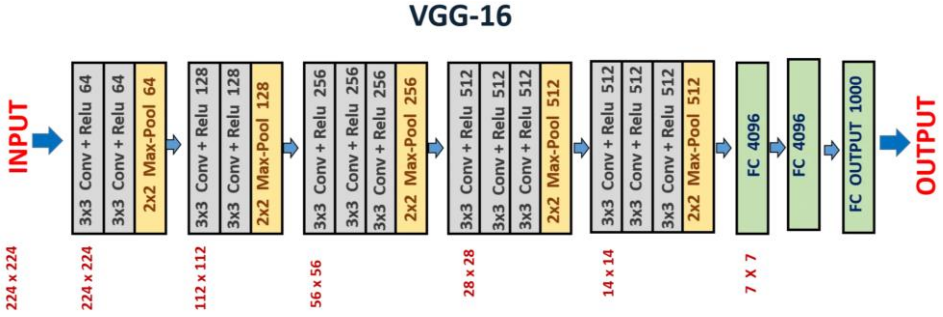
Fuente: https://medium.com/@siddheshb008/vgg-net-architecture-explained-71179310050f

Estas redes se destacan por su diseño uniforme, donde las convoluciones repetidas de tamaño pequeño se utilizan para aprender características en diferentes niveles de abstracción.

6.3.12. Codificamos VGG

En nuestro estudio, hemos elegido VGG16 debido a su rendimiento sólido y su relativa facilidad de implementación.

Figura 32. Arquitectura de capas de VGG-16



Fuente: https://medium.com/@siddheshb008/vgg-net-architecture-explained-71179310050f

Aunque VGG19 ofrece una mayor profundidad, lo que podría resultar en una representación más detallada de las características, hemos optado por VGG16 por su equilibrio entre complejidad y eficiencia computacional.

```
1. import pandas as pd
2. import numpy as np
3. from sklearn.preprocessing import LabelEncoder
4. from tensorflow.keras.applications import VGG16
5. from tensorflow.keras.models import Sequential
6. from tensorflow.keras.layers import Dense, GlobalAveragePooling
   2D, Dropout
7. from tensorflow.keras.optimizers import Adam
8. from tensorflow.keras.callbacks import EarlyStopping, ReduceLRO
   nPlateau
9.
10.
11.     # Creamos el modelo VGG16 preentrenado
12.     base_model = VGG16(weights='imagenet', include_top=False,
    input_shape=(128, 128, 3))
13.
14.     # Congelamos las capas convolucionales de VGG16
15.     for layer in base_model.layers:
16.         layer.trainable = False
17.
18.     # Añadimos capas adicionales al modelo
19.     model = Sequential([
20.         base_model,
21.         GlobalAveragePooling2D(),
22.         Dense(256, activation='relu'),
23.         Dropout(0.5),
24.         Dense(128, activation='relu'),
25.         Dropout(0.5),
26.         Dense(1, activation='sigmoid')  # Solo necesitas una
   neurona para la clasificación binaria
27.     ])
28.
29.     # Compilamos el modelo
30.     model.compile(optimizer=Adam(learning_rate=0.001), loss='
   binary_crossentropy', metrics=['accuracy'])
31.
32.     # Reducimos la tasa de aprendizaje cuando la métrica de
   validación se estanca
33.     reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=
   0.2, patience=5, min_lr=0.0001)
34.
35.     # Configuramos la detención temprana (Early Stopping)
36.     early_stopping = EarlyStopping(monitor='val_loss', patien
   ce=10, restore_best_weights=True)
37.
```

```
    # Entrenamos el modelo
    history = model.fit(
        X_train, y_train,
        epochs=100,
        batch_size=32,
        validation_data=(X_test, y_test),
        callbacks=[reduce_lr, early_stopping]
    )

    # Evaluamos el modelo en el conjunto de test
    test_loss, test_acc = model.evaluate(X_test, y_test)
    print(f'Precisión en el conjunto de prueba: {test_acc}')
    print(f'Pérdida en el conjunto de prueba: {test_loss}')
```

Esta elección se alinea con nuestro objetivo de desarrollar un modelo robusto y efectivo para nuestra tarea específica, manteniendo al mismo tiempo una implementación práctica y eficiente en términos de recursos computacionales.

6.3.13. ¿Qué es EfficientNet?

EfficientNet es una familia de arquitecturas de CNN, diseñada por **Google Brain**. Destacada por su eficiencia en términos de recursos computacionales y su alto rendimiento en una variedad de tareas de visión por computadora, EfficientNet utiliza un enfoque de **escalado compuesto que optimiza simultáneamente la profundidad, el ancho y la resolución de la red**. Esto permite que el modelo se adapte de manera eficiente a diferentes restricciones de recursos, como el tiempo de entrenamiento y la memoria, mientras mantiene un rendimiento competitivo.

6.3.14. Codificamos EfficientNet

Hemos optado por utilizar **EfficientNetB0** en nuestro proyecto, debido a su eficiencia y alto rendimiento en tareas de clasificación de imágenes.

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.applications import EfficientNetB0
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, GlobalAveragePooling
2D, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLRO
nPlateau

    # Creamos el modelo EfficientNetB0 preentrenado
    base_model = EfficientNetB0(weights='imagenet', include_t
op=False, input_shape=(128, 128, 3))
```

```
12.
13.     # Congelamos las capas convolucionales de EfficientNetB0
14.     for layer in base_model.layers:
15.         layer.trainable = False
16.
17.     # Añadimos capas adicionales al modelo
18.     model = Sequential([
19.         base_model,
20.         GlobalAveragePooling2D(),
21.         Dense(256, activation='relu'),
22.         Dropout(0.5),
23.         Dense(128, activation='relu'),
24.         Dropout(0.5),
25.         Dense(1, activation='sigmoid')  # Solo necesitas una
    neurona para la clasificación binaria
26.     ])
27.
28.     # Compilamos el modelo
29.     model.compile(optimizer=Adam(learning_rate=0.001), loss='
    binary_crossentropy', metrics=['accuracy'])
30.
31.     # Reducimos la tasa de aprendizaje cuando la métrica de
    validación se estanca
32.     reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=
    0.2, patience=5, min_lr=0.0001)
33.
34.     # Configuramos la detención temprana (Early Stopping)
35.     early_stopping = EarlyStopping(monitor='val_loss', patien
    ce=10, restore_best_weights=True)
36.
37.     # Entrenamos el modelo
38.     history = model.fit(
39.         X_train, y_train,
40.         epochs=100,
41.         batch_size=32,
42.         validation_data=(X_test, y_test),
43.         callbacks=[reduce_lr, early_stopping]
44.     )
45.
46.     # Evaluamos el modelo en el conjunto de test
47.     test_loss, test_acc = model.evaluate(X_test, y_test)
48.     print(f'Precisión en el conjunto de prueba: {test_acc}')
49.     print(f'Pérdida en el conjunto de prueba: {test_loss}')
```

La arquitectura preentrenada de EfficientNetB0, combinada con capas adicionales que hemos agregado para adaptar el modelo a nuestras necesidades específicas, nos brinda una solución robusta y escalable para nuestro problema de clasificación binaria.

6.3.15. Analizamos los resultados

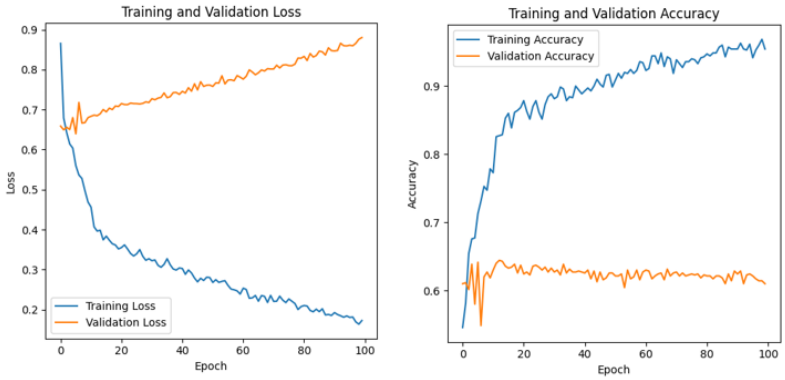
Una vez que hemos visto y explicado otras arquitecturas y modelos CNN vamos a ver los resultados y comprarlos entre sí, tanto la tasa de precisión como la de pérdida, así como una tabla comparativa entre todas y sus gráficas durante el entrenamiento y la validación:

Tabla 4. Comparativa resultados distintas arquitecturas CNN

Arquitectura	Época de Parada	Precisión en Prueba	Pérdida en Prueba	Total Params	Trainable Params	Non-Trainable Params	Tamaño Total (MB)
MobilenetV2	100	0.610	0.880	2,422,081	164,097	2,257,984	9.24
MobilenetV3	83	0.609	0.686	1,119,986	180,866	939,120	4.27
ResNet50	11	0.609	0.670	24,145,281	557,569	23,587,712	92.11
Inception G.	14	0.569	0.671	22,360,353	557,569	21,802,784	85.30
DenseNet121	16	0.601	0.642	7,332,929	295,425	7,037,504	27.97
VGG	28	0.601	0.642	14,879,041	164,353	14,714,688	56.76
EfficientNet	11	0.609	0.674	4,410,532	360,961	4,049,571	16.82

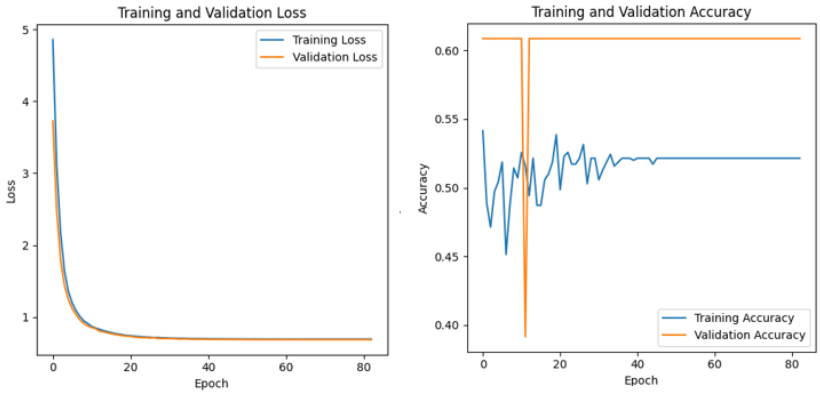
Observando los datos finales de las distintas arquitecturas CNN vemos que, si bien algunas de ellas alcanzan una alta precisión en el conjunto de prueba, como MobilenetV2, MobilenetV3 y EfficientNet, también muestran signos de sobreajuste. Esto se refleja en el hecho de que estas arquitecturas se detienen antes de completar todas las épocas de entrenamiento debido al early stopping. Específicamente, MobilenetV2, MobilenetV3 y EfficientNet alcanzan una precisión en el conjunto de prueba del 0.610, 0.609 y 0.609 respectivamente, mientras que sus pérdidas en el conjunto de prueba son 0.880, 0.686 y 0.674 respectivamente. Además, cuando identificamos el momento de detención del entrenamiento, esto nos proporciona información valiosa sobre la capacidad de adaptación del modelo y su rendimiento en datos de validación, lo que destaca la importancia del early stopping como una técnica efectiva para mejorar la capacidad de generalización de los modelos de redes neuronales.

Figura 33. Diagramas del entrenamiento y validación para la pérdida y precisión de MobileNetV2



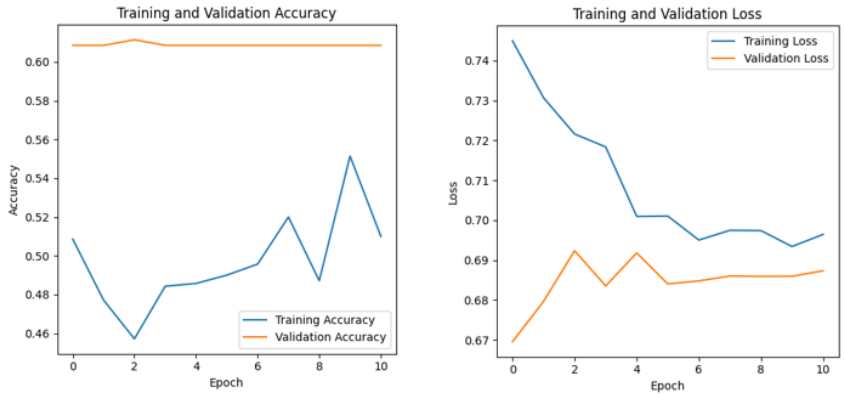
- **Rendimiento del modelo durante el entrenamiento**: Durante las primeras épocas, nuestro modelo MobileNetV2 logra una disminución significativa en la pérdida de entrenamiento, lo que sugiere una asimilación efectiva de los patrones presentes en los datos de entrenamiento. Sin embargo, el aumento continuo en la pérdida de validación indica que el modelo comienza a sobreajustarse a los datos de entrenamiento después de la época 20.
- **Capacidad de generalización**: A pesar de la mejora en la precisión de entrenamiento, la precisión de validación se mantiene relativamente constante y baja en comparación con la precisión de entrenamiento. Esto indica que el modelo MobileNetV2 puede tener dificultades para generalizar correctamente a nuevos datos, lo que se refleja en una precisión final del 0.610 y una pérdida de 0.880 en el conjunto de prueba.

Figura 34. Diagramas del entrenamiento y validación para la pérdida y precisión de MobileNetV3



- **Rendimiento durante el entrenamiento**: La disminución gradual de la pérdida tanto en el conjunto de entrenamiento como en el de validación indica que nuestro modelo fue capaz de aprender efectivamente los patrones presentes en los datos de entrenamiento y generalizarlos a datos nuevos.
- **Estabilidad de la precisión**: Aunque la precisión de entrenamiento alcanzó un nivel relativamente estable alrededor de 0.53, la precisión de validación mostró una variabilidad mayor, con fluctuaciones durante las primeras épocas seguidas de un pico pronunciado en la época 16. Sin embargo, la precisión de validación logró recuperarse posteriormente y alcanzar un nivel similar al de entrenamiento hacia el final del proceso.

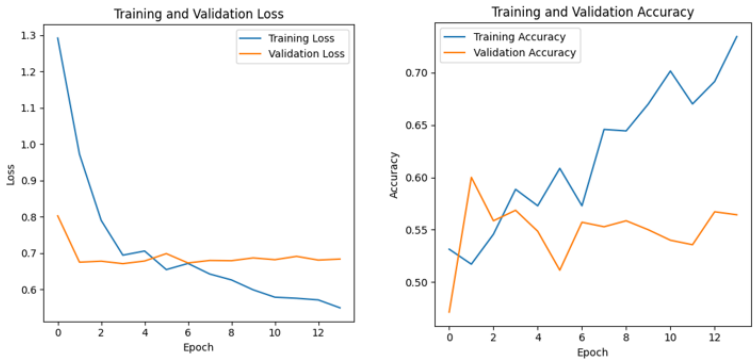
Figura 35. Diagramas del entrenamiento y validación para la pérdida y precisión de ResNet



Rendimiento del Modelo: Nuestro modelo ResNet ha demostrado un rendimiento razonablemente bueno durante el entrenamiento, con una pérdida final de 0.670. Esto sugiere que el modelo ha aprendido efectivamente de los datos de entrenamiento y ha logrado reducir su error durante el proceso de aprendizaje.

Estabilidad de la Precisión: A pesar de una disminución inicial en la precisión del conjunto de entrenamiento, el modelo logra estabilizarse alrededor de una precisión del 0.51 en la época 11. Esto indica que nuestro modelo es capaz de clasificar correctamente aproximadamente el 51% de los ejemplos del conjunto de entrenamiento.

Figura 36. Diagramas del entrenamiento y validación para la pérdida y precisión de Inception Google

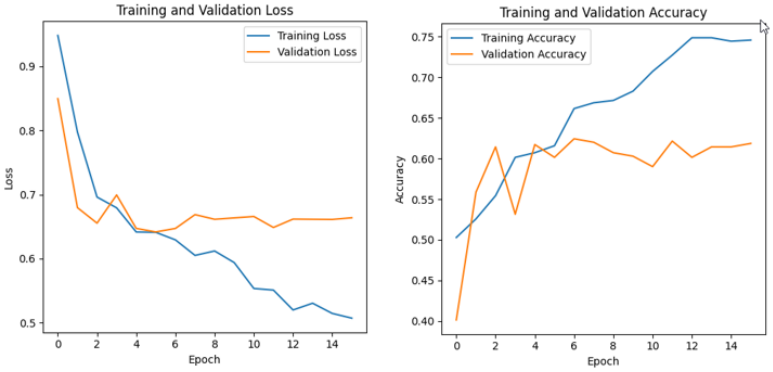


Rendimiento del Modelo: Aunque la pérdida del modelo ha disminuido gradualmente durante el entrenamiento, con una reducción constante desde un valor inicial de 1.3 hasta 0.55 al final del entrenamiento, la pérdida final de 0.671 aún es relativamente alta. Esto sugiere que el modelo puede beneficiarse de un ajuste adicional para mejorar su capacidad para hacer predicciones precisas.

Estabilidad de la Precisión: Si bien la precisión del modelo ha experimentado un aumento constante a lo largo del entrenamiento, con un incremento desde 0.53 hasta 0.75 en el conjunto de entrenamiento y desde 0.45 hasta 0.57 en el conjunto de validación, la precisión final del modelo en el conjunto de validación es relativamente baja. Esto indica que el modelo puede tener dificultades para generalizar a datos no vistos.

Consideraciones Adicionales: Dada la discrepancia entre la precisión en el conjunto de entrenamiento y en el de validación, es posible que el modelo esté sobre ajustando los datos de entrenamiento.

Figura 37. Diagramas del entrenamiento y validación para la pérdida y precisión de DenseNet



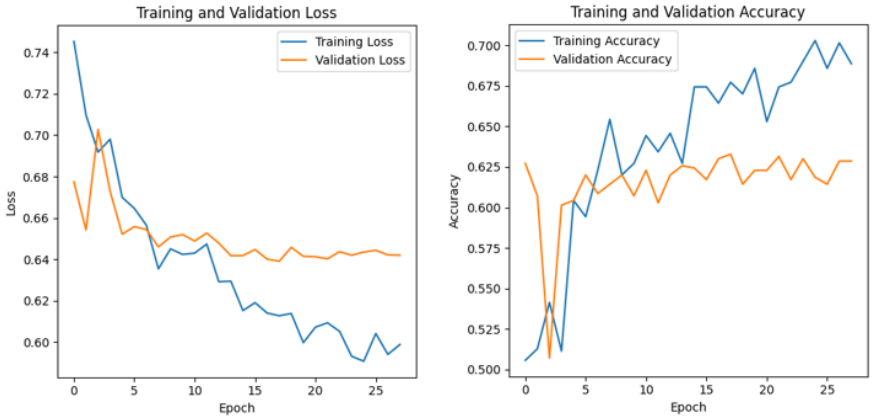
Rendimiento durante el Entrenamiento: El modelo muestra una disminución constante en la pérdida tanto en el conjunto de entrenamiento como en el de validación a lo largo de las épocas. Esta tendencia indica que el modelo está aprendiendo de manera efectiva y mejorando su capacidad para hacer predicciones precisas.

La pérdida final del modelo en el conjunto de validación es del 0.642, lo que sugiere que el modelo logra una generalización aceptable, ya que la pérdida es relativamente baja y similar en ambos conjuntos de datos.

Estabilidad de la Precisión: La gráfica de precisión muestra un aumento gradual en la precisión tanto en el conjunto de entrenamiento como en el de validación a medida que avanza el entrenamiento. Este comportamiento indica que nuestro modelo es capaz de capturar eficazmente la estructura de los datos y realizar predicciones precisas.

La precisión final del modelo en el conjunto de validación es del 0.601, lo que indica que el modelo es capaz de hacer predicciones con una precisión razonable en datos no vistos.

Figura 38. Diagramas del entrenamiento y validación para la pérdida y precisión de VGG

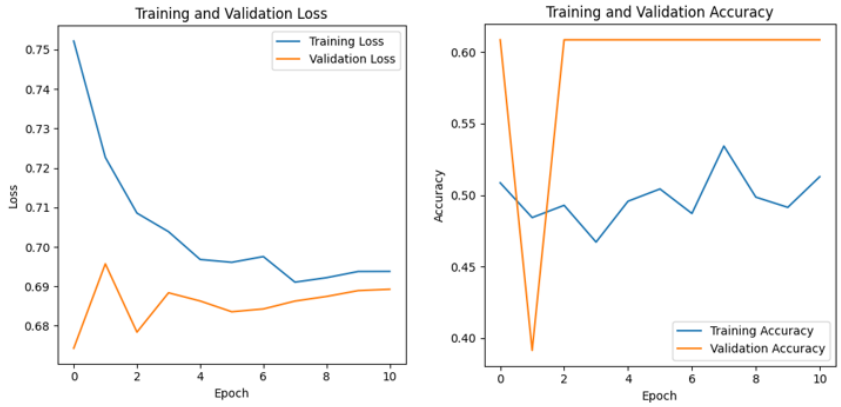


A partir de las gráficas de pérdida y precisión de nuestro modelo VGG, con una precisión final de 0.601 y una pérdida final de 0.642, podemos concluir que:

Rendimiento durante el Entrenamiento: El descenso gradual de la pérdida en el conjunto de entrenamiento indica que el modelo está aprendiendo de manera efectiva de los datos de entrenamiento. La estabilización de la pérdida y la precisión en el conjunto de validación sugiere que el modelo no está sobre ajustando significativamente los datos de entrenamiento y que está generalizando bien a datos no vistos.

Estabilidad de la Precisión: A pesar de las fluctuaciones iniciales, la precisión en el conjunto de validación muestra una tendencia ascendente hacia el final del entrenamiento, lo que indica una mejora gradual en la capacidad del modelo para clasificar correctamente las muestras de validación.

Figura 39. Diagramas del entrenamiento y validación para la pérdida y precisión de EfficieNet



Rendimiento durante el Entrenamiento: Observamos una tendencia decreciente en la pérdida del conjunto de entrenamiento a medida que avanzan las épocas, lo que sugiere una efectiva asimilación de los datos de entrenamiento por parte del modelo.

Por otro lado, la pérdida en el conjunto de validación muestra una estabilización hacia el final del entrenamiento, indicando una adecuada generalización del modelo sobre datos no observados.

Estabilidad de la Precisión: Esta estabilidad (salvo por el pico decreciente) nos está mostrando un claro sobre ajuste en el modelo a la hora de validar la precisión.

7. Conclusiones y trabajo futuro

7.1. Conclusiones del trabajo

Basándonos en los hallazgos obtenidos y en consonancia con los objetivos planteados inicialmente en este trabajo de investigación, hemos podido constatar la consecución exitosa del propósito fundamental: desarrollar una red neuronal capaz de realizar diagnósticos precisos sobre la detección de cáncer en mamografías, con un prometedor porcentaje de precisión que más adelante explicaré como podría ser mejorado. A lo largo del estudio, hemos logrado alcanzar una serie de objetivos más específicos, los cuales incluyeron la mejora progresiva de la precisión del modelo mediante la implementación de estrategias avanzadas, como el uso de capas convolucionales y técnicas de regularización, así como la aplicación del aprendizaje transferido. Estos enfoques demostraron ser efectivos para incrementar la capacidad predictiva de nuestro modelo original de base y mejorar su habilidad para generalizar patrones de detección.

También destacar otros conocimientos adquiridos transversales que no estaban ni como objetivo general ni específico de este trabajo y es trabajar con soltura y profundizar con las distintas librerías utilizadas y las distintas versiones de Python, tensorflow, CUDA, cuDNN, así como distintos entornos de trabajo, ya sea el nuestro (IDE IntelliJ IDEA) o Google Colab.

No obstante, resulta relevante resaltar ciertos matices que emergieron en la fase final de este trabajo. En este sentido, observamos una tendencia generalizada hacia el sobreajuste en todos los modelos evaluados, a pesar de los esfuerzos destinados a realizar ajustes meticulosos de los hiperparámetros. Esta situación se tradujo en una ausencia de mejoras sustanciales en los indicadores de rendimiento, manteniendo los modelos en niveles de precisión y pérdida similares. Este hallazgo plantea interrogantes sobre la idoneidad de la ingeniería de datos realizada a la hora de desarrollar y diseñar los csv de datos con los que hemos trabajado durante este TFE y nos conduce a reflexionar sobre posibles direcciones para futuras investigaciones y desarrollos en este ámbito.

7.2. Líneas de trabajo futuro

A la vista de los resultados obtenidos y las reflexiones surgidas durante la investigación, hemos identificado diversas líneas de trabajo futuro que podrían enriquecer y ampliar el ámbito de

detección de cáncer a través del análisis de mamografías. Una posible área de mejora radica en **la exploración de enfoques más avanzados de regularización y optimización de hiperparámetros** con el fin de mitigar el problema del sobreajuste, que ha emergido como una limitación significativa en nuestros modelos. La implementación de técnicas de regularización más sofisticadas, como la regularización L1 o L2, así como estrategias de aumento de datos más elaboradas, podría contribuir a mejorar la capacidad de generalización de los modelos y reducir la discrepancia entre los conjuntos de entrenamiento y prueba.

Además, **contemplamos la posibilidad de investigar la integración de información adicional en el proceso de análisis de mamografías, como datos clínicos complementarios, imágenes de otras modalidades** (por ejemplo, resonancia magnética) **o información genómica**. La combinación de múltiples fuentes de datos podría proporcionar una visión más completa y precisa de la condición del paciente, potencialmente mejorando la precisión y confiabilidad de los diagnósticos.

Asimismo, consideramos la exploración de arquitecturas de redes neuronales más avanzadas o específicamente diseñadas para tareas de detección médica. El campo de la inteligencia artificial en medicina está en constante evolución, y la adopción de arquitecturas más recientes, como **las redes neuronales convolucionales tridimensionales** (3D CNN) o las **redes neuronales adversarias generativas** (GAN), podría abrir nuevas posibilidades para mejorar la precisión y robustez de los modelos de detección de cáncer basados en imágenes.

Finalmente, sería beneficioso profundizar en la evaluación e interpretación de los modelos desarrollados, especialmente en lo que respecta a la transparencia y explicabilidad de las decisiones del modelo. La aplicación de técnicas de interpretación de modelos, como la generación de mapas de activación o la atención visual, podría proporcionar información valiosa sobre los patrones detectados por el modelo y aumentar la confianza de los profesionales de la salud en su uso clínico.

Referencias bibliográficas

Anywhere, E. e. (2024, febrero 15). *Anywhere*. Obtenido de https://anywhere.epam.com/es/blog/preguntas-entrevista-aprendizaje-automatico

Bangar, S. (2022, junio 28). *medium*. Obtenido de https://medium.com/@siddheshb008/vgg-net-architecture-explained-71179310050f

Barrios, J. (s.f.). *Health Big Data*. Obtenido de https://www.juanbarrios.com/redes-neurales-convolucionales/

Cao, Y. e. (2021, noviembre 1). *researchgate*. Obtenido de https://www.researchgate.net/figure/The-overall-schema-of-GoogLeNet-Inception-V4-model-The-inception-and-reduction-models_fig2_354236457

Cirugias de la mama. (2017, agosto 01). Obtenido de Cáncer de mama: https://www.cirugiasdelamama.com/cancer-de-mama?lightbox=dataItem-it4rqwkj

Dagdeviren, C. e. (2023, julio 28). Conformable ultrasound breast patch for deep tissue scanning and imaging. *ScienceAdvances. https://doi.org/10.1126/sciadv.adh5325*, Vol 9, Issue 30.

es.statista. (2022, diciembre). Obtenido de Número de muertes por cáncer registrado en mujeres en España en 2021, por tipo: https://es.statista.com/estadisticas/677111/muertes-por-cancer-en-mujeres-por-tipo-en-espana/

Farnós, J. D. (2023, mayo 08). *juandon. Innovación y conocimiento*. Obtenido de https://juandomingofarnos.wordpress.com/2023/05/08/aprendizaje-profundo-y-redes-neuronales-de-juan-domingo-farnos-en-procesos-algoritmicos-de-aprendizaje-con-eduacion-disruptiva-inteligencia-artificial/

García, U. (2019, junio 25). *Future LAB*. Obtenido de https://futurelab.mx/redes%20neuronales/inteligencia%20artificial/2019/06/25/intro-a-redes-neuronales-pt-1/

Gonzales, L. (2019, noviembre 3). *medium*. Obtenido de https://medium.com/@luis_gonzales/a-look-at-mobilenetv2-inverted-residuals-and-linear-bottlenecks-d49f85c12423

Health, C. (2023, marzo 07). *cnn*. Obtenido de Images show AI detecting breast cancer 4 years before it developed: https://edition.cnn.com/videos/health/2023/03/07/artificial-intelligence-breast-cancer-detection-mammogram-cnntm-vpx.cnn

Internacional, R. (s.f.). *ruberinternacional*. Obtenido de El tipo de tumor más frecuente entre las mujeres en nuestro país: https://www.ruberinternacional.es/en/patient/pathologyes/cancer-mama

J, G. (2020, enero 02). Obtenido de Google desarrolla una inteligencia artificial para detectar el cáncer de mama "capaz de superar a los expertos humanos": https://www.xataka.com/inteligencia-artificial/google-desarrolla-inteligencia-artificial-para-detectar-cancer-mama-capaz-superar-a-expertos-humanos

Jaamour, A. &. (2023, mayo 18). *journals*. Obtenido de https://doi.org/10.1371/journal.pone.0280841

Jaamour, A. M.-B. (2023, mayo 26). *journals*. Obtenido de https://doi.org/10.1371/journal.pone.0232127.g001

Lång, K. J. (agosto de 2023). Artificial intelligence-supported screen reading versus standard double reading in the Mammography Screening with Artificial Intelligence trial (MASAI): a clinical safety analysis of a randomised, controlled, non-inferiority, single-blinded, screening acc. *The Lancet Oncology, 24*(8), 936-944. doi:https://doi.org/10.1016/S1470-2045(23)00298-X

Li, Y. W. (2020, septiembre 22). *researchgate*. Obtenido de https://www.researchgate.net/figure/MobileNetV2-Layer-vs-MobileNetV3-Block_fig4_351564462

Liu, F. Z. (2021, noviembre 01). *ResearchGate*. Obtenido de https://www.researchgate.net/figure/The-illustration-of-the-forward-process-and-error-backpropagation-in-the-SSTDP-method_fig4_355905487

Pilar-Suárez, L. B. (2023, noviembre 17). *Reproducción Asistida ORG*. Obtenido de https://www.reproduccionasistida.org/cancer-de-mama/

Salud, O. M. (2023, julio 12). *who.int*. Obtenido de https://www.who.int/es/news-room/fact-sheets/detail/breast-cancer

Sawyer-Lee, R. G. (2016). *Curated Breast Imaging Subset of Digital Database for Screening Mammography (CBIS-DDSM) [Data set]*. Obtenido de The Cancer Imaging Archive.: https://doi.org/10.7937/K9/TCIA.2016.7O02S9CY

Soberanis, M. C. (2018, febrero 13). *Medium*. Obtenido de https://medium.com/soldai/inspiraci%C3%B3n-biol%C3%B3gica-de-las-redes-neuronales-artificiales-9af7d7b906a

Suárez, J. (s.f.). *Numerentur*. Obtenido de https://numerentur.org/convolucionales/

Vera, M. I. (s.f.). *cidecame*. Obtenido de http://cidecame.uaeh.edu.mx/lcc/mapa/PROYECTO/libro17/12_etapas.html

Anexo A. Script para normalizado de ruta de las imágenes

Importación de módulos:

```
1. import os
2. import shutil
3. import glob
```

os: Proporciona funciones para interactuar con el sistema operativo, como manipulación de archivos y directorios.

shutil: Ofrece funciones para operaciones de alto nivel en archivos, como copiar y mover.

glob: Permite realizar búsquedas de patrones en rutas de archivos.

Variables de entorno:

```
1. ruta_origen = "D:\\Documentos\\0.-Universidad Ingeniería
   Informática\\TFG\\proyecto\\src\\resources\\CBIS-DDSM"
2. rutas_archivos = glob.glob(ruta_origen + "\\*")
```

ruta_origen: Ruta de origen donde se buscarán las imágenes.

rutas_archivos: Lista de rutas de todos los archivos y carpetas directas dentro de ruta_origen.

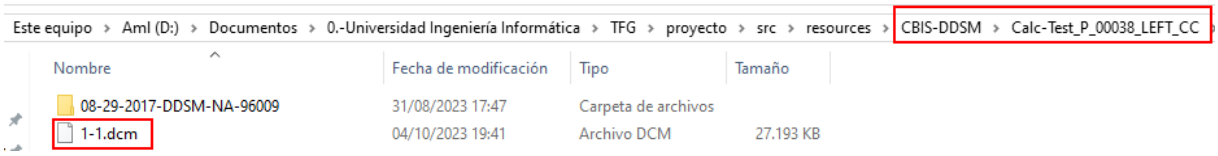
Función buscar_y_copiar_imagenes:

```
1.  def buscar_y_copiar_imagenes(ruta):
2.      # Lista para almacenar las rutas de las imágenes .dcm encontradas
3.      imagenes = []
4.
5.      # Busca archivos de imagen (dcm) en la ruta de origen y subdirectorios
6.      for root, dirs, files in os.walk(ruta):
7.          for file in files:
8.              if file.endswith('.dcm'):
9.                  imagenes.append(os.path.join(root, file))
10.
11.     print("Imágenes encontradas:")
12.     for imagen in imagenes:
13.         print(imagen)
14.         # Copiar la imagen a la ruta destino
15.         shutil.copy(imagen, ruta)
16.         print("Imagen copiada a:", ruta)
```

La función declara un array llamado “imagenes[]” donde almacenaremos las rutas de las imágenes DICOM encontradas. En el primer *for* utilizamos “*os.walk*” para recorrer de manera recursiva la estructura de directorios desde la “ruta” proporcionada. Con el segundo *for* que se encuentra anidado recorremos todos los archivos encontrados en dichas rutas y con el *if* cribamos por los archivos con extensión “.dcm”. Agregamos las rutas de los archivos con extensión “.dcm” a la lista “imagenes[]”.

En el tercer *for* recorremos el array de imágenes y utilizando la función **shutil.copy** copiamos las imágenes a la ruta donde queremos que se encuentren es decir en:

Figura 40. Imagen copiada a la ruta principal



Llamar a la función principal:

```
# Itera sobre todas las rutas encontradas
for ruta in rutas_archivos:
    buscar_y_copiar_imagenes(ruta)
```

Con este *for* iteramos sobre todas las rutas encontradas en “rutas_archivos” y llamamos a la función principal ya descrita en el párrafo anterior.

Figura 41. Ejemplo de salida por consola al ejecutar el script

```
Imágenes encontradas:
D:\Documentos\0.-Universidad Ingeniería Informática\TFG\proyecto\src\resources\CBIS-DDSM\Mass-Training_P_00420_RIGHT_CC_1\1-1.dcm
Imagen copiada a: D:\Documentos\0.-Universidad Ingeniería Informática\TFG\proyecto\src\resources\CBIS-DDSM\Mass-Training_P_00420_RIGHT_CC_1
D:\Documentos\0.-Universidad Ingeniería Informática\TFG\proyecto\src\resources\CBIS-DDSM\Mass-Training_P_00420_RIGHT_CC_1\1-2.dcm
Imagen copiada a: D:\Documentos\0.-Universidad Ingeniería Informática\TFG\proyecto\src\resources\CBIS-DDSM\Mass-Training_P_00420_RIGHT_CC_1
D:\Documentos\0.-Universidad Ingeniería Informática\TFG\proyecto\src\resources\CBIS-DDSM\Mass-Training_P_00420_RIGHT_CC_1\07-21-2016-DDSM-NA-06434\1.000000-ROI mask images-20367\1-1.dcm
Imagen copiada a: D:\Documentos\0.-Universidad Ingeniería Informática\TFG\proyecto\src\resources\CBIS-DDSM\Mass-Training_P_00420_RIGHT_CC_1
D:\Documentos\0.-Universidad Ingeniería Informática\TFG\proyecto\src\resources\CBIS-DDSM\Mass-Training_P_00420_RIGHT_CC_1\07-21-2016-DDSM-NA-06434\1.000000-ROI mask images-20367\1-2.dcm
Imagen copiada a: D:\Documentos\0.-Universidad Ingeniería Informática\TFG\proyecto\src\resources\CBIS-DDSM\Mass-Training_P_00420_RIGHT_CC_1
```

Anexo B. Estructura completa de las tablas de test y training

A continuación, se muestra el número y nombre completo de las columnas de los csv utilizando pandas (los cuatro csv tienen la misma estructura de columnas por lo que solo se detallará un solo resultado):

```
1. import pandas as pd
2. pd.set_option('display.max_colwidth', None)
3. # Cargar datos de cáncer de masa
4. df_mass_train = pd.read_csv('./resources/csv/mass_case_description_train_set.csv')
5. df_mass_test = pd.read_csv('./resources/csv/mass_case_description_test_set.csv')
6.
7. # Cargar datos de cáncer de calcificación
8. df_calc_train = pd.read_csv('./resources/csv/calc_case_description_train_set.csv')
9. df_calc_test = pd.read_csv('./resources/csv/calc_case_description_test_set.csv')

1. print(df_mass_train.columns)
2. print(df_mass_test.columns)
3. print(df_calc_train.columns)
4. print(df_calc_test.columns)
```

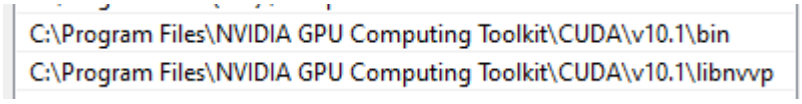
```
Index(['patient_id', 'breast_density', 'left or right breast', 'image view',
       'abnormality id', 'abnormality type', 'mass shape', 'mass margins',
       'assessment', 'pathology', 'subtlety', 'image file path',
       'cropped image file path', 'ROI mask file path'],
      dtype='object')
```

Anexo C. Configuración del entorno para poder utilizar la GPU

Para poder utilizar la GPU con Tensorflow, primero, nos tenemos que asegurar de que nuestro sistema cumple con los requisitos mínimos para la instalación de CUDA y cuDNN, incluyendo una tarjeta gráfica compatible con CUDA y un sistema operativo adecuado. En este caso, la tarjeta gráfica que utilizamos es la **NVIDIA GeForce GTX 1660 SUPER** que es compatible con CUDA.

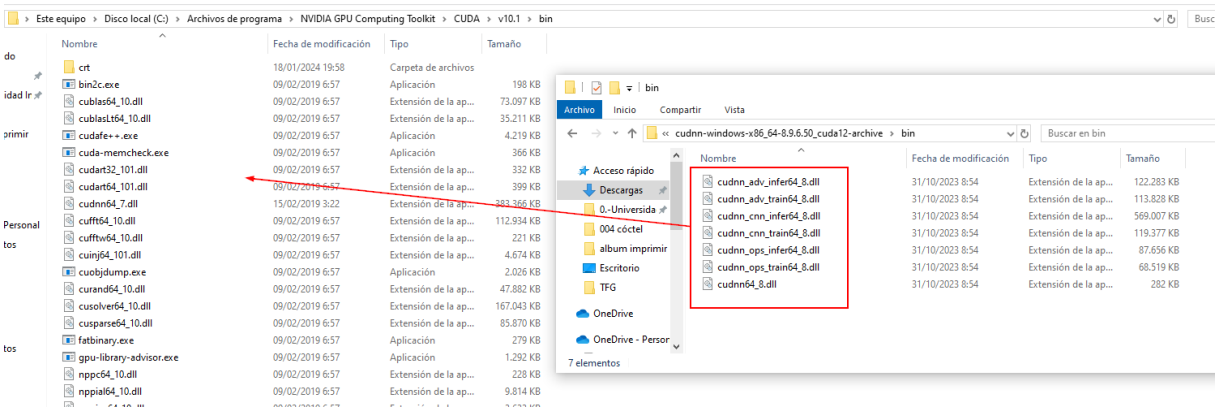
Para saber que versión de CUDA es compatible con nuestra tarjeta gráfica debemos ir a *https://developer.nvidia.com/cuda-gpus* y buscar nuestra versión. Una vez que sabemos la versión de CUDA la descargamos (en nuestro caso es CUDA 10.1) desde el sitio web oficial de NVIDIA y procedemos a su instalación. Durante este proceso, agregaremos la ruta de CUDA al PATH del sistema y configuraremos las variables de entorno según las indicaciones.

Figura 42. Variables del sistema



Una vez completada la instalación de CUDA, procederemos a descargar cuDNN. Para saber que versión de cuDNN es compatible con nuestro CUDA nos deberemos dirigir a la página oficial de Tensorflow. Nuestra versión compatible es la 7.5.0 que procederemos a descargar desde la página oficial de NVIDIA. Una vez descargado el fichero lo descomprimimos y copiamos los archivos en la carpeta de instalación de CUDA del sistema.

Figura 43. Copiamos los binarios de cuDNN a la ruta del sistema de CUDA



Finalmente, para utilizar una versión correcta de TensorFlow con CUDA y cuDNN, iremos nuevamente a la página oficial de Tensorflow y allí con nuestra versión de CUDA y cuDNN seleccionamos la versión que podemos utilizar.

Figura 44. Versión correcta de Tensorflow

Version	Python version	Compiler	Build tools	cuDNN	CUDA
tensorflow-2.3.0	3.5-3.8	GCC 7.3.1	Bazel 3.1.0	7.6	10.1
tensorflow-2.2.0	3.5-3.8	GCC 7.3.1	Bazel 2.0.0	7.6	10.1
tensorflow-2.1.0	2.7, 3.5-3.7	GCC 7.3.1	Bazel 0.27.1	7.6	10.1
tensorflow-2.0.0	2.7, 3.3-3.7	GCC 7.3.1	Bazel 0.26.1	7.4	10.0
tensorflow_gpu-1.15.0	2.7, 3.3-3.7	GCC 7.3.1	Bazel 0.26.1	7.4	10.0
tensorflow_gpu-1.14.0	2.7, 3.3-3.7	GCC 4.8	Bazel 0.24.1	7.4	10.0
tensorflow_gpu-1.13.1	2.7, 3.3-3.7	GCC 4.8	Bazel 0.19.2	7.4	10.0
tensorflow_gpu-1.12.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.15.0	7	9
tensorflow_gpu-1.11.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.15.0	7	9
tensorflow_gpu-1.10.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.15.0	7	9
tensorflow_gpu-1.9.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.11.0	7	9
tensorflow_gpu-1.8.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.10.0	7	9
tensorflow_gpu-1.7.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.9.0	7	9

Crearemos un nuevo entorno virtual y luego instalaremos TensorFlow, asegurándonos de que la versión sea compatible con CUDA 10.1 y cuDNN 7.5.0. Después de la instalación, verificaremos que TensorFlow reconozca correctamente la GPU ejecutando un script simple que utilice TensorFlow en el entorno virtual configurado. Si todo se ha configurado correctamente, podremos utilizar TensorFlow con aceleración GPU utilizando CUDA y cuDNN en nuestra tarjeta GeForce GTX 1660 SUPER y se vería así:

Figura 45. GPU configurada y lista para usar