

UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

GRADO EN INGENIERÍA INFORMÁTICA EN
TECNOLOGÍAS DE LA INFORMACIÓN



"CLASIFICACIÓN DE IMÁGENES
MÉDICAS CON DISPOSITIVOS MÓVILES
UTILIZANDO REDES NEURONALES
CONVOLUCIONALES"

TRABAJO FIN DE GRADO

Junio - 2024

AUTOR: José Antonio Juárez Limorte

DIRECTOR: Antonio Peñalver Benavent

Resumen

El uso de la inteligencia artificial ha aumentado significativamente en los últimos años debido a una combinación de múltiples factores. Este Trabajo de Fin de Grado comienza con un estudio de esta rama de la informática, desde sus orígenes hasta la actualidad.

Sin embargo, el objetivo principal de este Trabajo de Fin de Grado ha sido llevar a cabo la creación de una red neuronal convolucional para clasificar imágenes médicas. En última instancia esta red neuronal será ejecutada en una aplicación para dispositivos móviles Android, por este motivo la red se ha optimizado para dispositivos móviles.

La red neuronal convolucional ha sido implementada mediante la librería de aprendizaje automático PyTorch de Python y la optimización para dispositivos móviles incluye la cuantificación de la red neuronal para reducir uso de memoria y proporcionar una mayor velocidad de inferencia.



Agradecimientos

Quiero expresar mi más sincero agradecimiento a mi familia, por apoyarme en todo momento.

Quiero agradecer a los profesores de la titulación por todo lo que me han enseñado y por su dedicación. En especial quiero agradecer a mi tutor, Antonio Peñalver por su apoyo y guía para realizar este TFG.



Índice

CAPÍTULO 1: INTRODUCCIÓN.....	11
1.1 Introducción a la visión por computador.....	12
1.1.1 Clasificación de imágenes mediante inteligencia artificial.....	12
1.2 Inteligencia artificial en la medicina.....	12
1.2.1 Inteligencia artificial para las imágenes médicas.....	13
1.3 Inteligencia artificial en dispositivos móviles.....	14
CAPÍTULO 2: ESTADO DE LA CUESTIÓN.....	15
2.1 Inteligencia artificial.....	16
2.1.1 Categorías de la IA.....	16
2.1.2 Subconjuntos de la IA.....	17
2.1.3 Aprendizaje supervisado y no supervisado.....	18
2.2 ¿Por qué ha mejorado tanto?.....	18
2.3 Aceleración hardware para aprendizaje profundo.....	19
2.3.1 Hardware para uso personal.....	19
2.3.2 Hardware para centros de datos y uso profesional.....	20
2.4 Redes neuronales artificiales.....	21
2.4.1 Pesos y sesgo.....	21
2.4.2 Función de activación.....	21
2.4.3 Capas.....	23
2.5 Tipos de redes neuronales.....	24
2.5.1 Perceptrón.....	24
2.5.2 Perceptrón multicapa.....	24
2.5.3 Redes neuronales convolucionales.....	25
2.5.4 Red neuronal recurrente.....	25
2.5.5 Long Short Term Memory.....	26
2.5.6 Red neuronal modular.....	26
2.5.7 Red generativa adversativa.....	26
2.5.8 Modelo transformer.....	26
2.6 Entrenamiento de una red neuronal convolucional.....	27
2.6.1 Descenso de gradiente.....	27
2.6.2 Backpropagation.....	28
2.6.3 Data augmentation.....	28
2.6.4 Overfitting y underfitting.....	28
2.7 Optimización de un modelo para dispositivos móviles.....	30
2.8 Proyectos relacionados.....	31
CAPÍTULO 3: OBJETIVOS.....	33
3.1 Objetivos generales.....	34
3.2 Objetivos didácticos.....	34
3.3 Objetivos personales.....	35

CAPÍTULO 4: HIPÓTESIS DE TRABAJO.....	36
4.1 Herramientas software para el desarrollo de la red neuronal.....	37
4.1.1 Lenguaje de programación Python.....	37
4.1.2 PIP.....	37
4.1.3 PyTorch.....	38
4.1.4 Torchvision.....	39
4.1.5 CUDA.....	39
4.1.6 Otras librerías.....	39
4.1.7 Sublime Text 3.....	39
4.1.8 Command Prompt.....	39
4.2 Herramientas software para el desarrollo de la aplicación móvil.....	40
4.2.1 Android Studio.....	40
4.2.2 Lenguaje de programación JAVA.....	40
4.2.3 Camera2 y CameraX.....	40
4.2.4 Torchvision lite y Pytorch lite.....	41
4.2.5 Otros.....	41
4.3 Hardware.....	41
4.3.1 Especificaciones del equipo.....	42
4.3.2 Especificaciones del dispositivo móvil.....	42
CAPÍTULO 5: METODOLOGÍA Y RESULTADOS.....	43
5.1 Planificación.....	44
5.1.1 Desglose de tareas.....	44
5.1.2 Diagrama de Gantt.....	44
5.2 Creación del modelo y entrenamiento.....	45
5.2.1 Información del dataset.....	45
5.2.2 Primeras decisiones del modelo y entrenamiento.....	46
5.2.3 Primer modelo y análisis de las variables de entrenamiento.....	47
5.2.4 Segundo modelo y primer análisis de sus variables.....	54
5.2.5 Tercer modelo. Aumento de la complejidad.....	56
5.2.6 Cuarto modelo. Tamaño del primer kernel.....	58
5.2.7 Quinto modelo. Últimos ajustes.....	59
5.2.8 Cuantificación del modelo.....	61
5.3 Pruebas de rendimiento y comparación con otros modelos.....	63
5.3.1 Modelo 5. Rendimiento con y sin cuantificación.....	63
5.3.2 Comparativa con modelos pre-entrenados.....	65
5.4 Aplicación para dispositivos móviles Android.....	70
5.4.1 Análisis de requisitos.....	70
5.4.2 Desarrollo de la aplicación.....	73
5.4.3 Importar el modelo a la aplicación.....	73
5.5 Rendimiento en la aplicación.....	75
CAPÍTULO 6: CONCLUSIONES Y TRABAJOS FUTUROS.....	76

6.1 Conclusiones.....	77
6.2 Trabajos futuros.....	78
CAPÍTULO 7: BIBLIOGRAFÍA.....	80
ANEXO I. GLOSARIO.....	91
I.I Glosario general.....	92
1. ASIC.....	92
2. Convolución.....	92
3. CUDA.....	93
4. Dataset.....	93
5. Época.....	93
6. Exactitud.....	93
7. FPGA.....	93
8. Iteración.....	94
9. Logit.....	94
10. Lote.....	94
11. Modelo.....	94
12. Matriz de confusión.....	94
13. Optimizador.....	94
14. Pérdida.....	95
15. Precisión.....	95
16. Regularización de abandono.....	95
I.II. Glosario de Pytorch.....	95
1. Model.....	95
2. Capa “Conv2d”.....	95
3. Capa “Linear”.....	96
4. Max-pooling “MaxPool2d”.....	96
5. Avg-pooling “AvgPool2d”.....	96
6. Funcion de activación “LeakyReLU”.....	96
7. Funcion de activación “ReLU”.....	96
8. Normalización “BatchNorm2d”.....	97
9. Abandono “Dropout2D”.....	97
10. Función de pérdida “CrossEntropyLoss”.....	97
11. Algoritmo optimizador “Adam”.....	97
12. Panificador de la tasa de aprendizaje “ReduceLRonPlateau”.....	97
13. “torch.jit.script”.....	98
14. “optimize_for_mobile”.....	98
15. “_save_for_lite_interpreter”.....	98
ANEXO II. Interfaz de la aplicación Android.....	99

Índice de ilustraciones

Ilustración 1. Diagrama de IA, ML y DL.....	16
Ilustración 2. Ejemplo de una sección de una red neuronal.....	20
Ilustración 3. Funciones de activación [21].....	21
Ilustración 4. Ejemplo de perceptrón.....	23
Ilustración 5. Ejemplo de perceptrón multicapa.....	24
Ilustración 6. Representación gráfica del descenso de gradiente.....	26
Ilustración 7. Gráfico de pérdida de un modelo con overfitting [31].....	28
Ilustración 8. Logotipo de Python.....	36
Ilustración 9. Logotipo de PyTorch.....	37
Ilustración 10. Tabla de compatibilidad de PyTorch.....	37
Ilustración 11. Logotipo de Android Studio.....	39
Ilustración 12. Diagrama de Gantt.....	44
Ilustración 13. Gráfica de Pérdida y Exactitud. Modelo 1, entrenamiento 1.....	47
Ilustración 14. Gráficas de pérdida con distinto tamaño de lote.....	48
Ilustración 15. Tiempo de ejecución frente a Tamaño de lote.....	48
Ilustración 16. Memoria insuficiente para entrenar el modelo.....	49
Ilustración 17. Gráficas de entrenamiento con distinta tasa de aprendizaje.....	50
Ilustración 18. Comparación de paciencia = 5 y paciencia = 10.....	52
Ilustración 19. Gráficas del segundo modelo.....	54
Ilustración 20. Matriz de confusión del segundo modelo.....	55
Ilustración 21. Gráficas del tercer modelo.....	56
Ilustración 22. Matriz de confusión del tercer modelo.....	56
Ilustración 23. Gráficas del cuarto modelo.....	58
Ilustración 24. Matriz de confusión del cuarto modelo.....	58
Ilustración 25. Gráficas del quinto modelo.....	59
Ilustración 26. Matriz de confusión del cuarto modelo.....	60
Ilustración 27. Modelo 5 sin cuantificación.....	62
Ilustración 28. Modelo 5 con cuantificación consciente del entrenamiento.....	62
Ilustración 29. Modelo 5 con cuantificación estática posterior al entrenamiento.....	63
Ilustración 30. Exactitud en entrenamiento y exactitud en validación.....	63
Ilustración 31. Pérdida en entrenamiento y pérdida en validación.....	64
Ilustración 32. Tiempo medio por época.....	64
Ilustración 33. Gráficas de VGG16.....	65
Ilustración 34. Gráficas de EficientNet_v2_1.....	66
Ilustración 35. Gráficas de Mobilenet_v2.....	66
Ilustración 36. Gráficas de Modelo 5.....	67
Ilustración 37. Comparativa de pérdida en validación en los modelos pre-entrenados y el modelo 5.....	67
Ilustración 38. Comparativa de exactitud en los modelos pre-entrenados y el	

modelo 5.....	68
Ilustración 39. Comparativa de los tiempos de ejecución del entrenamiento en los modelos pre-entrenados y el modelo 5.....	68
Ilustración 40. Diagrama de casos de uso.....	70
Ilustración 41. Página principal de la aplicación.....	99
Ilustración 42. Información sobre el uso de la cámara.....	100
Ilustración 43. Elegir modelo.....	101
Ilustración 44. Solicitud de permiso.....	102
Ilustración 46. Información de la red neuronal.....	104
Ilustración 46. Ejemplos de clasificación.....	105



Índice de tablas

Tabla 1. Desglose de tareas.....	43
Tabla 2. Resumen del dataset.....	44
Tabla 3. Variables básicas del modelo.....	45
Tabla 4. Variables básicas de entrenamiento.....	45
Tabla 5. Variables del primer modelo.....	46
Tabla 6. Variables básicas de entrenamiento.....	47
Tabla 7. Tiempo de ejecución según el tamaño de lote.....	48
Tabla 8. Variables del segundo modelo.....	54
Tabla 9. Variables de entrenamiento.....	54
Tabla 10. Variables del tercer modelo.....	56
Tabla 11. Variables del cuarto modelo.....	57
Tabla 12. Variables del quinto modelo.....	59
Tabla 13. Comparativa de tiempos de entrenar las redes con cuantificación estática posterior al entrenamiento (CPU).....	69
Tabla 14. Actor - Usuario.....	69
Tabla 15. Actor - Desarrollador.....	70
Tabla 16. Caso de uso 1.....	70
Tabla 17. Caso de uso 2.....	71
Tabla 18. Requisito funcional 1.....	71
Tabla 19. Requisito funcional 2.....	71
Tabla 20. Requisito funcional 3.....	71
Tabla 21. Requisito no funcional 1.....	71
Tabla 22. Requisito funcional 3.....	71
Tabla 23. Requisito funcional 3.....	72
Tabla 24. Tiempo medio para procesar una imagen en la aplicación.....	74

CAPÍTULO 1: INTRODUCCIÓN



1.1 Introducción a la visión por computador

La visión por computador o visión artificial es un campo de investigación centrado en dotar a los computadores la capacidad de comprender e interpretar el contenido de imágenes y videos. Las primeras investigaciones en este campo se remontan a los años sesenta, cuando comenzó la comercialización de los primeros ordenadores de uso personal. Durante estos años se desarrollaron técnicas de procesamiento de imagen como el filtro Sobel para detección de bordes en 1968 [1].

Tras décadas de investigación, los resultados obtenidos fueron muy limitados y solamente eran aplicables a entornos muy controlados, no a casos reales [2].

1.1.1 Clasificación de imágenes mediante inteligencia artificial

La llegada del aprendizaje automático supuso un gran avance en la visión por computador. Finalmente, el uso de redes neuronales convolucionales supuso una revolución en este campo, puesto que esto produjo grandes mejoras en la detección de patrones como bordes, texturas y formas.

1.2 Inteligencia artificial en la medicina

Recientemente el uso de la inteligencia artificial está teniendo relevancia en múltiples campos de la medicina. La inteligencia artificial puede convertirse en una herramienta de apoyo capaz de realizar automáticamente tareas tales como:

- Diagnóstico de enfermedades: Analizar muestras de tejidos para identificar signos de enfermedades o análisis de imágenes.

- Gestión y análisis de datos: Analizar grandes volúmenes de datos para identificar patrones o anticipar complicaciones de enfermedades.
- Monitoreo de Pacientes: Utilizando dispositivos portátiles para monitorear en tiempo real signos vitales, permitiendo una atención continua y proactiva.
- Gestión de recursos: Ayudar a optimizar la gestión de recursos como material y personal en hospitales [3] [4].

1.2.1 Inteligencia artificial para las imágenes médicas

El diagnóstico por imágenes permite a los médicos buscar afecciones en el interior del cuerpo de los pacientes. Algunas de estas imágenes son: rayos X, tomografías computarizadas (TC), estudios de medicina nuclear, imágenes por resonancia magnética, ecografías [5].

Históricamente, el análisis de imágenes médicas lo han realizado médicos de forma manual, pero desde hace años los médicos también han contado con el apoyo de tecnologías como los algoritmos de detección de bordes y más recientemente la segmentación que sirve para dividir la imagen en partes significativas [6].

Recientemente se está empleando inteligencia artificial en este campo, como es el uso de una red neuronal convolucional para el prediagnóstico de cáncer de páncreas a partir de TCs tridimensionales [7]. Este campo se conoce como diagnósticos asistidos por computador o CAD (computer aided diagnosis), esto no significa que la máquina sustituye al médico, sino que debe complementar a los médicos para ayudar a estos en la detección temprana de enfermedades. También existe el concepto de diagnóstico automatizado por ordenador, el cual está basado únicamente en algoritmos informáticos con muy poca o sin la intervención humana en el diagnóstico [8].

Puesto que estas tecnologías apenas se están empezando a poner a prueba recientemente, todavía tienen una disponibilidad muy limitada en hospitales y centros de salud.

1.3 Inteligencia artificial en dispositivos móviles

A pesar de estar limitados por las capacidades de procesamiento de estos dispositivos, esto no ha impedido que surgieran distintas utilidades que hacen uso de la inteligencia artificial, como el reconocimiento facial o asistentes virtuales [9]. Sin embargo, la realización de tareas más complejas requiere el uso de modelos₍₁₁₎ que necesitan unas capacidades de procesamiento demasiado altas para estos dispositivos, lo cual ha llevado al desarrollo de técnicas para optimizar el rendimiento. También existe la opción de utilizar el dispositivo móvil conectado a servidor que realice el procesamiento y envíe los resultados de nuevo al dispositivo móvil. Además, recientemente algunos teléfonos móviles cuentan con un CPU que integra un procesador (NPU) específicamente diseñado para acelerar el procesamiento de redes neuronales .



CAPÍTULO 2: ESTADO DE LA CUESTIÓN



2.1 Inteligencia artificial

El término inteligencia artificial (IA o AI en inglés) se utiliza para clasificar las máquinas que imitan la inteligencia y funciones cognitivas humanas como la resolución de problemas y el aprendizaje. La IA utiliza predicciones y automatización para resolver tareas complejas que los humanos también son capaces de realizar, como el reconocimiento facial, el reconocimiento de voz, la toma de decisiones o realizar traducciones [10].

Recientemente han habido grandes avances en inteligencia artificial y por ello se ha convertido en un campo muy relevante actualmente. La idea de inteligencia artificial es anterior a los ordenadores, el artículo *Computing machinery and intelligence* de 1950 de Alan Turing introduce la cuestión ¿Pueden pensar las máquinas? y el test de Turing, una prueba que consiste en evaluar si se están obteniendo respuestas de una persona o una máquina que está imitando el comportamiento humano.

2.1.1 Categorías de la IA

Según sus capacidades, la inteligencia artificial se divide en los siguientes tres grupos:

Inteligencia artificial débil

Este tipo de inteligencia artificial se centra en realizar una tarea específica y engloba todos los tipos de IA actuales: procesamiento del lenguaje natural, visión por computadora, asistentes virtuales, etc. Está por debajo de la inteligencia y capacidades humanas [10].

Inteligencia artificial general

Es una IA teórica que también se puede llamar IA fuerte. No hay ejemplos claros de este tipo de IA, puesto que nunca se ha desarrollado. Tendría una inteligencia y capacidades similar a la humana [11].

Super inteligencia artificial

También es una IA teórica e IA fuerte, pero esta superaría la inteligencia y capacidad de un humano [11]. Recientemente hay personas que afirman que la super inteligencia artificial será una realidad en los próximos años [12].

2.1.2 Subconjuntos de la IA

“Inteligencia artificial” es un término muy general, en este trabajo nos vamos a centrar en el subconjunto de la inteligencia artificial llamado aprendizaje automático y el subconjunto de éste, el aprendizaje profundo. La relación entre estos se muestra en la Ilustración 1.

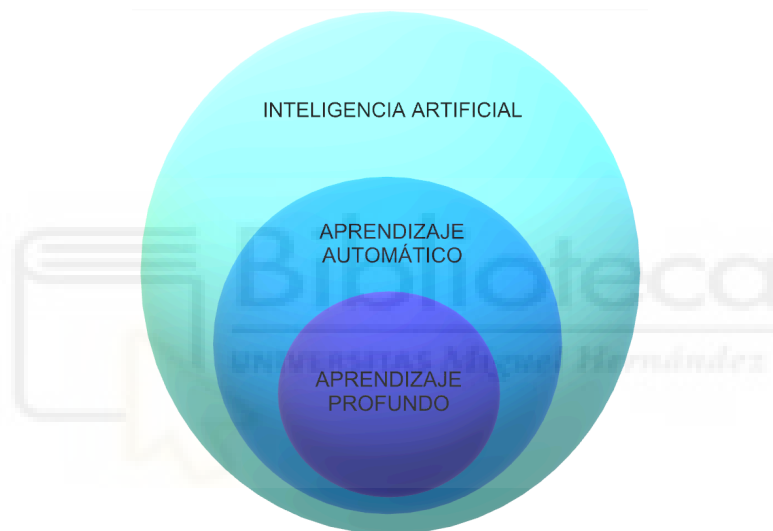


Ilustración 1. Diagrama de IA, ML y DL

Aprendizaje automático

El aprendizaje automático o machine learning (ML) es capaz de aprender e identificar patrones en un conjunto de datos₍₄₎, pero depende de la intervención humana para que el sistema aprenda [13].

Aprendizaje profundo

El aprendizaje profundo o deep learning (DL) utiliza tres o más capas de redes neuronales y es capaz de realizar una toma de decisiones compleja. Esta técnica de inteligencia artificial se aplica a campos como la visión por computador y el procesamiento de lenguaje natural [14].

2.1.3 Aprendizaje supervisado y no supervisado

Dos tipos diferentes de modelos de aprendizaje automático son:

- Aprendizaje supervisado: Los datos de entrenamiento están etiquetados, por tanto el modelo tiene que aprender a predecir la etiqueta esperada en datos no etiquetados tras haber sido entrenado con datos etiquetados. Por ejemplo, los modelos de clasificación de imágenes.
- Aprendizaje no supervisado: Los datos de entrenamiento no están etiquetados, por tanto el modelo tiene que encontrar patrones subyacentes en el conjunto de datos. Por ejemplo, sugerir productos relacionados a otros basándose en patrones de compra [15].

2.2 ¿Por qué ha mejorado tanto?

En los últimos años han habido varios motivos por los cuales la inteligencia artificial ha avanzado tan significativamente recientemente. Los estos son algunos de los más relevantes:

- Mayor potencia de procesamiento: Un hardware con mayor rendimiento permite ejecutar modelos más complejos y reducir el tiempo de ejecución y entrenamiento. El uso de GPUs en lugar de CPUs ha mejorado drásticamente el rendimiento en los últimos años y recientemente se está extendiendo el uso de procesadores especializados en redes neuronales.
- Mejoras en los modelos: Tanto el desarrollo de modelos más complejos (aprendizaje profundo) como el desarrollo de distintos tipos de redes neuronales con distintos propósitos (las redes neuronales convolucionales para imágenes, transformers para procesamiento de lenguaje, etc).
- Mejores herramientas: Librerías como PyTorch, TensorFlow y Keras han facilitado el desarrollo de modelos.
- Mayor cantidad de datos: Cada vez hay más datasets₍₄₎ de distintos tipos disponibles, además de grandes volúmenes de datos (Big Data).

2.3 Aceleración hardware para aprendizaje profundo

La aceleración por hardware ayuda a mejorar la eficiencia y la velocidad de los modelos de aprendizaje automático al ejecutarlos en hardware específico en lugar de un procesador de uso general. Como ya se ha mencionado anteriormente, este es uno de los motivos por los cuales la IA ha mejorado tanto en los últimos años.

2.3.1 Hardware para uso personal

A continuación se muestran los principales aceleradores hardware disponibles para uso personal. Es notorio resaltar que muchas de estas tecnologías son muy recientes y por tanto tienen un soporte sumamente limitado.

GPU

Se utilizan los núcleos del procesador gráfico a través de una API.

CUDA₍₃₎ es la más relevante. Intel ofrece oneAPI, y AMD ROCm como alternativa desde 2016. Pero el soporte de CUDA está mucho más extendido [16].

GPU con núcleos para IA

Algunas tarjetas gráficas cuentan con unos núcleos específicos para aprendizaje automático; en el caso de los Tensor cores, estos son capaces de multiplicar dos matrices (FP16) de tamaño 4×4 y sumar la matriz (FP32) al acumulador por cada GPU clock.

Nvidia Tensor cores: Desde la microarquitectura Turing de 2018.

Intel XMX: Desde su primera microarquitectura de GPUs: Alchemist de 2022.

AMD AI Accelerators: Desde la microarquitectura RDNA 3 de 2022.

NPU

Una unidad de procesamiento neural (Neural Processing Unit) está construida para para acelerar la inferencia de modelos IA mientras mantiene un bajo nivel de consumo. El

diseño de las NPU está estrechamente relacionado con la dirección de la industria de la IA [17] [18]. Distintos dispositivos disponen de distintas NPU, estas son las más relevantes actualmente:

- En ordenadores portátiles
 - AMD Ryzen™ AI (x86)
 - Intel® AI Boost (x86)
 - Qualcomm® Hexagon (ARM)
- En dispositivos móviles:
 - MediaTek NPU 790
 - Apple Neural Engine
 - Qualcomm Hexagon
 - Google Tensor
 - HUAWEI Da Vinci Architecture

2.3.2 Hardware para centros de datos y uso profesional

Los principales aceleradores hardware para centros de datos y uso profesional son:

GPU

Las tarjetas gráficas Nvidia de uso profesional son las más utilizadas, del mismo modo que ocurre en el hardware de uso personal. Cuentan con un mayor número de Tensor Cores (y desde una arquitectura anterior: Volta de 2017) que las tarjetas gráficas de uso personal y también cuentan con núcleos CUDA.

TPU

Google Cloud TPUs son propietarias de google . Son ASICs₍₁₎ diseñados por Google con integración nativa con TensorFlow [18].

Otros

- ASICs: Existen otros ASICs, como Wafer Scale Engine (WSE).
- FPGAs₍₇₎ : Como la tarjeta AMD Alveo U250 con el motor de procesamiento Xilinx xDNN.

2.4 Redes neuronales artificiales

La inteligencia artificial sienta sus bases en el trabajo de 1943 de Warren y Pitts, que teoriza sobre la lógica de un sistema nervioso a partir de la neurofisiología. Esto inspiró el perceptrón, el modelo más simple de una neurona. Por tanto, las redes neuronales artificiales tienen su origen en la búsqueda de imitar un sistema nervioso biológico. Aunque en realidad el funcionamiento de una red neuronal artificial dista mucho de una biológica.

2.4.1 Pesos y sesgo

Son los valores numéricos asociados a las conexiones entre neuronas (nodos). Los valores de los pesos cambian al aprender durante el entrenamiento, pero se mantienen iguales durante la ejecución. Por tanto, dos modelos iguales tendrán pesos distintos si se entrenan con datasets diferentes. El sesgo es un valor que se añade antes de la función de activación y ayuda a cambiar el valor de esta. En la Ilustración 2 se muestra una representación de una sección de una red neuronal.

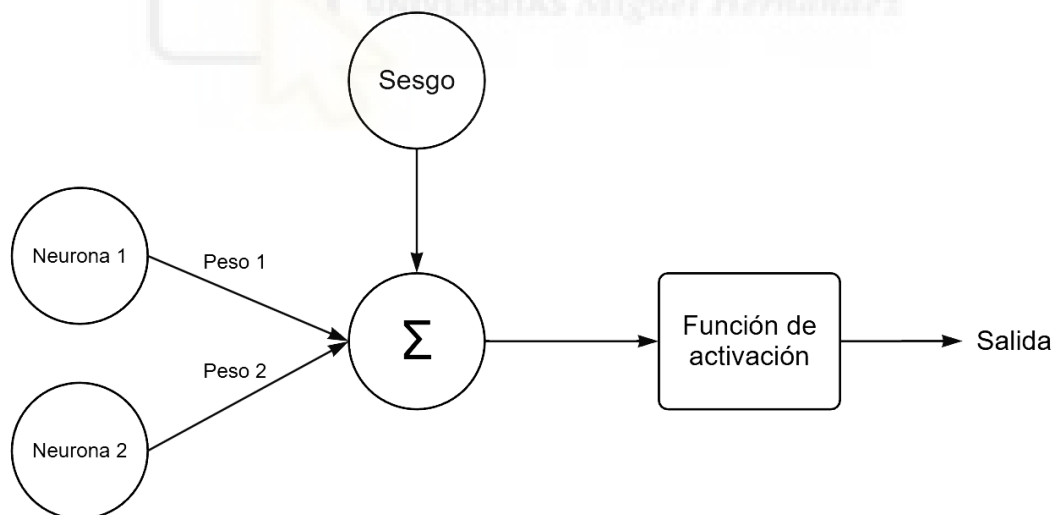


Ilustración 2. Ejemplo de una sección de una red neuronal

2.4.2 Función de activación

La función de activación es una función matemática que determina la salida de un nodo a partir de una entrada (o conjunto de entradas). La función de activación es la que

decide si una neurona debe activarse o no. Esto se obtiene calculando la suma ponderada y agregándole el sesgo. La función de activación introduce no linealidad a la salida de una neurona. La función de activación hace posible el back-tracking [19].

Funciones de activación

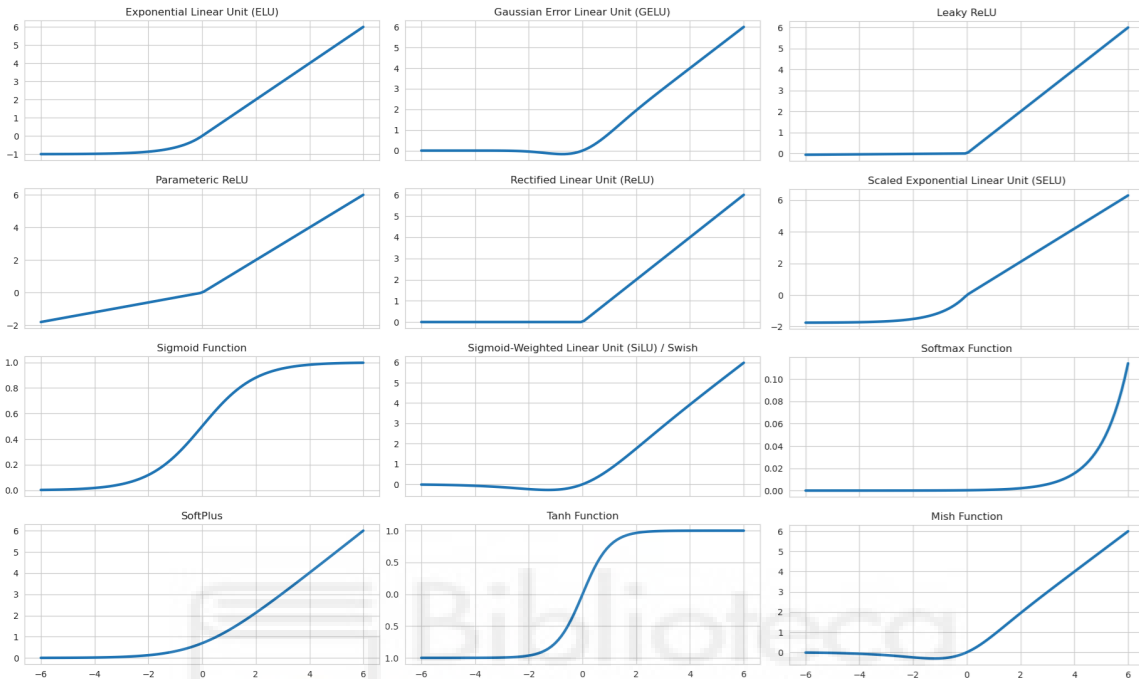


Ilustración 3. Funciones de activación [21].

La Ilustración 3 muestra una lista de funciones de activación no lineales. Las más relevantes se muestran a continuación:

Sigmoide

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

Esta función de activación se muestra por razones históricas y nunca se utiliza en modelos reales porque no está centrada a cero y es susceptible de causar el problema de desvanecimiento de gradiente [20].

Tahn

Tampoco se utiliza porque es similar a la sigmoide, pero centrada a cero [20].

Softmax

Es una forma más generalizada de la sigmoide y se usa en la capa de salida en problemas de clasificación [20].

ReLU

$$f(x) = \max(0, x)$$

Es una función de activación muy utilizada, especialmente en redes convolucionales. Tiene un buen rendimiento, pero sus desventajas son no estar centrada a cero y que su salida es cero para todas las entradas negativas (“dying ReLU”) [20].

Leaky ReLU

$$f(x) = \max(\alpha x, x)$$

Esta función resuelve parcialmente el problema de “dying ReLU”. Pero cuando $\alpha = 1$ la función es $f(x) = x$ y no sería útil [20].

Parametric ReLU

Esta función se obtiene a partir de Leaky ReLU al configurar α como un hiperparámetro para cada neurona [20].

Swish

$$f(x) = x * \text{sigmoid}(x)$$

Tiene un rendimiento un poco mejor que ReLU, puesto que converge un poco mejor durante el entrenamiento. Pero tiene un coste computacional elevado [20].

2.4.3 Capas

- Capa de entrada: Representa las dimensiones de los datos de entrada.
- Capas ocultas: Es la capa o capas que realiza todos los cálculos de la red neuronal. Existen distintos tipos de capas ocultas con diferentes funciones.
- Capa de salida: Representa los datos de salida de la red neuronal.

El número de capas de una red neuronal suele estar asociado con las capacidades de esta. Los modelos más complejos y con altas capacidades suelen tener un gran número de capas. Sin embargo, si para un determinado problema se utilizan más capas de las necesarias, el coste computacional aumenta aunque el rendimiento de la red no mejore. Existen distintos tipos de capas, como convolucionales, fully connected (o linear layers), recurrentes.

También existen capas que no contienen neuronas, por tanto tampoco aprenden, puesto que no tienen pesos y sesgos. Estas “capas” realizan operaciones concretas, como pooling, normalización, dropout.

2.5 Tipos de redes neuronales

A continuación se muestra un breve resumen de algunas de las redes neuronales más relevantes.

2.5.1 Perceptrón

El perceptrón, también conocido como unidad lógica de umbral, es la primera red neuronal artificial. Es una red muy simple que solamente tiene una capa de entrada y otra de salida. Puede implementar puertas lógicas AND, OR y NAND. La Ilustración 4 muestra un ejemplo de perceptrón [22] [23].

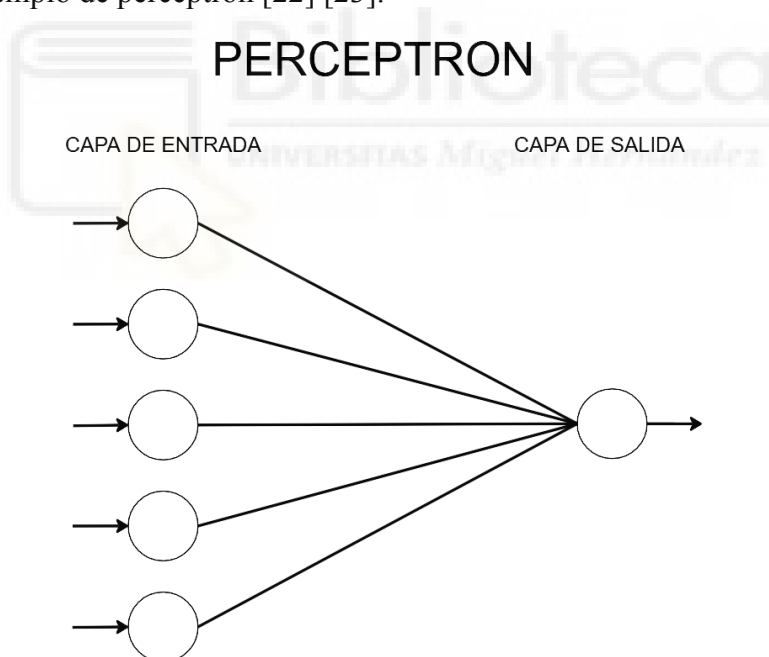


Ilustración 4. Ejemplo de perceptrón.

2.5.2 Perceptrón multicapa

El perceptrón multicapa consta de una o múltiples capas ocultas. Cada neurona está conectada a todas las neuronas de la capa anterior y de la capa siguiente, como se

muestra en la Ilustración 5. Es capaz de resolver tareas mucho más complejas que el perceptrón, pero también tiene un diseño más complejo [23].

PERCEPTRON MULTICAPA

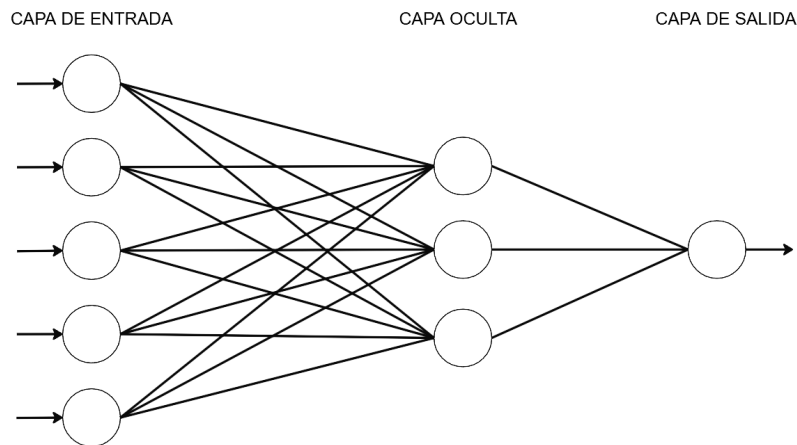


Ilustración 5. Ejemplo de perceptrón multicapa.

2.5.3 Redes neuronales convolucionales

Las redes neuronales convolucionales (CNN) contienen al menos una capa convolucional. Las capas convolucionales procesan la información de entrada por lotes₍₁₀₎, es decir, realizan una convolución₍₂₎. Las capas convolucionales funcionan bien para extraer características de imágenes, pero también se pueden utilizar con datos de entradas de una o más de 2 dimensiones [23].

2.5.4 Red neuronal recurrente

Estas redes neuronales (RNN) tienen capas recurrentes que almacenan la información que tenían en el paso anterior. Estas redes neuronales funcionan bien para procesar datos secuenciales, como traducciones y sugerencias de texto automáticas [23].

2.5.5 Long Short Term Memory

Las LSTM son un tipo de red neuronal recurrente, pero estas pueden almacenar información en una celda de memoria durante periodos de tiempo más largos que las RNN [23].

2.5.6 Red neuronal modular

Estas redes neuronales tienen varias redes neuronales diferentes que funcionan independientemente unas de otras.

2.5.7 Red generativa adversativa

Las GAN o redes generativas antagónicas están compuestas por dos redes neuronales que compiten entre sí continuamente. Una red es la generadora y la otra es la discriminadora. Por ejemplo, el generador crea una imagen y el discriminador intenta predecir si es una imagen real o creada por el generador. Si el discriminador es capaz de diferenciar entre las imágenes reales y las creadas, este proporciona una orientación al generador para que mejore sus salidas.

Se utilizan para generar imágenes, música, procesamiento de imágenes, etc [24].

2.5.8 Modelo transformer

El modelo transformer se ha convertido en el más importante en los últimos años. Este modelo usa el aprendizaje no supervisado y es capaz de aprender contexto, por tanto no se entrena con datasets etiquetados (entrenamiento supervisado).

Los transformers utilizan codificadores posicionales y unidades de atención para mapear relaciones entre datos. Las consultas de atención suelen ejecutarse en paralelo con atención de múltiples encabezados, permitiendo al modelo identificar patrones.

Los transformers están reemplazando en algunos casos a las redes neuronales convolucionales y recurrentes. Aunque son muy conocidos por sus usos en modelos de lenguaje grande, como GPTs. También se está investigando su uso para diseñar fármacos, prevenir fraudes, hacer recomendaciones online, etc [25] [26].

2.6 Entrenamiento de una red neuronal convolucional

En este apartado se muestran los conceptos más importantes en el proceso de aprendizaje de una red neuronal, específicamente para CNNs, aunque muchos de estos conceptos son iguales a otras redes neuronales.

2.6.1 Descenso de gradiente

El descenso de gradiente o gradient descent es un enfoque de optimización para entrenar modelos de aprendizaje automático. Los modelos van ajustando sus pesos y sesgos durante el entrenamiento, y el propósito del descenso de gradiente es minimizar la función de pérdida⁽¹⁴⁾, mejorando así la exactitud⁽⁶⁾ del modelo. El modelo continuará cambiando sus parámetros hasta que la función de pérdida esté cerca o sea igual a cero como se muestra en la Ilustración 6 [27].

En aprendizaje automático se aplica el descenso de gradiente utilizando un algoritmo optimizador⁽¹³⁾. En redes convolucionales, el algoritmo Adam es el más utilizado.

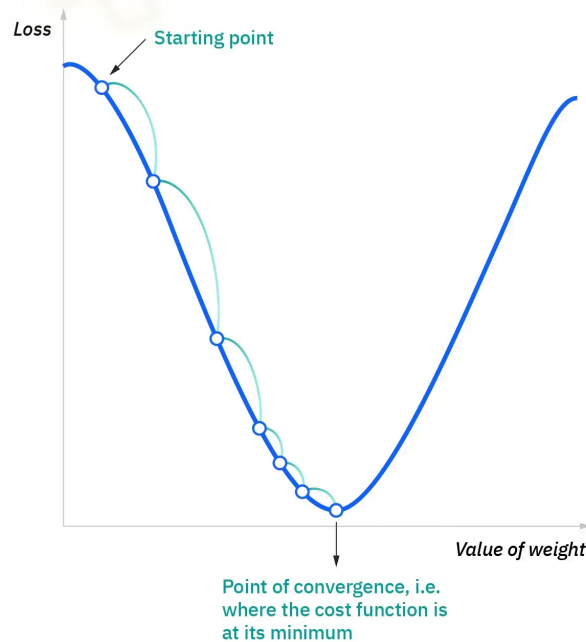


Ilustración 6. Representación gráfica del descenso de gradiente.

2.6.2 Backpropagation

La propagación hacia atrás o retropropagación es un algoritmo fundamental en las redes neuronales. Su funcionamiento consiste en realizar una propagación hacia adelante y calcular la pérdida entre la predicción y el valor de la etiqueta, seguido de una propagación hacia atrás para reducir la pérdida ajustando los pesos y sesgos de la red neuronal [28].

2.6.3 Data augmentation

El aumento de datos es un proceso para aumentar artificialmente el número de datos de entrenamiento de un modelo. Esto puede tener beneficios como mejoras en la exactitud del modelo, menor dependencia de los datos y mitigar el overfitting.

En el caso de CNNs para imágenes, el aumento de datos se puede realizar mediante procesamientos de imagen simples como cambios en el brillo, el contraste o la saturación y voltear o rotar la imagen [29].

2.6.4 Overfitting y underfitting

Un modelo está bien ajustado cuando no tiene overfitting ni underfitting. Por ejemplo, un modelo entrenado para detectar caras humanas está bien ajustado cuando detecta sin problemas las caras de los datos de entrenamiento y también las detecta bien en los datos de validación.

Un modelo con underfitting no aprende suficientemente el problema y tiene un rendimiento mejorable con los datos de entrenamiento y peor con los datos de validación [30]. Siguiendo el ejemplo anterior, un modelo entrenado para detectar caras humanas con underfitting no solo tendría problemas para detectar caras, también identificaría como una cara cosas que tienen muy poca (o ninguna) similitud con una cara humana.

Un modelo con overfitting aprende demasiado bien el conjunto de datos de entrenamiento y tiene un muy buen rendimiento en este, pero funciona peor con los datos de validación [30]. Finalizando con el ejemplo anterior, un modelo entrenado

para detectar caras humanas con overfitting solamente detectaría bien las caras de las personas que forman parte de los datos de entrenamiento o se parecen a estas.

La Ilustración 7 muestra cómo varía la función de pérdida en un modelo cuando aumenta su complejidad. Con una baja complejidad el modelo sufre underfitting. Al aumentar la complejidad pasa a tener un buen ajuste (optimum). Pero al seguir aumentando la complejidad, la función de pérdida en los datos de validación (generalization loss) aumenta (overfitting).

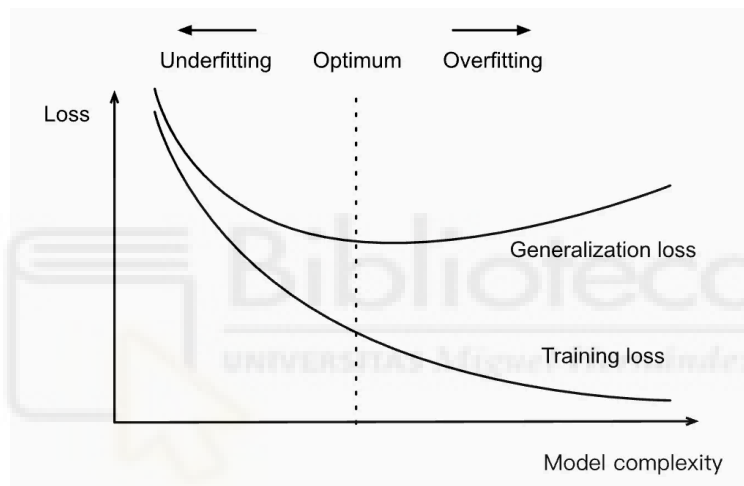


Ilustración 7. Gráfico de pérdida de un modelo con overfitting [31]

El underfitting se puede reducir aumentando la capacidad del modelo, es decir, aumentando el número de capas y/o el número de neuronas de estas [30].

Para evitar el overfitting se utilizan métodos de regularización. El más simple y quizás más utilizado es:

- Regularización de peso (weight decay): penaliza el modelo durante el entrenamiento en función de la magnitud de los pesos [32].

Otros métodos de regulación:

- Regularización de actividades (activity regularization): penaliza el modelo durante el entrenamiento en función de la magnitud de las activaciones [32].
- Restricción de peso (weight constraint): limitar la magnitud de los pesos dentro de un rango o por debajo de un límite [32].
- Abandono (dropout): elimina aleatoriamente datos de entrada durante el entrenamiento [32].
- Ruido (noise): agregar ruido a las entradas durante el entrenamiento.
- Detención anticipada (early stopping): supervisar el rendimiento del modelo utilizando un conjunto de validación y detener el entrenamiento cuando el rendimiento en este desciende [32].

2.7 Optimización de un modelo para dispositivos móviles

Al importar un modelo a Android es necesario tener en cuenta que, normalmente, el modelo se crea en una arquitectura x86 y se lleva a una arquitectura ARM. Este problema se soluciona utilizando librerías como XNNPACK [33], QNNPACK [34] o FBGEMM [35]. Aunque en algunos casos PyTorch pueda encargarse de esto de forma casi automática (por ejemplo, al usar “optimize_for_mobile” internamente se está utilizando XNNPACK) esto puede causar limitaciones, como no poder entrenar un modelo con la GPU [36].

Otro aspecto importante es el rendimiento y memoria limitada de los dispositivos móviles, aquí es donde entra la cuantificación. Cuantificar un modelo consiste en reducir levemente la exactitud de este al sustituir variables float de 32 bits por int de 8 bits. Esto puede reducir el tamaño del modelo hasta una cuarta parte del tamaño original y puede aumentar la velocidad de inferencia de dos a cuatro veces.

La cuantificación es un paso opcional para llevar un modelo a dispositivos móviles, sin embargo el consumo y el tiempo de ejecución se verían perjudicados, y estos son factores cruciales en un dispositivo portátil. Además, es posible que los modelos que

requieren mucha memoria no se puedan ejecutar en algunos dispositivos sin cuantificación.

Tipos de cuantificación:

- Cuantización dinámica posterior al entrenamiento. Convierte todos los pesos en un modelo de números flotantes de 32 bits a enteros de 8 bits pero no convierte las activaciones a int8 hasta justo antes de realizar el cálculo de las activaciones. Actualmente solo se puede aplicar a capas nn.Linear y nn.LSTM [37].
- Cuantización estática posterior al entrenamiento. Este método convierte de antemano los pesos y activaciones a enteros de 8 bits, por lo que no habrá una conversión en las activaciones durante la inferencia, como ocurre con la cuantificación dinámica. Pero este método puede degradar más la exactitud que la cuantificación dinámica [37].
- Entrenamiento consciente de la cuantificación. El entrenamiento consciente de la cuantificación emula una cuantificación en todos los pesos y activaciones durante el proceso de entrenamiento del modelo y da como resultado una mayor exactitud de inferencia que los métodos de cuantificación posteriores al entrenamiento. Normalmente se utiliza en modelos CNN [37] [38].

También es posible usar un modelo pre-entrenado y ligero que ya está cuantificado como MobileNet v2, ResNet 18, ResNet 50, Inception v3, GoogleNet, entre otros [37].

2.8 Proyectos relacionados

Anteriormente a este Trabajo de Fin de Grado ha habido otro TFG en el que se desarrolla una red neuronal convolucional, inspirada en VGG16, con PyTorch para clasificación de imágenes médicas. En dicho TFG también se pone a prueba el modelo desarrollado con otros modelos pre-entrenados. Sin embargo, el objetivo en ese caso era la clasificación de tumores cerebrales y la utilización de la red neuronal desarrollada se proporcionaba a través de un servicio web [39]. Sin embargo en este TFG el desarrollo de la red neuronal y pruebas con otras redes neuronales es sólo una parte de este trabajo,

además la red neuronal en este proyecto se ha creado sin seguir ninguna otra red como referencia.

Otro TFG crea una red neuronal desarrollada en TensorFlow y Keras para la predicción de infartos cerebrovasculares a partir de un dataset con información de texto como edad, género, otras enfermedades, etc. Los resultados se visualizan en una aplicación multiplataforma a través de una API que hace de intermediario [40]. Dicho trabajo dista mucho de este, puesto que en este la red neuronal se ejecuta directamente en una aplicación móvil, además este está desarrollado con PyTorch.

En el último se ejecutan las redes neuronales pre-entrenadas VGG16, ResNet50, MobileNet y EfficientNet en móviles Android. Está desarrollado con TensorFlow y Keras, haciendo uso de la librería TensorFlow Lite para realizar la optimización de los modelos. En dicho trabajo se realiza un exhaustivo análisis de cómo afecta en los modelos anteriores el estar o no optimizados para dispositivos móviles [41]. Aunque dicho trabajo tiene muchos puntos en común con este, en este trabajo se aporta el hecho de crear una red neuronal para resolver un problema concreto, optimizar la red neuronal desarrollada para su uso en dispositivos móviles y finalmente compararla con otros modelos con y sin optimización.

CAPÍTULO 3:

OBJETIVOS



En este capítulo se exponen los distintos objetivos que se pretenden alcanzar con la realización de este proyecto. Estos objetivos están divididos en tres grupos:

3.1 Objetivos generales

Estos son los objetivos que pretendo llevar a cabo en este proyecto:

- Desarrollar una red neuronal convolucional para la clasificación de imágenes de melanomas benignos y malignos.
- Optimizar y adaptar la red neuronal para ejecutarla en dispositivos móviles.
- Desarrollar una aplicación para teléfonos móviles Android que ejecute la red neuronal utilizando la cámara del dispositivo móvil como fuente de la imagen a clasificar.
- Comparar la mejora en el rendimiento (en coste computacional) de la red neuronal sin cuantificar y cuantificada. Así como analizar la reducción en la exactitud provocada por la cuantificación (en el caso de que se produzca).
- Comparar el rendimiento y exactitud de la red neuronal creada con otras redes pre-entrenadas cuantificadas (MobileNet) y no cuantificadas (EficientNet).

3.2 Objetivos didácticos

Esto es lo se espera aprender durante la realización de este trabajo:

- Programar en Python.
- Crear una red neuronal convolucional con PyTorch, sentando las bases necesarias para ser capaz de desarrollar otros tipos de redes neuronales.
- Analizar las métricas relacionadas con la exactitud de la red y ser capaz de mejorar la red neuronal a partir de estas.
- Optimizar y ejecutar un modelo de red neuronal de PyTorch en un dispositivo Android.

3.3 Objetivos personales

Uno de mis objetivos en este trabajo es aumentar mis conocimientos teóricos y prácticos relacionados con la inteligencia artificial, especialmente el aprendizaje automático y aprendizaje profundo, puesto que desconozco mucho de este campo de la informática y recientemente está teniendo mucha relevancia.

Hace ya una década desde que comencé a utilizar redes neuronales para el procesamiento de imágenes, empezando por filtros para escalar imágenes [42] [43] [44] hasta interpolación de fotogramas en tiempo real [45] y escalado de imagen en videojuegos utilizando información temporal [46]. Pero mis conocimientos en este campo están limitados a los de un usuario, por ese motivo tengo especial interés en aprender más en profundidad sobre este campo y ser capaz de desarrollar una red neuronal.

Por último, también me interesa aprender a programar en Python y utilizar PyTorch, debido a que Python es uno de los lenguajes de programación más utilizados y PyTorch es una de las librerías más relevantes en aprendizaje automático.

CAPÍTULO 4:

HIPÓTESIS DE

TRABAJO



A continuación se muestran las herramientas que se han utilizado para el desarrollo de este proyecto, esto incluye programas, librerías, lenguajes de programación, etc. Estas están divididas entre las herramientas para el desarrollo de la red neuronal y las herramientas empleadas para el desarrollo de la aplicación móvil.

4.1 Herramientas software para el desarrollo de la red neuronal

4.1.1 Lenguaje de programación Python



Python es uno de los lenguajes de programación más utilizados en la actualidad. Es un lenguaje fácil de aprender tanto para programadores novatos como programadores experimentados. Está escrito en lenguaje C, y su sintaxis es parecida a este lenguaje aunque tiene múltiples diferencias [47].

Para este trabajo era necesario utilizar un lenguaje de programación que disponga de herramientas para facilitar la implementación de un modelo de aprendizaje automático, y Python es uno de los lenguajes más utilizados para aprendizaje automático.

Versión utilizada: 3.10.11.

4.1.2 PIP

Pip es un instalador de paquetes por línea de comandos para Python. Mediante este se han instalado PyTorch, Torchvision, etc [48].

Versión utilizada: 24.0.

4.1.3 PyTorch



Ilustración 9. Logotipo de PyTorch

PyTorch es una biblioteca de Python (también dispone de una API para Java y C++) de código abierto para el aprendizaje automático y utilizada ampliamente para desarrollar y entrenar modelos de aprendizaje automático y profundo. PyTorch soporta CUDA, ROCm (solamente en Linux) y no tiene soporte para oneAPI [49].

PyTorch Build	Stable (2.2.2)	Preview (Nightly)		
Your OS	Linux	Mac	Windows	
Package	Conda	Pip	LibTorch	Source
Language	Python		C++ / Java	
Compute Platform	CUDA 11.8	CUDA 12.1	ROCm 5.7	CPU
Run this Command:	<pre>pip3 install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu118</pre>			

Ilustración 10. Tabla de compatibilidad de PyTorch.

Otra posibilidad es utilizar la biblioteca TensorFlow para aprendizaje automático, junto con Keras para redes neuronales. Pero finalmente se ha decidido utilizar PyTorch porque está más centralizado y porque permite la creación de prototipos más rápido que TensorFlow [50].

Versión utilizada: 2.0.1.

4.1.4 Torchvision

Torchvision es una librería que forma parte de PyTorch. Incluye datasets y modelos pre-entrenados. Esta librería se ha utilizado para cargar fácilmente modelos pre-entrenados.

Versión utilizada: 0.15.2.

4.1.5 CUDA

Utilizado para aumentar la velocidad de inferencia en el entrenamiento y ejecución de los modelos [51].

Versión utilizada: 11.8.

4.1.6 Otras librerías

- Matplotlib: Es una biblioteca de generación de gráficos en dos dimensiones [52]. Utilizada para crear gráficas de pérdida, exactitud y matriz de confusión₍₁₂₎.
- NumPy: Es una biblioteca para vectores y matrices [53].
- Pillow: Biblioteca para añadir capacidades de procesamiento de imágenes a Python [54]. Se ha utilizado para trabajar con las imágenes del dataset.

4.1.7 Sublime Text 3

Editor de texto para distintos lenguajes de programación. Utilizado para crear y editar el archivo .py con todo el código del modelo y su entrenamiento [55]. Se ha elegido este editor debido a su facilidad de uso y porque no eran necesarias funcionalidades de entornos de desarrollo más complejos.

Versión utilizada: Build 4169.

4.1.8 Command Prompt

Consola de comandos de Windows. Se ha utilizado para instalar paquetes y ejecutar código en Python. Se ha elegido esta herramienta porque ya viene en la máquina y no era necesario el uso de una herramienta más completa.

4.2 Herramientas software para el desarrollo de la aplicación móvil

4.2.1 Android Studio



Ilustración 11. Logotipo de Android Studio

Es el entorno de desarrollo integrado (IDE) oficial para el desarrollo de aplicaciones de Android. En este se ha desarrollado la aplicación [56]. Se ha elegido este entorno de desarrollo porque es el utilizado en la asignatura Desarrollo de aplicaciones para dispositivos móviles de la titulación, y además es el más relevante en el desarrollo de aplicaciones Android.

Versión utilizada: Android Studio Iguana | 2023.2.1 Patch 1, Build #AI-232.10300.40.2321.11567975.

4.2.2 Lenguaje de programación JAVA

Android Studio soporta los lenguajes de programación JAVA y Kotlin [57]. Para este proyecto se ha utilizado JAVA en lugar de Kotlin porque, al igual que sucede con Android Studio, este es el lenguaje aprendido en la asignatura Desarrollo de aplicaciones para dispositivos móviles de la titulación.

4.2.3 Camera2 y CameraX

Camera2 es una API de bajo nivel que reemplaza la API Camera [58].

CameraX es una API de nivel alto más fácil de usar que Camera2 [59].

Estas librerías se han utilizado para trabajar con la cámara.

La versión utilizada en ambas librerías es la 1.0.0-alpha05.

4.2.4 Torchvision lite y Pytorch lite

Son las librerías necesarias para ejecutar los modelos de redes neuronales en Android. Aunque también es posible utilizar la versión normal, sin “lite”, se ha utilizado esta porque es la recomendada para dispositivos móviles.

La versión utilizada en ambas librerías es la 2.1.0.

4.2.5 Otros

- GIT: Android Studio soporta distintos softwares de control de versiones. En este proyecto se ha utilizado GIT.
- Maven: Es una herramienta para construir proyectos. Se ha utilizado para agregar el repositorio Sonatype Nexus Repository [60].
- Sonatype Nexus Repository: Es un administrador de repositorios. Utilizado junto con Maven [61].
- JCenter: Es un repositorio público que contiene bibliotecas Android OSS. Se ha utilizado para añadir Torchvision lite y Pytorch lite [62].
- fbgemm: FBGEMM (Facebook GEneral Matrix Multiplication) es una biblioteca para entrenamiento e inferencia utilizada como backend de los operadores cuantificados de PyTorch en máquinas x86. Se utiliza para la cuantificación estática posterior al entrenamiento [63].

4.3 Hardware

Tanto el desarrollo del trabajo, como el entrenamiento de los modelos se ha llevado a cabo en un equipo personal. El hardware en el que se ejecute el modelo afecta directamente a la velocidad de entrenamiento del modelo y también limita la cantidad de memoria que puede utilizar.

4.3.1 Especificaciones del equipo

A continuación se muestran las especificaciones técnicas relevantes en el entrenamiento y ejecución de los modelos:

- CPU: AMD Ryzen 7 5700X3D (x86). Arquitectura Zen 3 con 8 núcleos a 3 GHz de frecuencia base.
- GPU: Nvidia GeForce RTX 4080 SUPER. Arquitectura Ada Lovelace. 16 GB de memoria dedicada (VRAM), 24 GB de memoria total disponible (16 GB de VRAM y 8 GB de RAM), 10240 núcleos CUDA a frecuencias base/máxima de 2295/2580 MHz (+110 MHz) y límite de potencia al 80%.
- RAM: 16GB DDR4 3600MHz.
- SO: Windows 11

4.3.2 Especificaciones del dispositivo móvil

La ejecución de la aplicación Android se ha llevado a cabo tanto en el emulador de Android Studio como en un dispositivo móvil real. Las especificaciones técnicas relevantes del dispositivo móvil son las siguientes:

- Dispositivo real: **Xiaomi Mi 9T Pro**.
- CPU: Snapdragon 855 (ARM).
 - Principales 1× Kryo 485 Prime (Cortex-A76) a 2.84 GHz
 - Secundarios 3× Kryo 485 Gold (Cortex-A76) a 2.42 GHz
 - Terciarios 4× Kryo 485 Silver (Cortex-A55) a 1.8 GHz
 - GPU: Adreno 640 a 585 MHz.
- RAM: 6GB 2133 MHz.
- SO: Android: 11.

- Dispositivo virtual: **Pixel 3a API 34 extension level 7 x86 64**.
- CPU: 8 núcleos.
- RAM: 1,5GB.
- SO: Android 14.

CAPÍTULO 5:

METODOLOGÍA Y

RESULTADOS



5.1 Planificación

5.1.1 Desglose de tareas

El desarrollo completo de este TFG se ha dividido en una serie de tareas. A continuación la Tabla 1 que muestra la lista de tareas con su tipo de etapa correspondiente, duración estimada y tareas precedentes necesarias para terminar una tarea.

ID	Tarea	Etapas	Duración	Precedente
1	Estudio preliminar y decidir el tema del TFG.	Investigación	9 días	
2	Estudio sobre IA y clasificación de imágenes.	Investigación	24 días	1
3	Estudio de las herramientas de trabajo (Python, PyTorch...).	Investigación	23 días	1
4	Creación del modelo y entrenamiento.	Desarrollo	18 días	
5	Cuantificación del modelo.	Desarrollo	6 días	3, 4
6	Pruebas de rendimiento y comparación con otros modelos.	Análisis	5 días	2, 5
7	Desarrollo de la aplicación Android.	Desarrollo	18 días	5
8	Pruebas de rendimiento en la aplicación.	Análisis	4 días	7
9	Redacción de la memoria.	Documentación	65 días	6, 8

Tabla 1. Desglose de tareas.

5.1.2 Diagrama de Gantt

El diagrama de Gantt es una herramienta para visualizar de forma gráfica las tareas y tiempo estimado dedicado a estas como se muestra en la Ilustración 12.

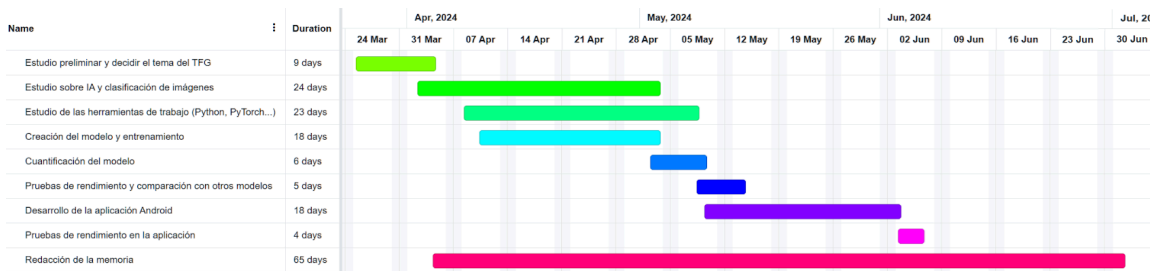


Ilustración 12. Diagrama de Gantt.

5.2 Creación del modelo y entrenamiento

5.2.1 Información del dataset

El dataset a utilizar ha sido obtenido de la página web Kaggle. Se trata de “Melanoma Cancer Image Dataset” [54]. Este dataset se ha creado para permitir a investigadores y profesionales crear modelos de aprendizaje automático para distinguir entre casos benignos y malignos. Cuenta con un total de 13900 imágenes obtenidas de recursos disponibles públicamente y tiene licencia Creative Commons CC0: Public Domain. No se espera que el dataset sea actualizado con nuevas imágenes.

Resolución	224x224
Canales	3 (RGB)
Imágenes de melanomas malignos para entrenamiento	5590
Imágenes de melanomas benignos para entrenamiento	6289
Imágenes de melanomas malignos para validación	1000
Imágenes de melanomas benignos para validación	1000
Relacion entrenamiento/validación	85,59/14,41 (%)

Tabla 2. Resumen del dataset.

Como se muestra en la Tabla 2, el dataset ya viene dividido entre conjunto de datos de entrenamiento y conjunto de datos de validación. Todas las imágenes del dataset son únicas (no tienen data augmentation) y han sido seleccionadas meticulosamente. La mayoría de las imágenes están centradas en una sección de piel muy pequeña, de unos cuatro centímetros cuadrados. Sin embargo, también hay imágenes a distancias mucho mayores en las que se pueden ver (total o parcialmente) manos, brazos, caras, etc.

5.2.2 Primeras decisiones del modelo y entrenamiento

Antes de empezar a crear un modelo₍₁₎ en Pytorch y comenzar con el entrenamiento, vamos a ver las variables posibles de estos.

Modelo	
Número de capas	Por decidir
Tipos de capas	El modelo tendrá como mínimo una capa convolucional y capa de salida lineal . Pero también puede tener capas de normalización, pooling, dropout, etc
Características de las capas	La capa de salida tiene que tener dos salidas (maligno y benigno). El resto de capas dependerá de
Funciones de activación	Por decidir. Puede ser ReLU, LeakyReLU, etc

Tabla 3. Variables básicas del modelo

Entrenamiento	
Algoritmo optimizador	Adam*
Funcion de perdida	Cross-Entropy Loss**
Número de épocas	Por decidir
Tamaño de lotes	Por decidir
Preparación de los datos	mean = [0.485, 0.456, 0.406], std = [0.229, 0.224, 0.225]*** Puede tener más modificaciones.

Tabla 4. Variables básicas de entrenamiento

*Es el algoritmo más utilizado para redes convolucionales [65].

** Es una de las funciones de pérdida más utilizadas en clasificación binaria [66].

***Es la normalización oficial utilizada en todos los modelos pre-entrenados de PyTorch [67].

En la Tabla 3 y Tabla 4 se encuentra un resumen de las variables que serán decididos posteriormente y los que serán comunes a todos los modelos posibles.

5.2.3 Primer modelo y análisis de las variables de entrenamiento

Este primer modelo es solo la base para decidir muchas de las variables de entrenamiento posteriores.

Modelo 1	
Capa convolucional	Conv2d(3, 8, 3) ₍₂₎
Pooling	AvgPool2d(3, 1) ₍₅₎
Función de activación	LeakyReLU() ₍₆₎
Normalización	BatchNorm2d(8) ₍₈₎
Abandono ₍₁₆₎	Dropout2d(3) ₍₉₎
Capa lineal	Linear(8*110*110, 2) ₍₃₎

Tabla 5. Variables del primer modelo

La Tabla 5 muestra la estructura de este modelo que comienza con una capa convolucional con 224 parámetros ($3*8*3*3 + 8$), contiene una capa de pooling, la función de activación, una capa de normalización, una capa de abandono (estas capas no contienen neuronas) y finaliza con una capa lineal con 193602 parámetros ($((8*110*110)*(2) + 2)$).

Entrenamiento	
Algoritmo optimizador	Adam(model.parameters(), 0.001) ₍₁₁₎
Funcion de perdida	CrossEntropyLoss() ₍₁₀₎
Número de épocas	40

Tamaño de lotes	32
Preparación de los datos	mean = [0.485, 0.456, 0.406], std = [0.229, 0.224, 0.225] Resize((112, 112))

Tabla 6. Variables básicas de entrenamiento

En la Tabla 6 se muestran las variables elegidas para el primer entrenamiento. El Resize es para acelerar el entrenamiento al reducir el tamaño de las imágenes.

Elegir el tamaño de lote para el entrenamiento

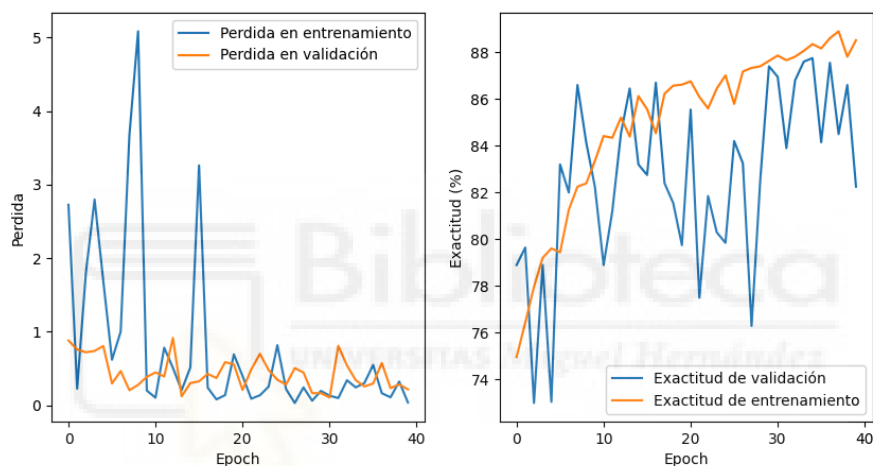


Ilustración 13. Gráfica de Pérdida y Exactitud. Modelo 1, entrenamiento 1.

La Ilustración 13 muestra la gráfica de pérdida en entrenamiento y validación y la gráfica de exactitud de validación y entrenamiento. En estas gráficas se puede observar que durante las veinte primeras épocas de entrenamiento el modelo mejora notablemente en entrenamiento. Sin embargo, es aunque las gráficas de validación muestran también muestran una mejora, es más difícil de apreciar debido a la alta variabilidad entre las épocas; se producen grandes mejoras o deterioros entre épocas.

La causa de esta alta inestabilidad puede ser el tamaño del lote. Idealmente los modelos se entrenan con todos los datos de entrada simultáneamente y se actualizan los pesos una vez por época. Pero debido al alto coste computacional, este proceso se realiza dividido en lotes de datos más pequeños que el conjunto total de datos.

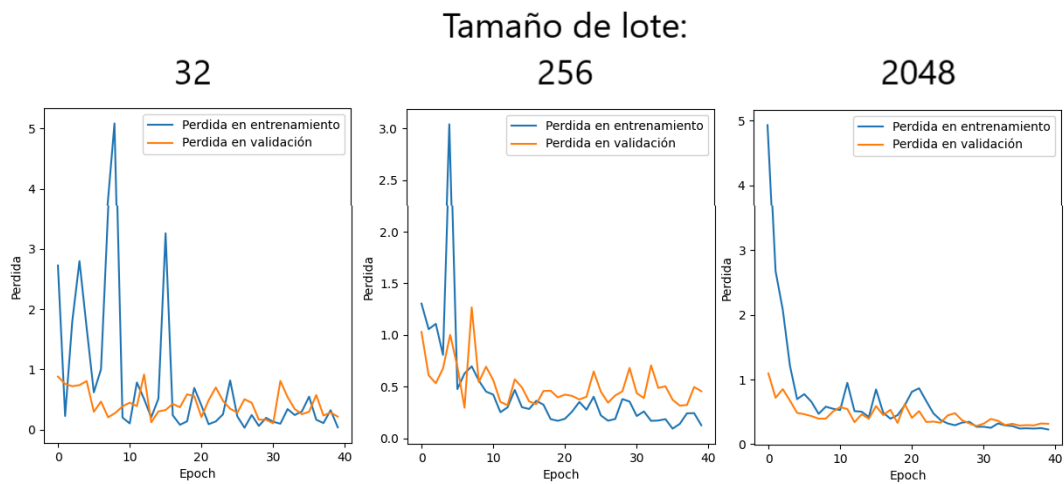


Ilustración 14. Gráficas de pérdida con distinto tamaño de lote.

La ilustración 14 confirma que un aumento en el tamaño de lote ayuda a mejorar la estabilidad en el aprendizaje del modelo.

Tamaño de lote	32	256	2048	8096
Tiempo de ejecución (s)	480	506	538	1158
Memoria utilizada (MB)	615	1124	5223	21913

Tabla 7. Tiempo de ejecución según el tamaño de lote

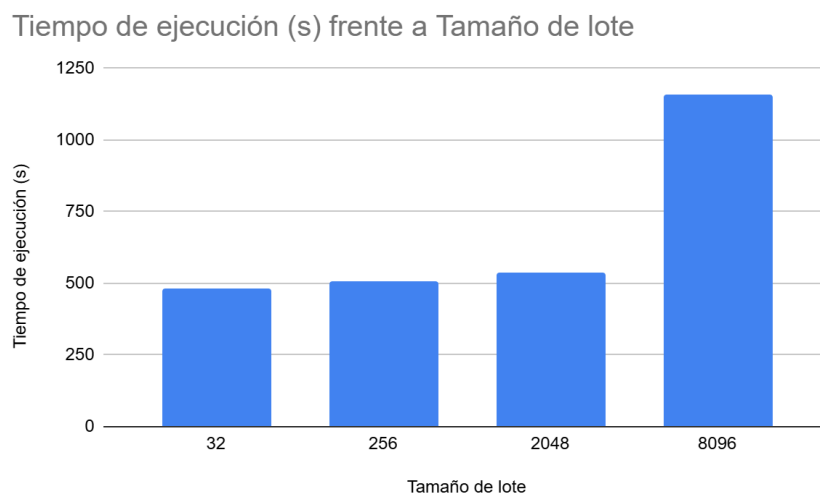


Ilustración 15. Tiempo de ejecución frente a Tamaño de lote.

La Tabla 7 muestra cómo aumenta el tiempo de ejecución y el uso de memoria al aumentar el tamaño de lote. En este caso la GPU dispone de 16GB de memoria dedicada, por tanto, el tiempo de ejecución aumenta dramáticamente si se supera dicha cantidad, como se puede observar en la Ilustración 15. Aumentando el tamaño de lote a 10240 se excede la capacidad de memoria disponible (24GB) y el modelo no se puede entrenar como se muestra en la Ilustración 16.

Es de esperar que en el caso de realizar la ejecución en CPU suceda algo similar, es decir, la velocidad de ejecución será menor si se excediera la capacidad de memoria RAM. Y en caso de superar la memoria total del sistema el modelo no se podrá entrenar.

```
torch.cuda.OutOfMemoryError: CUDA out of memory. Tried to allocate 3.69 GiB (GPU 0; 15.99 GiB total capacity; 16.35 GiB already allocated; 0 bytes free; 20.04 GiB reserved in total by PyTorch) If reserved memory is >> allocated memory try setting max_split_size_mb to avoid fragmentation. See documentation for Memory Management and PYTORCH_CUDA_ALLOC_CONF
```

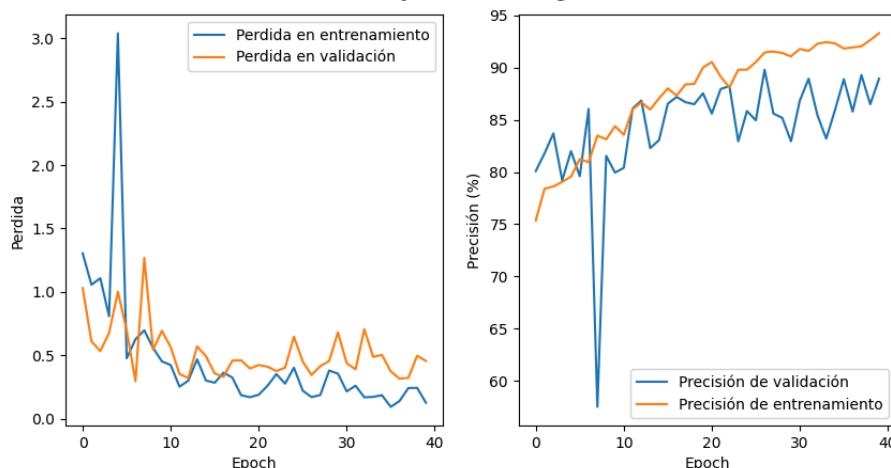
Ilustración 16. Memoria insuficiente para entrenar el modelo.

Por tanto, a partir de ahora en este trabajo se tratará de usar un tamaño de lote grande, pero siempre que no se exceda la capacidad de memoria dedicada durante la ejecución. Además hay que tener en cuenta que la complejidad del modelo también en el uso de memoria.

Elegir la tasa de aprendizaje

Otra variable importante en el entrenamiento es la tasa de aprendizaje proporcionada al algoritmo optimizador.

Tasa de aprendizaje 0,001



Tasa de aprendizaje 0,00001

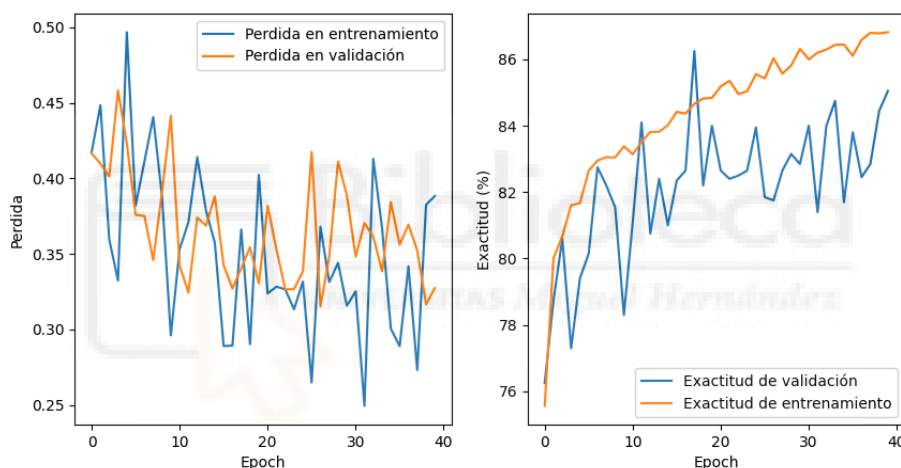


Ilustración 17. Gráficas de entrenamiento con distinta tasa de aprendizaje

En la Ilustración 17 se muestran de nuevo los resultados del entrenamiento del modelo, esta vez con la tasa de aprendizaje utilizada anteriormente (0,001) en la parte superior y 0,00001 en la parte inferior. En ambos casos el tamaño del lote es de 256. En esta imagen se puede observar que la tasa de aprendizaje menor ayuda al modelo a aprender de forma más estable (aunque visualmente las gráficas inferiores aparentan tener picos más pronunciados, las escalas de la imagen inferior son mucho menores). Pero esto produce una disminución en la capacidad de aprendizaje máxima por cada época, por tanto el modelo necesitará mayor número de épocas cuanto menor sea la tasa de aprendizaje. Esto se puede ver en la gráfica inferior de exactitud, que no llega a alcanzar valores tan altos como los de la gráfica superior.

Elegir el número de épocas

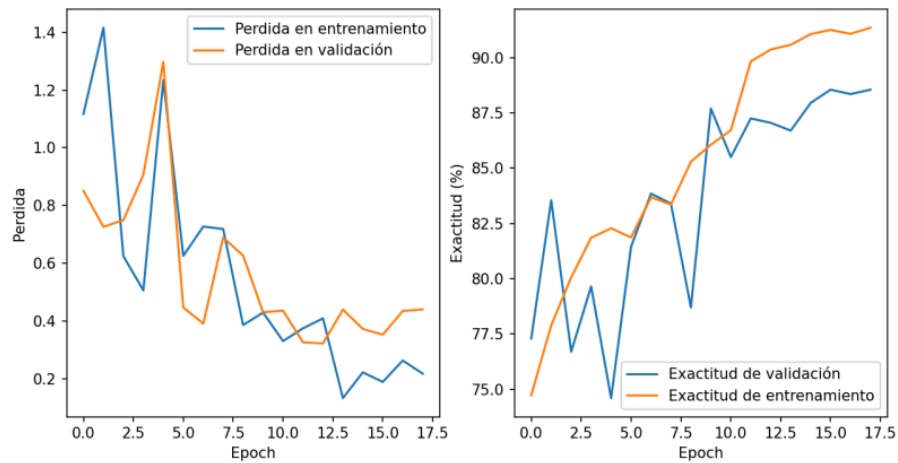
El caso anterior ha mostrado que hace falta un sistema para evitar que un modelo se ejecute con menos épocas de las necesarias. Pero, ¿cómo sabemos cuántas épocas se necesitan? Si entrenamos el modelo con muy pocas épocas, este no llegará a ser tan exacto como debería. Y si un modelo entrena demasiadas épocas se habrán desperdiciado tiempo y recursos, además en el peor de los casos podría producir overfitting.

Una solución a esto consiste en detener el entrenamiento una vez que el modelo deja de mejorar el valor de pérdida en validación durante un número concreto de épocas (paciencia) como se muestra a continuación:

```
if PerdidaMedia < mejorPerdida:
    mejorPerdida = PerdidaMedia
    pacienciaActual = 0
else:
    pacienciaActual += 1
    if pacienciaActual >= paciencia:
        print('Parada anticipada')
        break
```

Además, combinando esto con un planificador de la tasa de aprendizaje, se puede lograr que un modelo aprenda rápidamente durante las primeras épocas y posteriormente vaya reduciendo la tasa de aprendizaje.

Paciencia = 5



Paciencia = 10

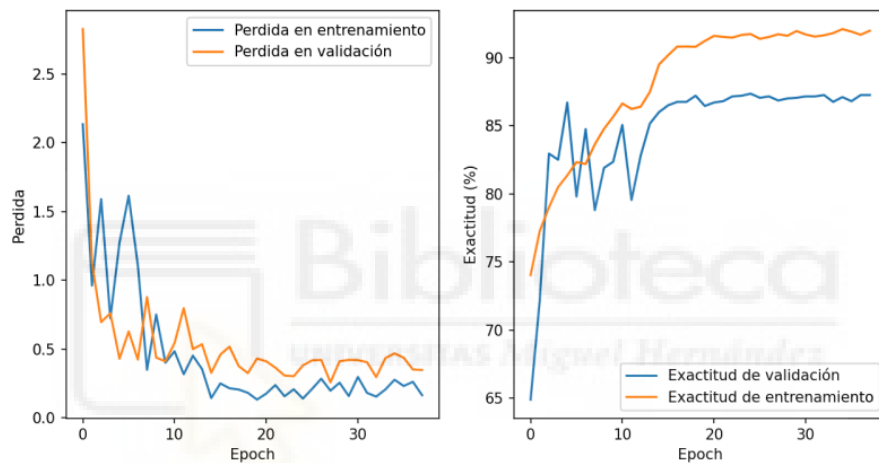


Ilustración 18. Comparación de paciencia = 5 y paciencia = 10.

La Ilustración 18 muestra el resultado del uso de una paciencia de cinco y diez antes de detener la ejecución además del un planificador de la tasa de aprendizaje con los siguientes valores:

```
optimizador = optim.Adam(model.parameters(), lr = 0.001)
ReduceLRonPlateau(optimizador, threshold = 0.01, factor = 0.1,
patience = 3, min_lr = 1e-8, verbose = True)(12)*
```

*patience representa el número de épocas sin mejora antes de reducir la tasa de aprendizaje. Más información en el glosario.

Como se puede observar en la Ilustración 18, al principio se asemeja al caso superior de la Ilustración 17, puesto que tienen las mismas variables. Sin embargo en la Ilustración 18 se puede ver claramente cómo se reduce la tasa de aprendizaje (baja hasta 0.00001) a lo largo de las épocas (especialmente en las gráficas de paciencia = 10), terminando con unos valores muy estables. En el caso de paciencia = 10 se puede observar que el modelo ya no mejora desde antes de la época 10, por tanto es una paciencia demasiado alta. Esto contrasta con el caso de tasa de aprendizaje 0,00001 (Ilustración 17), puesto que en ese caso el modelo no alcanza su valor óptimo tras más de veinte épocas más.

5.2.4 Segundo modelo y primer análisis de sus variables

Ahora que las variables de entrenamiento son bastante adecuadas, podemos centrar la atención en el modelo a partir de ahora. El primer modelo queda descartado y ahora se muestra uno inspirado en algo habitual en modelos de este tipo: varias capas convolucionales, pooling o normalizaciones y varias capas lineales [58].

Modelo 2	
Capa convolucional	Conv2d(3, 8, 5)
Función de activación	ReLU() ₍₇₎
Pooling	MaxPool2d(2, 2) ₍₄₎
Capa convolucional	Conv2d(8, 16, 5) ₍₂₎
Función de activación	ReLU()
Pooling	MaxPool2d(2, 2)
Capa convolucional	Conv2d(16, 26, 5) ₍₂₎
Función de activación	ReLU()
Pooling	MaxPool2d(2, 2)
Capa lineal	Linear(14976, 30)
Función de activación	ReLU()
Capa lineal	Linear(30, 30)
Función de activación	ReLU()

Capa lineal

Linear(30, 2)

Tabla 8. Variables del segundo modelo

La Tabla 8 muestra las variables de este nuevo modelo y la Tabla 9 las variables de entrenamiento.

Entrenamiento	
Algoritmo optimizador	Adam(model.parameters(), 0.001) ₍₁₁₎
Funcion de perdida	CrossEntropyLoss() ₍₁₀₎
Número de épocas	Máximo 100. Parada anticipada con paciencia 5
Tamaño de lotes	1024
Preparación de los datos	mean = [0.485, 0.456, 0.406], std = [0.229, 0.224, 0.225]
Planificador de la tasa de aprendizaje	ReduceLROnPlateau(optimizer, threshold = 0.01, factor = 0.1, patience = 3, min_lr = 1e-8)

Tabla 9. Variables de entrenamiento

La Tabla 9 muestra las variables utilizadas para el entrenamiento, se utilizarán estas variables para todos los casos excepto cuando se indique lo contrario.

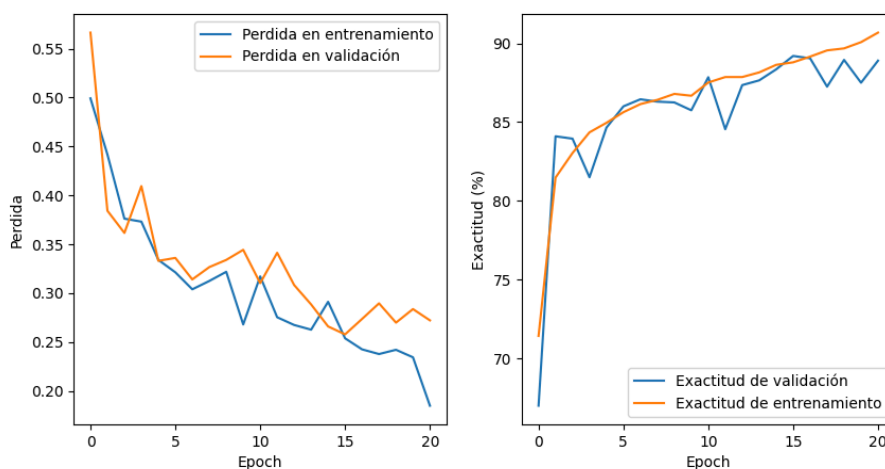


Ilustración 19. Gráficas del segundo modelo

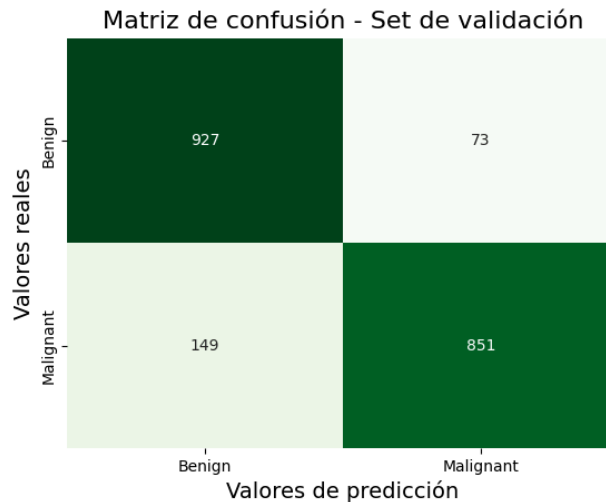


Ilustración 20. Matriz de confusión del segundo modelo

En la Ilustración 19 se muestran las gráficas del segundo modelo que muestran mejoras tanto en la reducción de pérdida en validación como una mayor exactitud en validación. En la Ilustración 20 se muestra la matriz de confusión, en esta se puede observar que el modelo identifica erróneamente 149 tumores malignos como benignos. Por tanto, la precisión del modelo para identificar tumores benignos es: $\frac{927}{927 + 149} = 86,1\%$, pero la exactitud se encuentra por debajo del 90%.

5.2.5 Tercer modelo. Aumento de la complejidad

El segundo modelo muestra en la Ilustración 19 un posible underfitting, puesto que los valores de validación están muy próximos a los de entrenamiento. Por ese motivo en el tercer modelo se ha aumentado el número de neuronas del modelo de las capas intermedias. El tamaño de lote utilizado es 512.

Modelo 3	
Capa convolucional	Conv2d(3, 6, 3)
Función de activación	ReLU() ₍₇₎
Pooling	MaxPool2d(2, 2)
Capa convolucional	Conv2d(6, 32, 5)
Función de activación	ReLU()

Pooling	MaxPool2d(2, 2)
Capa lineal	Linear(89888, 1000)
Función de activación	ReLU()
Capa lineal	Linear(1000, 500)
Función de activación	ReLU()
Capa lineal	Linear(500, 2)

Tabla 10. Variables del tercer modelo

Como se puede ver en la Tabla 10, también se han realizado otros cambios, como reducir el número de capas convolucionales y el tamaño del kernel de la primera capa.

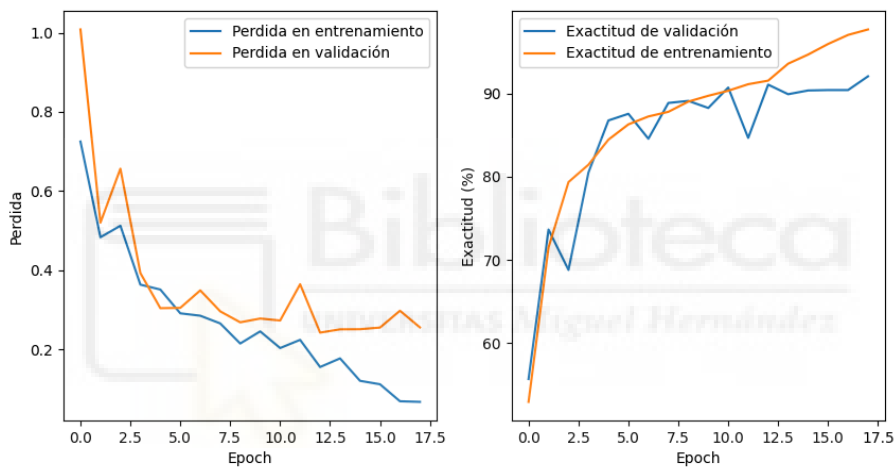


Ilustración 21. Gráficas del tercer modelo

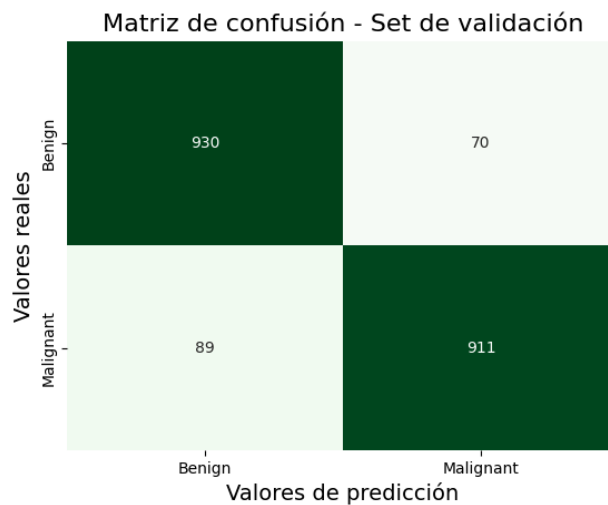


Ilustración 22. Matriz de confusión del tercer modelo

En la Ilustración 21 y la Ilustración 22 se puede ver que el modelo ha mejorado en todos los aspectos levemente en todos los aspectos.

5.2.6 Cuarto modelo. Tamaño del primer kernel

El cambio más relevante en este modelo es la reducción del tamaño del kernel de la primera capa convolucional a 2x2, puesto que al observar las características de las imágenes se ha considerado que esto podría mejorar el modelo [69] [70]. El tamaño de lote utilizado es 512.

Modelo 4	
Capa convolucional	Conv2d(3, 32, 2)
Función de activación	ReLU() ₍₇₎
Pooling	MaxPool2d(2, 2)
Capa convolucional	Conv2d(32, 64, 2)
Función de activación	ReLU()
Pooling	MaxPool2d(2, 2)
Capa lineal	Linear(193600, 1000)
Función de activación	ReLU()
Capa lineal	Linear(1000, 500)
Función de activación	ReLU()
Capa lineal	Linear(500, 2)

Tabla 11. Variables del cuarto modelo

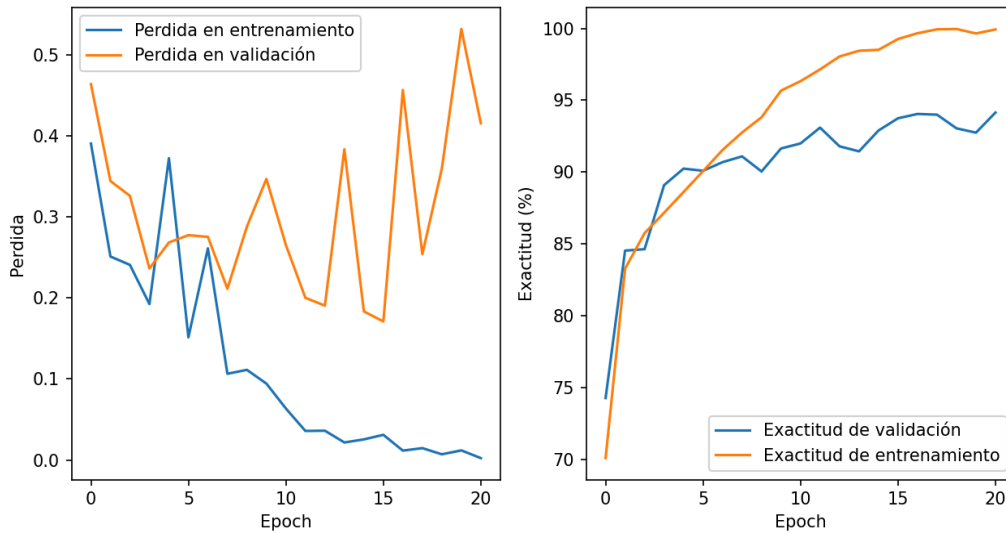


Ilustración 23. Gráficas del cuarto modelo

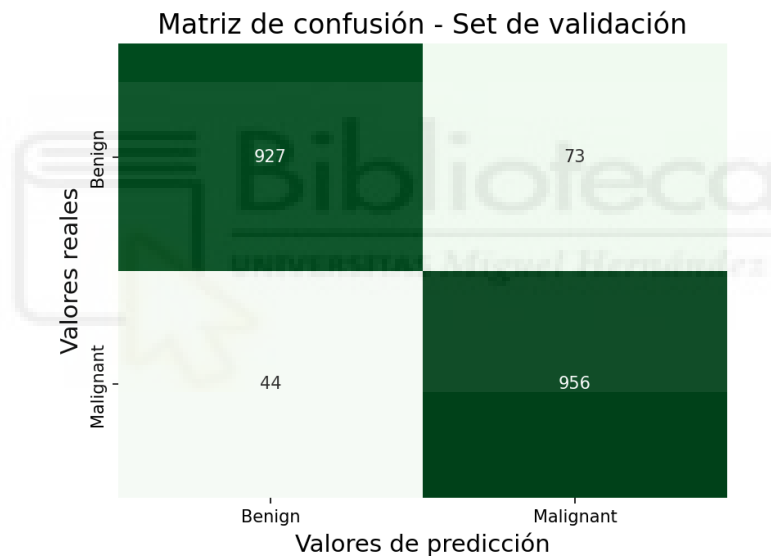


Ilustración 24. Matriz de confusión del cuarto modelo

Como se muestra en la la Ilustración 23 y la Ilustración 24, este modelo tiene mayor exactitud que los anteriores, llegando al 94%.

5.2.7 Quinto modelo. Últimos ajustes

La Ilustración 23 se parece mostrar overfitting puesto que la pérdida en validación parece demasiado alta. En este modelo se va a reducir la complejidad para reducir el overfitting. El tamaño de lote utilizado es 512.

Modelo 5	
Capa convolucional	Conv2d(3, 28, 2)
Función de activación	ReLU() ₍₇₎
Pooling	MaxPool2d(2, 2)
Capa convolucional	Conv2d(28, 28, 2)
Función de activación	ReLU()
Pooling	MaxPool2d(2, 2)
Capa lineal	Linear(84700, 1000)
Función de activación	ReLU()
Capa lineal	Linear(1000, 500)
Función de activación	ReLU()
Capa lineal	Linear(500, 2)

Tabla 12. Variables del quinto modelo

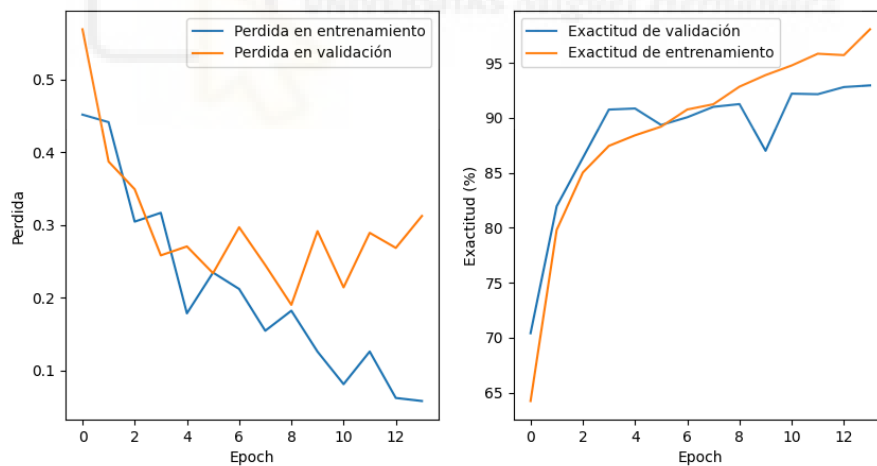


Ilustración 25. Gráficas del quinto modelo

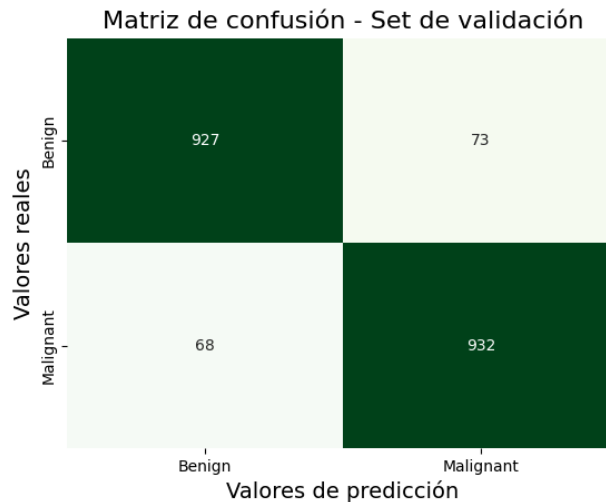


Ilustración 26. Matriz de confusión del cuarto modelo

Como se puede ver en la Ilustración 25 y la Ilustración 26 el overfitting se ha reducido considerablemente. Este modelo se encuentra muy cerca al anterior en la exactitud, y la pérdida en valoración ha mejorado de 0,5-0,4 a ~0,3. Además, debido a la reducción de la complejidad, la velocidad de ejecución del modelo y la cantidad de memoria que necesita es inferior al modelo anterior.

En este punto del trabajo se ha intentado mejorar el modelo realizando otros cambios sobre el modelo, como añadir data augmentation (con “RandomHorizontalFlip()” y “RandomRotation(degrees=90)”) y dropouts (Dropout2d(0.2) en capas intermedias). Pero ninguna de estas ha proporcionado alguna mejora sobre el modelo.

5.2.8 Cuantificación del modelo

En este apartado se va a ver las dos formas que se han utilizado para cuantificar, puesto que la tercera forma no se puede aplicar a capas convolucionales.

Cuantificación estática posterior al entrenamiento

Para realizar esta cuantificación se ha añadido el siguiente fragmento de código después de la declaración del modelo y antes del entrenamiento:

```
backend = "fbgemm"
```

```

model.qconfig = torch.quantization.get_default_qconfig(backend)
torch.backends.quantized.engine = backend
model_static_quantized = torch.quantization.prepare(model,
inplace=False)
model_static_quantized =
torch.quantization.convert(model_static_quantized,
inplace=False)

```

Se ha utilizado FBGEMM en lugar de QNNPACK debido a que la máquina no soporta QNNPACK.

Además, esta y otras cuantificaciones solamente funcionan si se utiliza el CPU, es decir no se puede utilizar la GPU para acelerar la inferencia.

Cuantificación consciente del entrenamiento

Para realizar esta cuantificación se ha añadido la primera línea del código siguiente al principio del `__init__` del modelo y la última línea al final del `__init__`:

```

self.quant = torch.quantization.QuantStub()
#capas del modelo
self.dequant = torch.quantization.DeQuantStub()

```

En el forward se ha hecho lo mismo con este código:

```

x = self.quant(x)
#forward
x = self.dequant(x)

```

5.3 Pruebas de rendimiento y comparación con otros modelos

En este apartado se va a realizar una comparación del rendimiento y exactitud del modelo 5 al estar o no cuantificado. También se van a realizar pruebas con otros modelos pre-entrenados.

5.3.1 Modelo 5. Rendimiento con y sin cuantificación

Debido a que se van a ejecutar en CPU, el tamaño de lote utilizado va a ser de 64.

Sin cuantificación (SC)

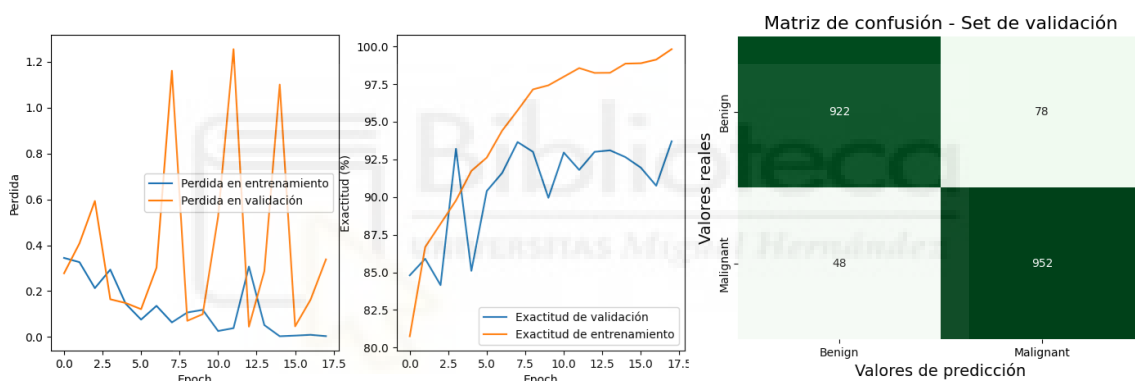


Ilustración 27. Modelo 5 sin cuantificación

Cuantificación consciente del entrenamiento (CCE)

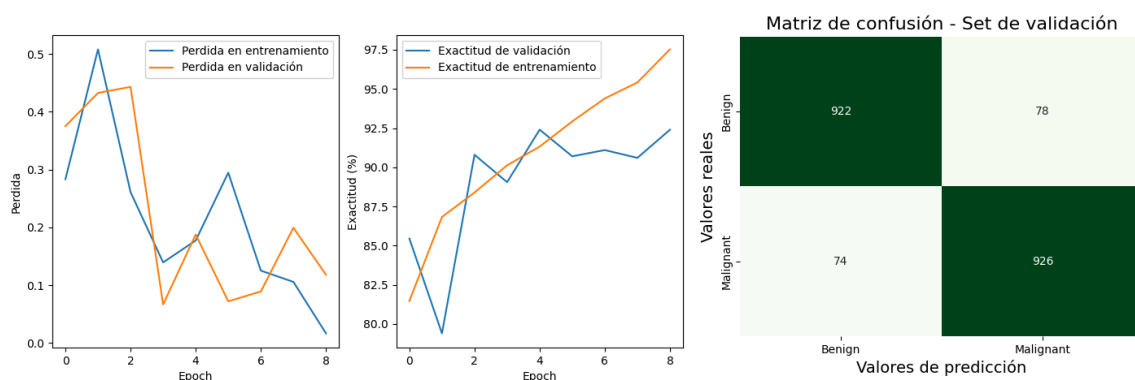


Ilustración 28. Modelo 5 con cuantificación consciente del entrenamiento

Cuantificación estática posterior al entrenamiento (CEPE)

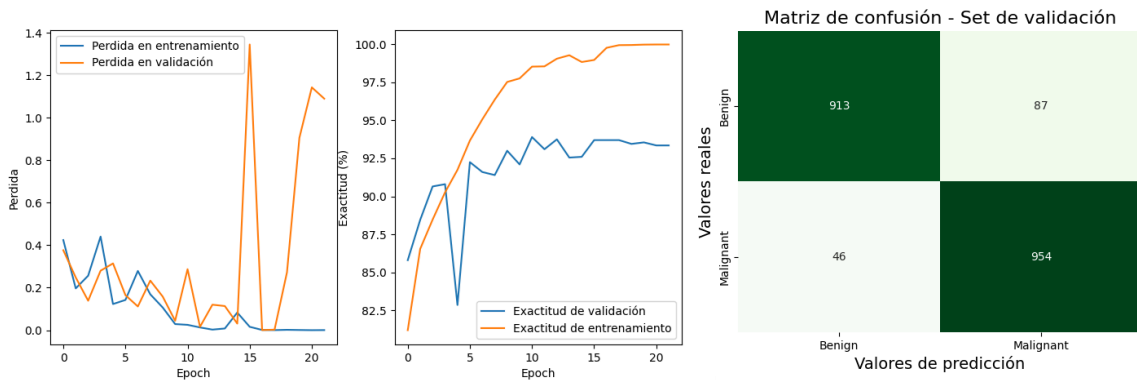


Ilustración 29. Modelo 5 con cuantificación estática posterior al entrenamiento

Las Ilustraciones 27, 28 y 29 muestran las gráficas del modelo 5 sin cuantificación, con cuantificación consciente del entrenamiento y con cuantificación estática posterior al entrenamiento. En estas se puede apreciar que las diferencias entre el modelo sin cuantificación y con cuantificación pueden deberse a las diferencias habituales entre distintas ejecuciones y al tamaño de lote (64). Esto también se muestra en las siguientes ilustraciones, que muestran los resultados de su última época y el tiempo medio de ejecución por época (Ilustración 32).

Exactitud en entrenamiento y Exactitud en validación

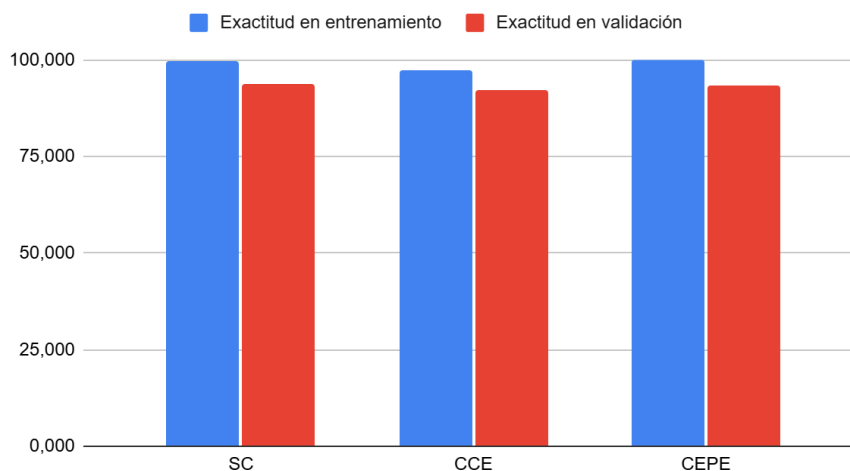


Ilustración 30. Exactitud en entrenamiento y exactitud en validación

Pérdida en entrenamiento y Pérdida en validación

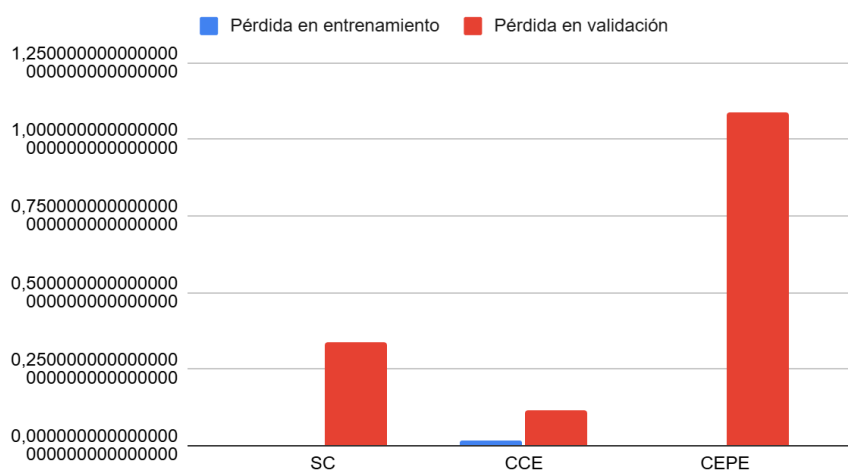


Ilustración 31. Pérdida en entrenamiento y pérdida en validación

Tiempo de ejecución/Nº de épocas

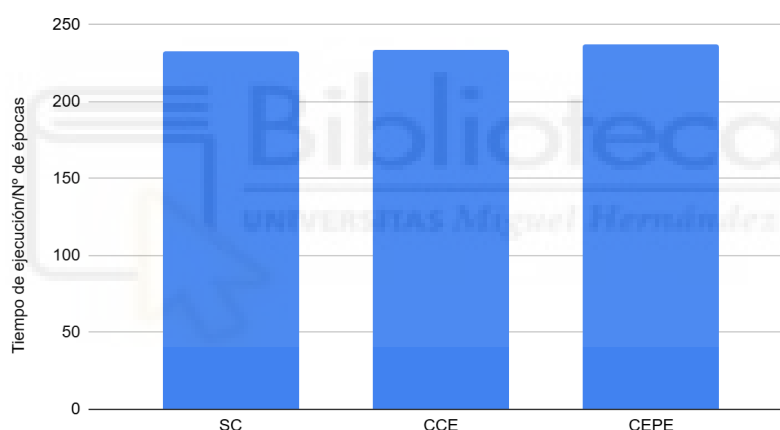


Ilustración 32. Tiempo medio por época

5.3.2 Comparativa con modelos pre-entrenados

Los modelos pre-entrenados probados son:

- VGG16 [71]: Visual Geometry Group es una red muy conocida puesto que fue una de las primeras en utilizarse en aprendizaje profundo. Se presentó en 2014 con dos modelos, uno de dieciséis (el utilizado en este trabajo) y otro de diecinueve capas. Tiene 138357544 parámetros.
- EfficientNet_v2_1 [72]: La primera versión de esta red se presentó en 2019, mostrando ocho versiones, cada una con número distinto de parámetros. Esta red mostró una gran exactitud en distintos datasets, además de requerir menor

número de parámetros para alcanzar la misma exactitud que otras redes como RedNet e Inception. Es la red utilizada que ha obtenido la mayor exactitud en el dataset utilizado [64]. El modelo elegido tiene 118515272 parámetros.

- Mobilenet_v2 [73]: Se presentó en 2017 con la intención de contradecir la tendencia general hasta ese momento, que consistía en hacer redes cada vez más complejas para alcanzar mayor exactitud pero con una baja eficiencia. Esta red mostró algunas decisiones de diseño que permiten desarrollar una red eficiente. Es una red utilizada en dispositivos móviles por su alta eficiencia. Este modelo tiene 3504872 parámetros.
- El modelo 5 desarrollado en este trabajo tiene:
 $((2*2*3*28)+28)+((2*2*28*28)+28)+((84700*1000)+1000)+((1000*500)+500)+((500*2)+2) = 85206030$ parámetros.

Debido a que EficientNet_v2_1 con un tamaño de lote de 64 necesita 28GB de memoria, y por tanto excede el máximo del sistema, el tamaño utilizado para todas las redes es 32.

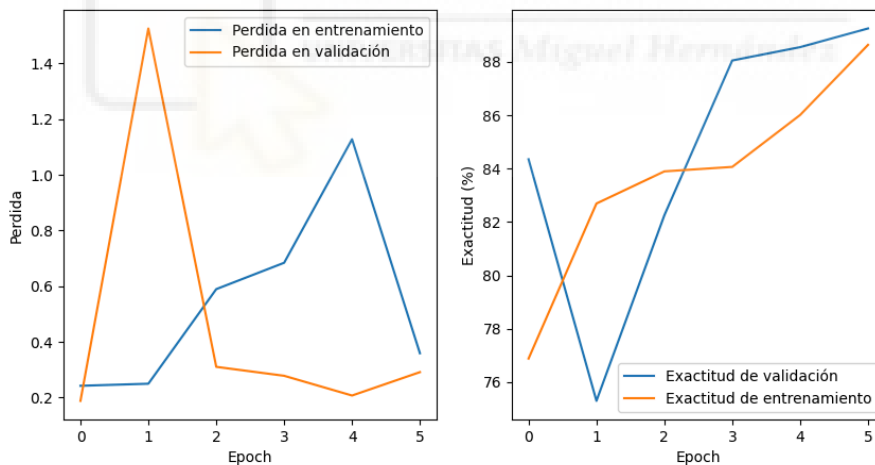


Ilustración 33. Gráficas de VGG16

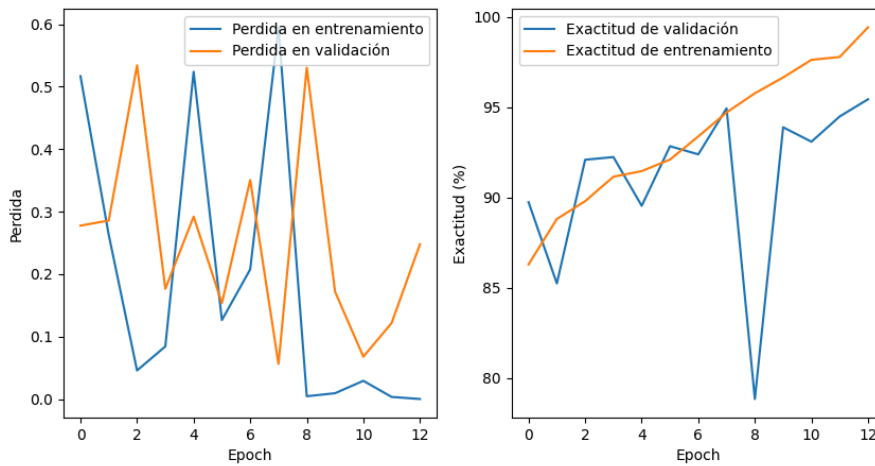


Ilustración 34. Gráficas de EfficientNet_v2_l

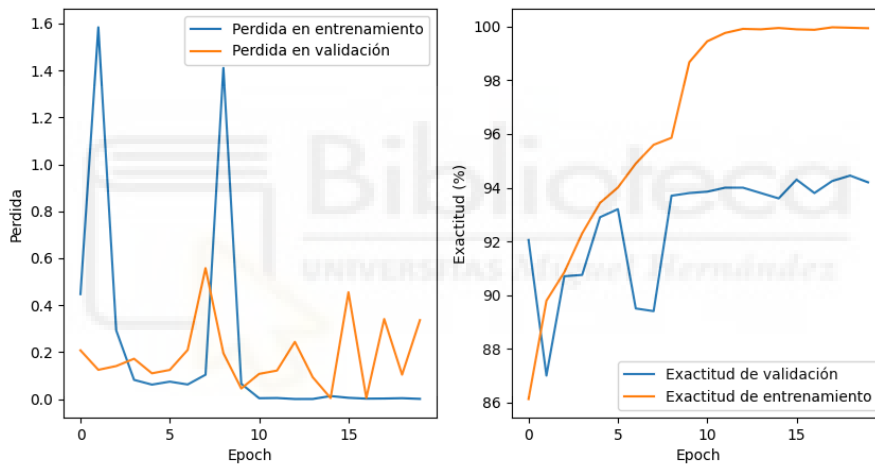


Ilustración 35. Gráficas de Mobilenet_v2

Al ejecutar Mobilenet_v2 con el procesamiento en la GPU, se ha comprobado que este modelo utiliza la CPU además de la GPU. De todos los modelos que se muestran aquí, este es el único modelo en el que sucede esto.

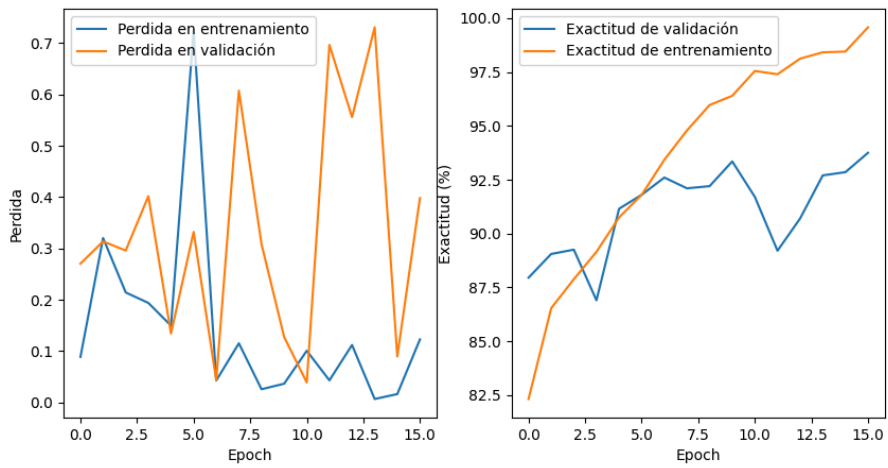


Ilustración 36. Gráficas de Modelo 5

EfficientNet_v2_l, Mobilenet_v2, Modelo 5 y VGG16

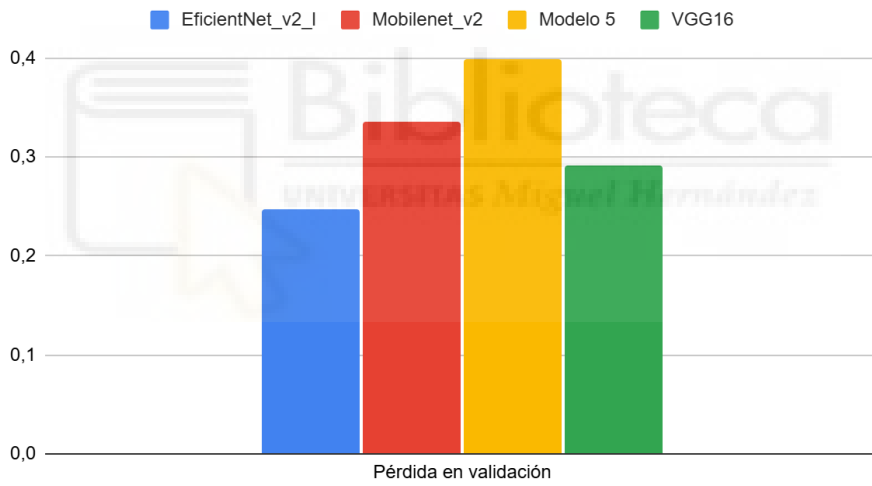


Ilustración 37. Comparativa de pérdida en validación en los modelos pre-entrenados y el modelo 5

Exactitud en validación

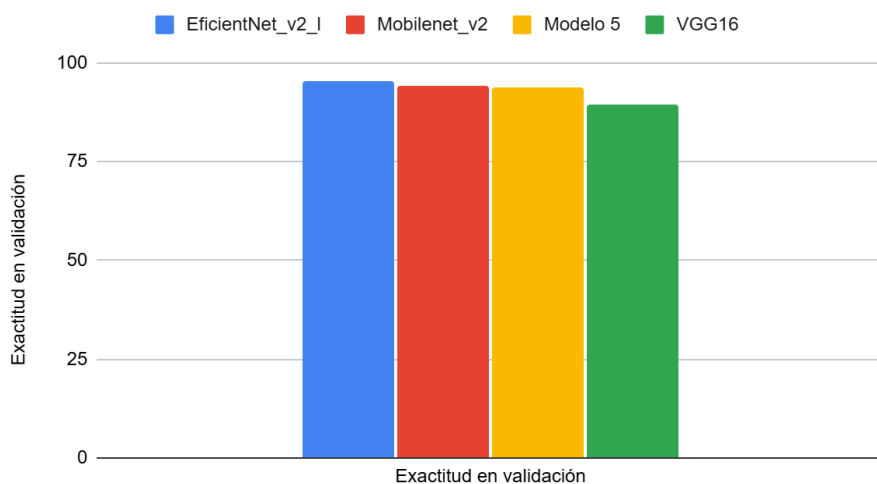


Ilustración 38. Comparativa de exactitud en los modelos pre-entrenados y el modelo 5

VGG16, EfficientNet_v2_I, Mobilenet_v2 y Modelo 5

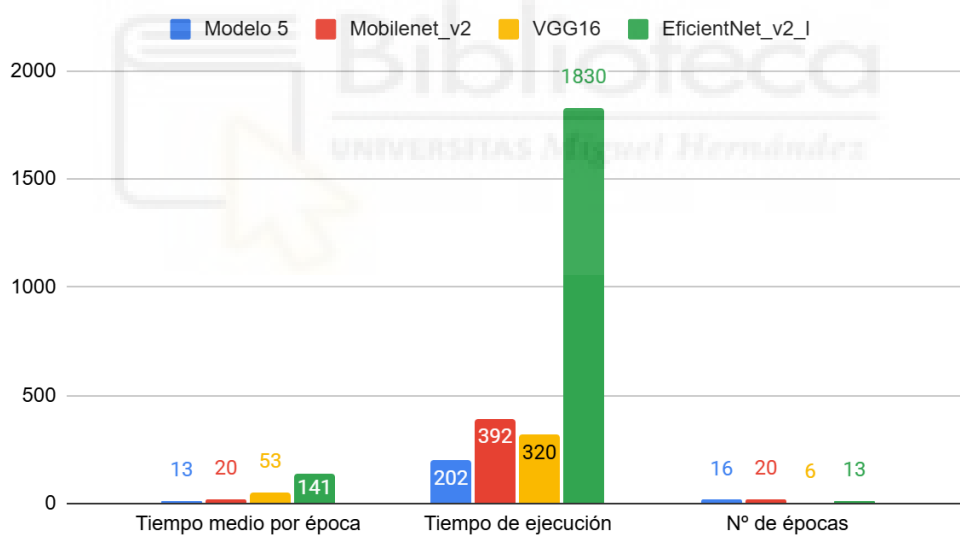


Ilustración 39. Comparativa de los tiempos de ejecución del entrenamiento en los modelos pre-entrenados y el modelo 5

Red	Modelo 5	Mobilenet_v2	VGG16	EfficientNet_v2_I
Tiempo por época* (s)	202	296	2856	11476

Tiempo total de entrenamiento estimado* (m)	53	98	285	2486
--	----	----	-----	------

Tabla 13. Comparativa de tiempos de entrenar las redes con cuantificación estática posterior al entrenamiento (CPU)

*El tiempo por época se corresponde con el tiempo que la red tarda en realizar un entrenamiento de una época.

*El tiempo total de entrenamiento estimado se ha calculado multiplicando el tiempo por época con el número de épocas que el modelo necesitó en el apartado anterior.

En las ilustraciones 37 y 38 que el mejor modelo es EficientNet_v2_1, seguido por Mobilenet_v2 y muy cerca de este está el Modelo 5. Sin embargo el tiempo de ejecución más bajo es el del Modelo 5, siendo casi un 50% más rápido que Mobilenet_v2 y decenas de veces más rápido que el resto de modelos.

5.4 Aplicación para dispositivos móviles Android

5.4.1 Análisis de requisitos

Actores

Las siguientes tablas muestran los posibles actores de la aplicación:

Actor	Usuario
Descripción	Este actor engloba a todos los usuarios de la aplicación. Desde dermatólogos que no necesiten la aplicación para realizar un diagnóstico, pero puedan utilizarla como una herramienta de apoyo. Personal sanitario no especializado en dermatología que puedan necesitar una segunda opinión. Y por último usuarios sin conocimientos médicos.
Casos de uso	C.U. 1: Realizar una análisis visual de la piel

Tabla 14. Actor - Usuario.

Actor	Desarrollador
Descripción	Este actor tiene control de las funcionalidades de la aplicación.

Casos de uso	C.U. 1: Realizar una análisis visual de la piel C.U. 2: Modificar un modelo
---------------------	--

Tabla 15. Actor - Desarrollador.

Casos de uso

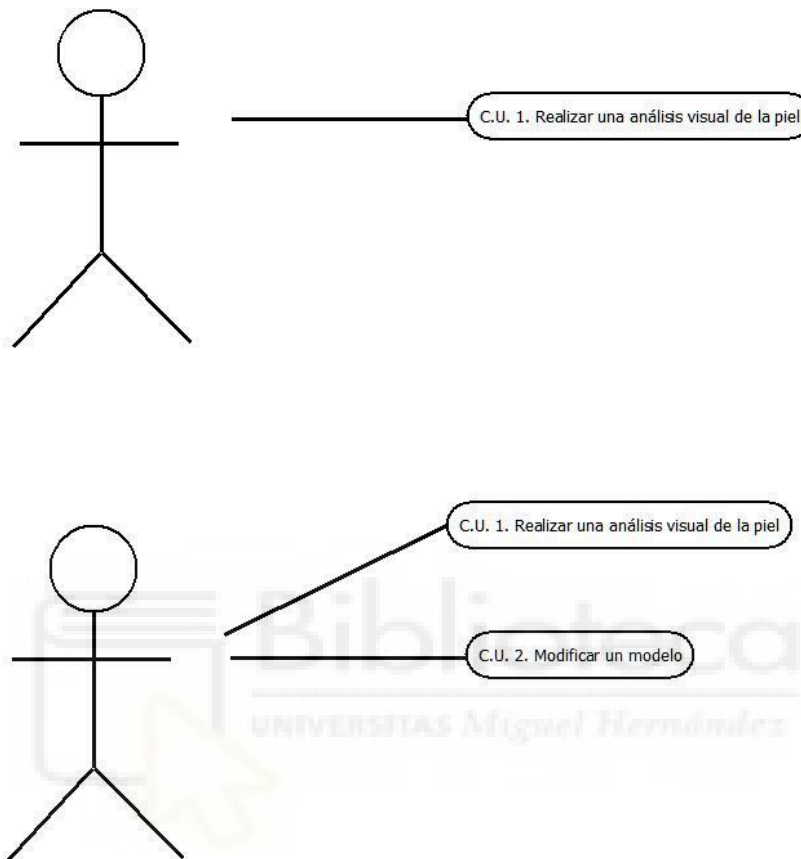


Ilustración 40. Diagrama de casos de uso.

C.U. 1	Realizar una análisis visual de la piel
Actores	Usuario
Descripción	La aplicación comienza a utilizar la cámara en modo análisis de imagen y ejecuta el modelo sobre las imágenes obtenidas por la cámara. La aplicación muestra por pantalla la imagen que recibe de la cámara y en la parte inferior se muestra en tiempo la probabilidad de estar detectando un melanoma maligno o benigno. También se muestra el número de fotogramas por segundo que se están analizando y los milisegundos que tarda en cada fotograma.
Secuencia normal	P1 - Acceder a la sección de análisis de imagen de la aplicación P2 - Seleccionar el modelo a utilizar
Frecuencia	Alta
Importancia	Alta

Tabla 16. Caso de uso 1.

C.U. 2	Modificar un modelo
Actores	Desarrollador
Descripción	El desarrollador añade, elimina o modifica un modelo de la aplicación
Secuencia normal	P1 - Modifica la aplicación P2 - Distribuye la nueva versión
Frecuencia	Baja
Importancia	Alta

Tabla 17. Caso de uso 2.

Requisitos funcionales

RF-1	Realizar un análisis de imagen
Descripción	Todos los usuarios pueden usar la cámara en modo análisis de imagen y ejecutar el modelo de clasificación de imágenes para mostrar por pantalla el porcentaje de certeza que tiene de si se está enfocando a un melanoma maligno o benigno.

Tabla 18. Requisito funcional 1.

RF-2	Cambiar el modelo de clasificación
Descripción	Todos los usuarios pueden elegir cuál de los modelos de clasificación de imagen van a utilizar.

Tabla 19. Requisito funcional 2.

RF-2	Añadir/quitar modelos
Descripción	El desarrollador puede añadir y quitar modelos. Estos modelos pueden estar entrenados para clasificar otro tipo de enfermedades.

Tabla 20. Requisito funcional 3.

Requisitos no funcionales

RNF-1	Alto rendimiento
Descripción	La aplicación ofrece un alto rendimiento, permitiendo ser ejecutada por dispositivos de bajas capacidades de cálculo.

Tabla 21. Requisito no funcional 1.

RNF-2	Fácil de usar
Descripción	La interfaz de la aplicación es sencilla y además contiene toda la información necesaria para entender su funcionamiento.

Tabla 22. Requisito funcional 3.

RNF-3	Escalabilidad
Descripción	El desarrollador puede añadir y mejorar fácilmente las funcionalidades de la aplicación.

Tabla 23. Requisito funcional 3.

5.4.2 Desarrollo de la aplicación

Esta aplicación ha sido desarrollada gracias a android-demo-app [61] que está disponible en Github y contiene distintos ejemplos de modelos en Android.

La aplicación utiliza la cámara en modo análisis de imágenes, por tanto la aplicación no guarda fotografías ni videos. Se muestran por pantalla las imágenes que capta la cámara y el CPU analiza dichas imágenes utilizando el modelo de red neuronal.

Para mostrar el porcentaje de certeza que tiene el modelo al clasificar las imágenes, por ejemplo 70% benigno y 30% maligno, es necesario normalizar la salida obtenida en logits. Normalmente esto se hace con la función softmax, pero como en este caso solamente hay dos salidas, se ha hecho implementando la función sigmoide en la aplicación: $Porcentaje = \frac{1}{1 + \text{Math.exp}(-\text{logit})}$

5.4.3 Importar el modelo a la aplicación

Para importar el modelo hace falta seguir los siguientes pasos:

- Ejecutar el entrenamiento exclusivamente con el CPU
- Guardar el modelo:

```
traced_script_module = torch.jit.script(model)(13)  
torchscript_model_optimized = optimize_for_mobile(traced_script_module)(14)  
torchscript_model_optimized._save_for_lite_interpreter("modelo.pt")(15)
```

- El uso de “optimize_for_mobile()” no es obligatorio.

- Añadir las dependencias de PyTorch Lite y TorchVision Lite a la aplicación:


```
implementation 'org.pytorch:pytorch_android_lite:2.1.0'  
implementation 'org.pytorch:pytorch_android_torchvision_lite:2.1.0'
```

- Introducir las etiquetas correspondientes a las clases del modelo

```
public static String[] MODEL_CLASSES = new String[]{  
    "Benigno",  
    "Maligno"  
};
```

- Agregar el modelo a los archivos de la aplicación y al código:

```
final Intent intent = new Intent(VisionListActivity.this,  
ImageClassificationActivity.class);  
intent.putExtra(ImageClassificationActivity.INTENT_MODULE_ASSET_NAME,  
"modelo.pt");
```

Metodología de pruebas del funcionamiento correcto de la aplicación

Para comprobar que el modelo funciona correctamente en la aplicación se han seguido los siguientes pasos:

- Instalar la aplicación en un dispositivo móvil.
- Mostrar imágenes del dataset en una pantalla y analizarlas con la aplicación.
 - Situar el móvil a suficiente distancia para evitar el efecto Muaré.
 - Ampliar la imagen para compensar la distancia.
 - Añadir un relleno a los bordes de la imagen.

Tras seguir estos pasos y probar a distintas distancias y porcentaje de ampliación de la imagen se ha comprobado que la aplicación clasifica correctamente las imágenes. Además se observa que las imágenes de entrenamiento suelen clasificarse con un porcentaje de certeza más alto que el de las imágenes de validación, lo cual también demuestra el correcto funcionamiento de la aplicación.

5.5 Rendimiento en la aplicación

Para probar los modelos en la aplicación estos se han entrenado con el CPU, con “optimize_for_mobile()” y cuantificación estática posterior al entrenamiento (el rendimiento ha sido el mismo al usar cualquiera de estos o ambos a la vez) para probar el rendimiento optimizados. Para su rendimiento sin optimización no se ha utilizado ninguna cuantificación ni “optimize_for_mobile()”. Se han ejecutado en un dispositivo móvil físico y el emulador de Android Studio.

Tiempo medio para procesar una imagen (ms)								
Red	Modelo 5		Mobilenet_v2		VGG16		EficientNet_v2_1	
Optimizado	Sí	No	Sí	No	Sí	No	Sí	No
Dispositivo real	60	195	61	156	540	524	577	1945
Dispositivo virtual	21	38	29	94	640	280	698	1528

Tabla 24. Tiempo medio para procesar una imagen en la aplicación

Los resultados de la Tabla 24 muestran que la optimización reduce enormemente la velocidad de ejecución, excepto en la red VGG16. Los modelos con mejor rendimiento son el Mobilenet_v2 y el modelo desarrollado durante este proyecto: el Modelo 5.

Los modelos VGG16 y EficientNet_v2_1 tienen un rendimiento muy bajo, especialmente EficientNet_v2_1 sin optimización, que no alcanza ni una imagen por segundo. Además, es relevante recordar que estos modelos están procesando imágenes de muy baja resolución (224x224 píxeles) y su rendimiento está directamente relacionado a esto.

CAPÍTULO 6:

CONCLUSIONES Y

TRABAJOS

FUTUROS



6.1 Conclusiones

Durante la realización de este trabajo se ha aprendido como el tamaño de los lotes de los modelos afectan a estos en el entrenamiento y cómo pueden ralentizar su entrenamiento.

También se ha aprendido a entrenar un modelo con una tasa de aprendizaje y número de épocas variable gracias al planificador de la tasa de aprendizaje y programar una parada anticipada en el aprendizaje.

Se han probado multitud de combinaciones de capas convolucionales, tamaño del kernel, otros parámetros, capas de pooling, etc. Se ha analizado el rendimiento de la red a partir de la pérdida, la exactitud y precisión. Gracias a todo esto se ha llegado a mejorar el modelo de clasificación en varias ocasiones, así como también su rendimiento computacional.

En la parte de cuantificación se ha aprendido que la reducción en la exactitud es tan baja que resulta muy difícil medirla, pero la mejora en el rendimiento es significativa. También se ha podido comprobar cómo la complejidad de un modelo afecta a la velocidad de inferencia.

También se ha aprendido a utilizar modelos pre-entrenados que funcionan muy bien, especialmente MobileNet, que ha demostrado ser capaz de clasificar las imágenes sin problemas y con un coste computacional muy bajo.

Uno de los aspectos más complicados de este trabajo ha sido llevar un modelo de PyTorch a Android. Esto se ha debido a multitud de motivos, como lo relativamente nuevo y poco habitual que es esto y por tanto lo limitada que está la información. También por lo rápido que están cambiando las cosas en este campo y, por supuesto, por mi falta de experiencia en este tema.

Uno de los descubrimientos más importantes de este trabajo ha sido el hecho de tener que entrenar una red con el CPU para utilizarla en Android (al menos actualmente) y

cómo afecta esto negativamente al tiempo de entrenamiento, especialmente cuando se requiere más memoria de la disponible en RAM.

Al realizar la aplicación se ha podido trabajar con algunos conceptos por primera vez, como utilizar la cámara para análisis de imágenes o medir el rendimiento de una aplicación en un teléfono móvil.

Finalmente me ha parecido interesante realizar este trabajo, especialmente abordar el desarrollo de la red neuronal ha resultado ser algo muy distinto a lo que estoy acostumbrado; puesto que en lugar de resolver un problema y trasladar la solución a un programa, en este caso es el programa el que “busca” la solución y yo tengo que guiarlo para que alcance una solución mejor.

6.2 Trabajos futuros

Antes de finalizar este trabajo se ha observado que este proyecto podría ser mucho más amplio y ambicioso. Tanto con mejoras en el modelo como en la aplicación.

El modelo podría mejorarse realizando un análisis de las imágenes del dataset para averiguar cómo son las características que diferencian los melanomas malignos de los benignos. De modo que se pueda desarrollar un modelo que se centre en diferenciar dichas características. El procedimiento que se ha seguido para el entrenamiento y desarrollo del modelo también es mejorable.

La aplicación podría incluir más funcionalidades:

- Sacar una foto, analizarla y guardarla.
- Analizar una foto de la galería.
- Implementar un servicio con un servidor que permita a los usuarios de los dispositivos móviles enviar imágenes para que puedan ser identificadas manualmente por profesionales, recibir una respuesta con el resultado y añadir la foto al dataset de entrenamiento.

- Agregar otros modelos clasificadores para otras enfermedades que puedan ser identificadas mediante imágenes.
- Dividir la aplicación en una versión para usuarios y otra para profesionales médicos que permita a estos comunicarse.



CAPÍTULO 7:

BIBLIOGRAFÍA



- [1] Irwin Sobel. “An Isotropic 3x3 Image Gradient Operator” (Febrero de 2014)
(Consultado: 3 de Junio de 2024)
https://www.researchgate.net/publication/239398674_An_Isotropic_3x3_Image_Gradient_Operator
- [2] Hernán Alejandro Olmí Reyes y Camilo Angel Peña Ramírez. “VISIÓN POR COMPUTADOR, SU HISTORIA Y ALGUNOS PRINCIPALES HITOS” (2021)
(Consultado: 3 de Junio de 2024)
https://revistaingenieriaaldia.ucentral.cl/rev_10/hernan_olmi.pdf
- [3] Tian-Gen Chang, Yingying Cao, Hannah J. Sfreddo, Saugato Rahman Dhruba, Se-Hoon Lee, Cristina Valero, Seong-Keun Yoo, Diego Chowell, Luc G. T. Morris, Eytan Ruppín. “LORIS robustly predicts patient outcomes with immune checkpoint blockade therapy using common clinical, pathologic and genomic features” (2024)
(Consultado: 3 de Junio de 2024)
<https://www.nature.com/articles/s43018-024-00772-7>
- [4] Página web del Instituto de Ingeniería del Conocimiento. “Análisis inteligente de imágenes médicas” (2024)
(Consultado: 3 de Junio de 2024)
<https://www.iic.uam.es/soluciones/salud/analisis-datos-salud/analisis-inteligente-imagenes-medicas>
- [5] Página web de MedlinePlus. “Diagnóstico por imágenes” (Última actualización 21 enero 2021)
(Consultado: 3 de Junio de 2024)
<https://medlineplus.gov/spanish/diagnosticimaging.html>
- [6] Marina Nikolic, Eva Tuba y Milan Tuba. “Edge detection in medical ultrasound images using adjusted Canny edge detection algorithm” (Noviembre de 2016)
(Consultado: 29 de Junio de 2024)
https://www.researchgate.net/publication/312570594_Edge_detection_in_medical_ultrasound_images_using_adjusted_Canny_edge_detection_algorithm

- [7] Panagiotis Korfiatis, Garima Suman, Nandakumar G. Patnam, Kamaxi H. Trivedi, Aashna Karbhari, Sovanlal Mukherjee, Cole Cook, Jason R. Klug, Anurima Patra, Hala Khasawneh, Naveen Rajamohan, Joel G. Fletcher, Mark J. Truty, Shounak Majumder, Candice W. Bolan, Kumar Sandrasegaran, Suresh T. Chari ,Ajit H. Goenka. “Automated Artificial Intelligence Model Trained on a Large Data Set Can Detect Pancreas Cancer on Diagnostic Computed Tomography Scans As Well As Visually Occult Preinvasive Cancer on Prediagnostic Computed Tomography Scans” (30 de Agosto de 2023) (Consultado: 3 de Junio de 2024) [https://www.gastrojournal.org/article/S0016-5085\(23\)04958-2/fulltext](https://www.gastrojournal.org/article/S0016-5085(23)04958-2/fulltext)
- [8] Kunio Doi. “Computer-Aided Diagnosis in Medical Imaging: Historical Review, Current Status and Future Potential” (8 de Marzo de 2007) (Consultado: 29 de Junio de 2024) <https://pubmed.ncbi.nlm.nih.gov/17349778/>
- [9] Página web de soporte de Apple. “About Face ID advanced technology” (Consultado: 3 de Junio de 2024) <https://support.apple.com/en-us/102381>
- [10] IBM Data and AI Team. “AI versus machine learning versus deep learning versus neural networks: What’s the difference?” (Julio 2023) (Consultado: 4 de Abril de 2024) <https://www.ibm.com/think/topics/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks>
- [11] Página web de IBM. “¿Qué es la IA fuerte?” (Consultado: 4 de Abril de 2024) <https://www.ibm.com/es-es/topics/strong-ai>
- [12] Mainichi Japan. “SoftBank's Son sets ambition to realize 'super' AI in 10 yrs” (21 de Junio de 2024) (Consultado: 29 de Junio de 2024) <https://mainichi.jp/english/articles/20240621/p2g/00m/0bu/069000c>

- [13] IBM Data and AI Team. “AI versus machine learning versus deep learning versus neural networks: What’s the difference?” (Julio 2023) (Consultado: 4 de Abril de 2024) <https://www.ibm.com/think/topics/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks>
- [14] Página web de IBM. “What is deep learning?” (Consultado: 4 de Abril de 2024) <https://www.ibm.com/topics/deep-learning>
- [15] Juan Francisco Vallalta Rueda. “Aprendizaje supervisado y no supervisado” (Consultado: 4 de Mayo de 2024) <https://healthdataminer.com/data-mining/aprendizaje-supervisado-y-no-supervisado/>
- [16] Jeremy Appleyard y Scott Yokim. “Programming Tensor Cores in CUDA 9” (17 de Octubre de 2017) (Consultado: 12 de Junio de 2024) <https://developer.nvidia.com/blog/programming-tensor-cores-cuda-9/>
- [17] Durga Malladi y Pat Lawlor. “What is an NPU? And why is it key to unlocking on-device generative AI?” (1 de Febrero de 2024) (Consultado: 12 de Junio de 2024) <https://www.qualcomm.com/news/onq/2024/02/what-is-an-npu-and-why-is-it-key-to-unlocking-on-device-generative-ai>
- [18] Stephanie Doyle. “AI 101: GPU vs. TPU vs. NPU” (2 de Agosto de 2023) (Consultado: 12 de Junio de 2024) <https://www.backblaze.com/blog/ai-101-gpu-vs-tpu-vs-npu>
- [19] Sakshi Tiwari. “Activation functions in Neural Networks” (Última actualización: 3 de Mayo de 2024) (Consultado: 28 de Junio de 2024) <https://www.geeksforgeeks.org/activation-functions-neural-networks/>

- [20] Vandit Jain. “Everything you need to know about “Activation Functions” in Deep learning models” (30 de Diciembre de 2019) (Consultado: 28 de Junio de 2024)
<https://towardsdatascience.com/everything-you-need-to-know-about-activation-functions-in-deep-learning-models-84ba9f82c253>
- [21] Gilbert Tanner. “Activation Functions” (20 de Septiembre de 2021) (Consultado: 3 de Junio de 2024)
<https://ml-explained.com/blog/activation-functions-explained>
- [22] Warren S. Mcculloch y Walter Pitts. “A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY”. (Bulletin of mathematical biophysics Volume 5, 1943) (Consultado: 8 de Abril de 2024)
- [23] Great Learning Team. “Types of Neural Networks and Definition of Neural Network” (23 de Noviembre de 2022) (Consultado: 4 de Mayo de 2024)
<https://www.mygreatlearning.com/blog/types-of-neural-networks>
- [24] Página web de Amazon AWS. “¿Qué es una GAN?” (Consultado: 6 de Mayo de 2024) <https://aws.amazon.com/es/what-is/gan/>
- [25] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin. “Attention Is All You Need” (12 de Junio de 2017) (Consultado: Mayo de 2024)
<https://arxiv.org/abs/1706.03762>
- [26] Rick Merritt. “¿Qué es un Modelo Transformer?” (19 de Abril de 2022) (Consultado: Junio de 2024)
<https://la.blogs.nvidia.com/blog/que-es-un-modelo-transformer/>

- [27] Daksh Bhatnagar. “How Neural Networks Learn using Gradient Descent” (5 de Noviembre de 2022) (Consultado: 6 de Mayo de 2024) <https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd>
- [28] Simeon Kostadinov. “Understanding Backpropagation Algorithm” (8 de Agosto de 2019) (Consultado: 6 de Mayo de 2024) <https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd>
- [29] Página web de Amazon AWS. “What is Data Augmentation?” (Consultado: 3 de Mayo de 2024) <https://aws.amazon.com/es/what-is/data-augmentation/>
- [30] Jason Brownlee. “How to Diagnose Overfitting and Underfitting of LSTM Models” (8 de Enero de 2020) (Consultado: 8 de Mayo de 2024) <https://machinelearningmastery.com/diagnose-overfitting-underfitting-lstm-models>
- [31] Rita Kurban. “Boost your Network Performance” (10 de Marzo de 2021) (Consultado: 8 de Mayo de 2024) <https://towardsdatascience.com/boost-your-network-performance-cc0a2a95c5ef>
- [32] Jason Brownlee. “How to Avoid Overfitting in Deep Learning Neural Networks” (6 de Agosto de 2019) (Consultado: 8 de Mayo de 2024) <https://machinelearningmastery.com/introduction-to-regularization-to-reduce-overfitting-and-improve-generalization-error/>
- [33] Repositorio de Google en Github. “XNNPACK” (Consultado: 11 de Mayo de 2024) <https://github.com/google/XNNPACK>
- [34] Repositorio de Pytorch en Github. “QNNPACK” (Consultado: 11 de Mayo de 2024) <https://github.com/pytorch/QNNPACK>

- [35] Repositorio de Pytorch en Github. “FBGEMM” (Consultado: 11 de Mayo de 2024) <https://github.com/pytorch/FBGEMM>
- [36] Página web de PyTorch. “torch.utils.mobile_optimizer” (Consultado: 11 de Mayo de 2024) https://pytorch.org/docs/stable/mobile_optimizer.html
- [37] Pytorch. “Quantization Recipe” (Consultado: 12 de Mayo de 2024) <https://pytorch.org/tutorials/recipes/quantization.html>
- [38] Página web de Tensor Flow. “Entrenamiento consciente de la cuantificación” (Consultado: 12 de Mayo de 2024) https://www.tensorflow.org/model_optimization/guide/quantization/training?hl=es
- [39] Begoña Gómez Pujante. “Redes Convolucionales. Aplicación a la clasificación de imágenes médicas” (Junio de 2023) (Consultado: 2 de Abril de 2024) <https://dspace.umh.es/handle/11000/30233>
- [40] Manuel López Martínez. “Seek Health: Sistema inteligente basado en deep learning para la ayuda en el diagnóstico médico” (Septiembre de 2021) (Consultado: 15 de Abril de 2024) <https://dspace.umh.es/handle/11000/26564>
- [41] Ángel Fragua Baeza. “Ejecución de redes neuronales en móviles Android con aceleración hardware mediante Keras y Tensorflow Lite” (Junio de 2021) (Consultado: 7 de Mayo de 2024) <https://repositorio.uam.es/handle/10486/698211>
- [42] Página web de AviSynth. “Nnedi3” (Consultado: 7 de Junio de 2024) <http://avisynth.nl/index.php/Nnedi3#Filters>
- [43] Página web de waifu2x. “waifu2x” (Consultado: 7 de Junio de 2024) <https://www.waifu2x.net/index.es.html>

- [44] Repositorio de bloc97 en Github. “Anime4K” (2019) (Consultado: 7 de Junio de 2024) <https://github.com/bloc97/Anime4K>
- [45] Repositorio de hzwer en Github. “Real-Time Intermediate Flow Estimation for Video Frame Interpolation” (2020) (Consultado: 7 de Junio de 2024) <https://github.com/hzwer/ECCV2022-RIFE>
- [46] Página web de Nvidia developer. “Getting Started with DLSS” (Consultado: 7 de Junio de 2024) <https://developer.nvidia.com/rtx/dlss/get-started>
- [47] Página web de Python. “About python” (Consultado: 16 de Abril de 2024) <https://www.python.org/about/>
- [48] Página web de PIP (Consultado: 22 de Abril de 2024) <https://pypi.org/>
- [49] Página web de PyTorch. “PyTorch GET STARTED” (Consultado: 16 de Abril de 2024) <https://pytorch.org/>
- [50] Gaudenz Boesch. “Pytorch vs Tensorflow: A Head-to-Head Comparison” (Consultado: 11 de Mayo de 2024) <https://viso.ai/deep-learning/pytorch-vs-tensorflow/>
- [51] Página web de CUDA. (Consultado: 11 de Mayo de 2024) <https://developer.nvidia.com/cuda-zone>
- [52] Página web de Matplotlib. (Consultado: 17 de Mayo de 2024) <https://matplotlib.org/>
- [53] Página web de NumPy. (Consultado: 11 de Mayo de 2024) <https://numpy.org/>
- [54] Página web de NumPy. (Consultado: 14 de Mayo de 2024) <https://python-pillow.org/>

- [55] Página web de Sublime Text 3. (Consultado: 9 de Mayo de 2024)
<https://www.sublimetext.com/3>
- [56] Página web de Android Studio. (Consultado: 22 de Mayo de 2024)
<https://developer.android.com/studio?hl=es-419>
- [57] Página web de Java. (Consultado: 22 de Mayo de 2024)
<https://www.java.com/es/>
- [58] Página web de Android Developers. “Descripción general de Camera2”
(Consultado: 2 de Mayo de 2024)
<https://developer.android.com/media/camera/camera2?hl=es-419>
- [59] Página web de Android Developers. “Descripción general de CameraX”
(Consultado: 2 de Mayo de 2024)
<https://developer.android.com/media/camera/camerax?hl=es-419>
- [60] Página web de Maven. “Descripción general de CameraX” (Consultado: 2 de Mayo de 2024) <https://maven.apache.org/>
- [61] Página web de Sonatype Nexus Repository. (Consultado: 2 de Mayo de 2024)
<https://www.sonatype.com/>
- [62] JCenter Repository en la web de MVNREPOSITORY. (Consultado: 2 de Junio de 2024) <https://mvnrepository.com/repos/jcenter>
- [63] Página web de PyTorch. “FBGEMM and FBGEMM_GPU Documentation Homepage” (Consultado: 24 de Mayo de 2024) <https://pytorch.org/FBGEMM/>
- [64] BHAVESH MITTAL. “Melanoma Cancer Image Dataset” (Ultima actualización: Febrero de 2024) (Consultado: 12 de Abril de 2024)
<https://www.kaggle.com/datasets/bhaveshmittal/melanoma-cancer-dataset/data>

- [65] Youming Wang, Zhao Xiao, Gongqing Cao. “A convolutional neural network method based on Adam optimizer with power-exponential learning rate for bearing fault diagnosis” (30 de Junio de 2022) (Consultado: 14 de Abril de 2024) <https://www.extrica.com/article/22271>
- [66] Nghi Huynh. “Understanding Loss Functions for Classification” (1 de Marzo de 2023) (Consultado: 14 de Abril de 2024) https://medium.com/@nghihuynh_37300/understanding-loss-functions-for-classification-81c19ee72c2a
- [67] Página web de PyTorch. “Models and pre-trained weights” (Consultado: 14 de Abril de 2024) <https://pytorch.org/vision/stable/models.html>
- [68] Jefkine. “Backpropagation In Convolutional Neural Networks” (5 de Septiembre de 2016) (Consultado: 10 de Abril de 2024) <https://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/>
- [69] Swarnima Pandey. “How to choose the size of the convolution filter or Kernel size for CNN?” (23 de Junio de 2020) (Consultado: 10 de Abril de 2024) <https://medium.com/analytics-vidhya/how-to-choose-the-size-of-the-convolution-filter-or-kernel-size-for-cnn-86a55a1e2d15>
- [70] Sabyasachi Sahoo. “Deciding optimal kernel size for CNN” (19 de Agosto de 2018) (Consultado: 10 de Abril de 2024) <https://towardsdatascience.com/deciding-optimal-filter-size-for-cnns-d6f7b56f9363>
- [71] Sandra Navarro, Karen Simonyan, Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition” (4 de Septiembre de 2014) (Consultado: 26 de Junio de 2024) <https://arxiv.org/abs/1409.1556>
- [72] Mingxing Tan, Quoc V. Le. “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks” (28 de Mayo de 2019) (Consultado: 29 de Mayo de 2019) <https://arxiv.org/abs/1905.11946>

- [73] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam. “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications” (17 de Abril de 2017) (Consultado: 26 de Junio de 2024) <https://arxiv.org/abs/1704.04861>
- [74] Repositorio de Pytorch en Github. “PyTorch Android Examples” (2019) (Consultado: 2 de Mayo de 2024) <https://github.com/pytorch/android-demo-app/tree/master>
- [75] Página web de Mathworks. “Introducción a la convolución” (Consultado: Abril de 2024) <https://es.mathworks.com/discovery/convolution.html>
- [76] Google for developers. “Glosario sobre aprendizaje automático” (Consultado: Abril de 2024) <https://developers.google.com/machine-learning/glossary?hl=es-419>
- [77] Jan Salomon Cramer. “The origins and development of the logit model” (Agosto de 2003) (Consultado: Abril de 2024) https://www.cambridge.org/resources/0521815886/1208_default.pdf
- [78] Página web de Pytorch. “PyTorch documentation” (Consultado: Abril de 2024) <https://pytorch.org/docs/stable/index.html>
- [79] Página web de tutoriales de Pytorch. “(beta) Efficient mobile interpreter in Android and iOS” (Consultado: Abril de 2024) https://pytorch.org/tutorials/recipes/mobile_interpreter.html
- [80] Página web de Pytorch. “TORCHVISION.MODELS” (Consultado: Abril de 2024) <https://pytorch.org/vision/0.9/models.html>

ANEXO I.

GLOSARIO



I.I Glosario general

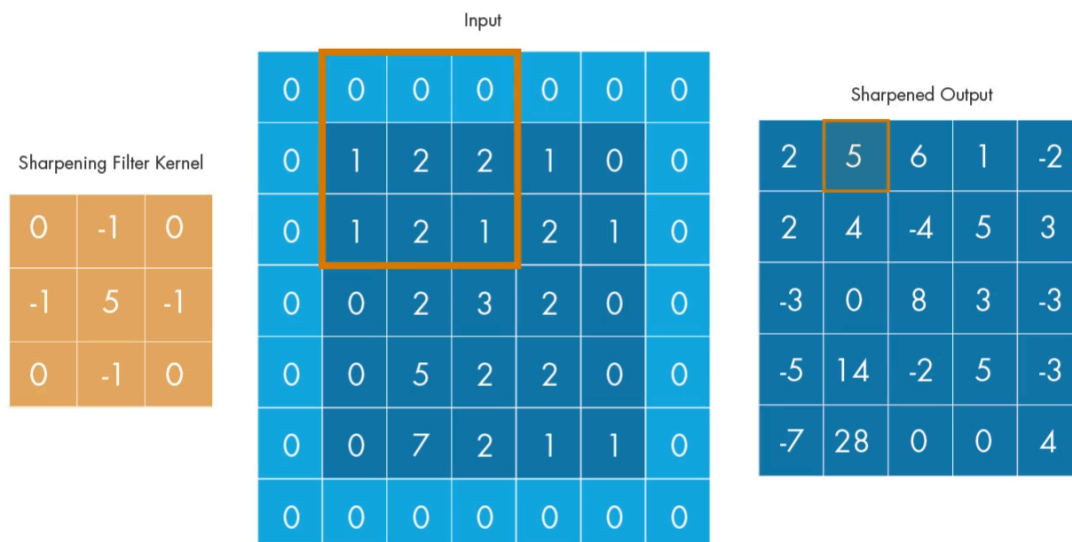
El desarrollo de la red neuronal requiere conocimientos previos acerca de múltiples conceptos con los que podríamos no estar familiarizados. Para facilitar la comprensión de estos, los más relevantes se muestran a continuación.

1. ASIC

Circuito integrado hecho para un uso en particular, en vez de ser para propósitos de uso general.

2. Convolución

Una convolución es una operación matemática que consiste en combinar dos funciones al “deslizar” una función sobre la otra, multiplicar los puntos de intersección y sumar los valores de los productos para crear una nueva función. Se utiliza principalmente en imágenes, pero se puede utilizar en vectores, matrices bidimensionales, matrices tridimensionales, etc.[75] Una de las funciones es un dato de entrada y la otra tiene un tamaño llamado tamaño de kernel. En el caso de imágenes, el tamaño de kernel suele ser cuadrado. Por ejemplo, un filtro de nitidez que aplica una convolución de tamaño 3x3 (píxeles) sobre una imagen y da como resultado una imagen de igual resolución a la imagen original pero con mayor nitidez, como se muestra a continuación.



3. CUDA

Es una API para ejecutar software paralelamente en los núcleos CUDA de GPUs de Nvidia. Es una extensión del lenguaje de programación C que añade la capacidad de especificar paralelismo a nivel de subprocesos y especificar operaciones específicas del dispositivo de la GPU.

4. Dataset

Es un conjunto de datos, en el caso de aprendizaje automático se trata de los datos utilizados para entrenar la red neuronal. Puesto que es altamente recomendable poder evaluar la red neuronal con datos con los que no ha sido entrenada, el conjunto total de datos se divide entre datos de entrenamiento y datos de validación.

Suelen dividirse con una relación entre 70:30 y 80:20 (entrenamiento:validación).

5. Época

Una época o epoch es una pasada completa del modelo por el conjunto de datos de entrenamiento.

6. Exactitud

La exactitud o accuracy es:

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

Donde:

- VP es la cantidad de verdaderos positivos (predicciones correctas).
- VN es la cantidad de verdaderos negativos (predicciones correctas).
- FP es la cantidad de falsos positivos (predicciones incorrectas).
- FN es la cantidad de falsos negativos (predicciones incorrectas).

7. FPGA

Un matriz de puerta programable en campo (Field-programmable gate array) es un dispositivo semiconductor que contiene componentes lógicos programables.

8. Iteración

Una iteración se produce cada vez que un modelo actualiza el valor de sus pesos y bias durante el entrenamiento. El número de iteraciones por época depende del tamaño del lote.

9. Logit

Los modelos de clasificación generan predicciones $(-\infty, \infty)$ que no están normalizadas $(0, 1)$. [76] [78]

10. Lote

Es el conjunto de datos de entrenamiento utilizados para una iteración durante el entrenamiento. El tamaño de lote o batch size es el número de datos que tiene un lote.

11. Modelo

Un modelo es una construcción matemática que toma unos datos de entrada como ejemplo e infiere una predicción de salida. Puede tratarse de un modelo de red neuronal, un modelo regresión lineal, un modelo de árbol de decisión, etc.

12. Matriz de confusión

Es una representación visual de los aciertos y errores del modelo. En un eje se encuentran los valores esperados y en el otro los valores de predicción. El tamaño de la matriz depende del número de etiquetas. Cuando la exactitud del modelo es del 100%, todos los elementos de la matriz son cero excepto la diagonal principal, esta diagonal representa los verdaderos positivos y verdaderos negativos. El resto de valores son falsos positivos y falsos negativos.

13. Optimizador

Es una implementación de un algoritmo de descenso de gradiente.

14. Pérdida

Es una medida que mide la diferencia entre el valor de predicción y el valor de la etiqueta. Su valor se obtiene con una función de pérdida.

15. Precisión

Es una métrica para los modelos de clasificación. Su valor se obtiene dividiendo el número de predicciones positivas correctas entre el número de predicciones totales (verdaderos positivos y falsos positivos).

16. Regularización de abandono

La regularización de abandono elimina una selección aleatoria de un número fijo de unidades en una capa de red.

I.II. Glosario de Pytorch

Estas son las principales funciones y clases de Pytorch utilizadas para la creación de la red neuronal.

1. Model

Es la clase que contiene toda la información de la red neuronal, incluyendo las capas y los pesos. Su clase base es `nn.Module` y está formado por módulos. Puede tratarse de un modelo (red neuronal) implementado manualmente en Python o un modelo pre-entrenado de una librería como Torchvision.[79] [81]

2. Capa “Conv2d”

Aplica una convolución 2D sobre una señal de entrada compuesta de varios planos de entrada. [79]

Parámetros obligatorios:

- `in_channels`: número de canales en la imagen de entrada.
- `out_channels`: número de canales producidos por la convolución.

- `kernel_size`: tamaño del kernel convolutivo.

3. Capa “Linear”

Aplica una transformación lineal a los datos entrantes. Es una capa completamente conectada, por tanto todos los nodos de entrada están conectados a todos los nodos de salida.[79]

Parámetros obligatorios:

- `in_features`: tamaño de cada muestra de entrada.
- `out_features`: tamaño de cada muestra de salida.

4. Max-pooling “MaxPool2d”

Aplica una agrupación promedio 2D sobre una señal de entrada compuesta de varios planos de entrada.[79]

Parámetros obligatorios:

- `kernel_size`: el tamaño de la ventana para tomar un máximo.
- `stride`: el avance de la ventana. El valor por defecto es `kernel_size`

5. Avg-pooling “AvgPool2d”

Aplica una agrupación máxima 2D sobre una señal de entrada compuesta de varios planos de entrada.[79]

Parámetros obligatorios:

- `kernel_size`: el tamaño de la ventana para tomar un máximo.
- `stride`: el avance de la ventana. El valor por defecto es `kernel_size`

6. Funcion de activación “LeakyReLU”

Aplica la función LeakyReLU por elementos.[79]

7. Funcion de activación “ReLU”

Aplica la función unitaria lineal rectificadora por elementos.[79]

8. Normalización “BatchNorm2d”

Aplica la normalización por lotes sobre una entrada 4D.

4D es un mini lote de entradas 2D con dimensión de canal adicional.[79]

Parámetros

- `in_features(int)` – tamaño de entrada esperado.

9. Abandono “Dropout2D”

Pone a cero aleatoriamente canales completos.[79]

Parámetros

- `in_features(int)` – tamaño de entrada esperado.

10. Función de pérdida “CrossEntropyLoss”

Este criterio calcula la pérdida de entropía cruzada entre los logits de entrada y el objetivo.[79]

11. Algoritmo optimizador “Adam”

Implementa el algoritmo con tasa de aprendizaje adaptativa Adam.

Parámetros:

- `params`: parámetros para optimizar.
- `lr (opcional)`: tasa de aprendizaje (predeterminado: $1e-3$). [79]

12. Panificador de la tasa de aprendizaje

“ReduceLROnPlateau”

Reduce la tasa de aprendizaje del optimizador cuando una métrica ha dejado de mejorar.

Los modelos a menudo se benefician al reducir la tasa de aprendizaje en un factor de 2 a 10 una vez que el aprendizaje se estanca. Este programador lee una cantidad de métricas y, si no se observa ninguna mejora durante un número de épocas de "paciencia", la tasa de aprendizaje se reduce.

Parámetros:

- optimizador: optimizador.
- factor: factor por el cual se reducirá la tasa de aprendizaje. Predeterminado: 0,1.
- paciencia: la cantidad de épocas permitidas sin mejoras después de las cuales se reducirá la tasa de aprendizaje. Predeterminado: 10.
- min_lr: un escalar o una lista de escalares. Un límite inferior en la tasa de aprendizaje de todos los grupos de parámetros o de cada grupo respectivamente. Predeterminado: 0. [79]

13. “torch.jit.script”

Sirve para crear una instancia de tipo ScriptModule de un modelo. [79]

Parámetro:

- objeto: modelo, tipo de clase, diccionario o lista a compilar..

14. “optimize_for_mobile”

Ejecuta una lista de optimizaciones a un módulo. [79]

Parámetros:

- script_module: una instancia de tipo ScriptModule.

15. “_save_for_lite_interpreter”

Sirve para guardar un modelo para la librería “pytorch_android_lite” de Android. [80]

Utilización:

```
modelo._save_for_lite_interpreter("archivo_modelo.pt").
```

ANEXO II. Interfaz de la aplicación Android



En este anexo se muestran las distintas pantallas y el funcionamiento de la aplicación desarrollada.

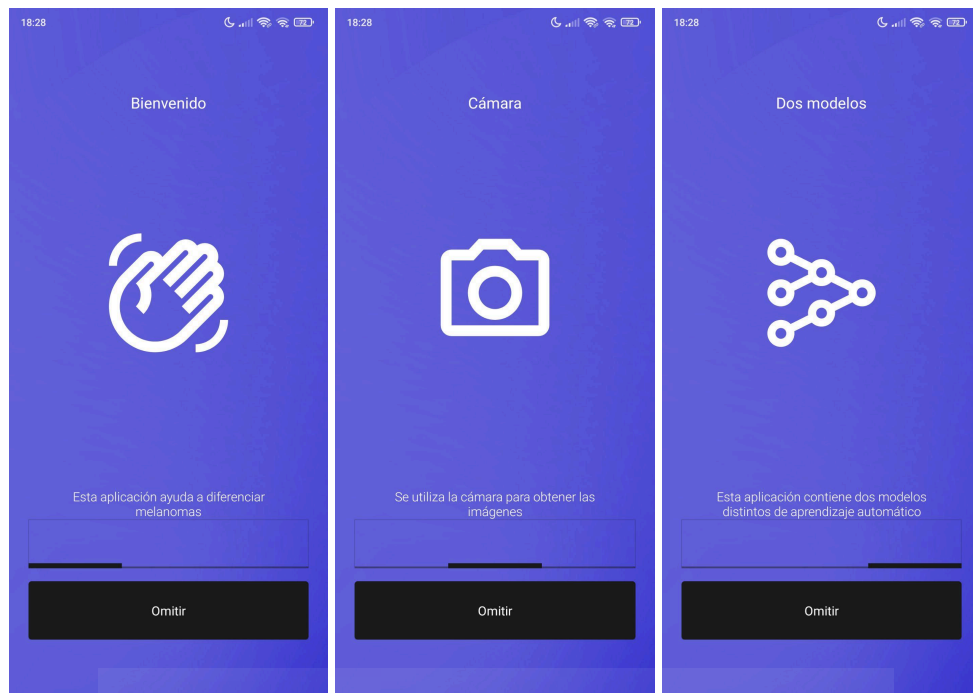


Ilustración 41. Página principal de la aplicación

Al iniciar la aplicación se muestra la página principal con una breve información acerca de la aplicación, como se muestra en la Ilustración 41.

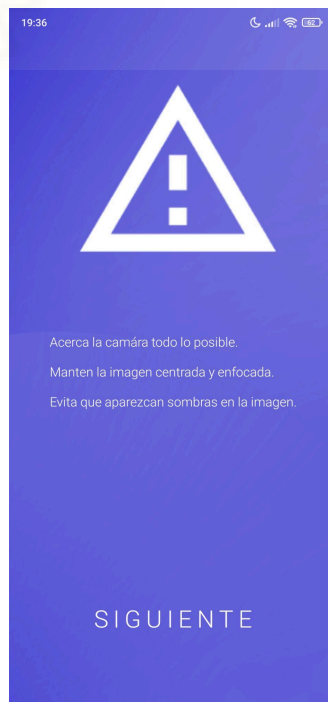


Ilustración 42. Información sobre el uso de la cámara

Al pulsar omitir se muestra la pantalla de la Ilustración 42 con un aviso del uso de la cámara y consejos para utilizar correctamente la cámara con la aplicación.

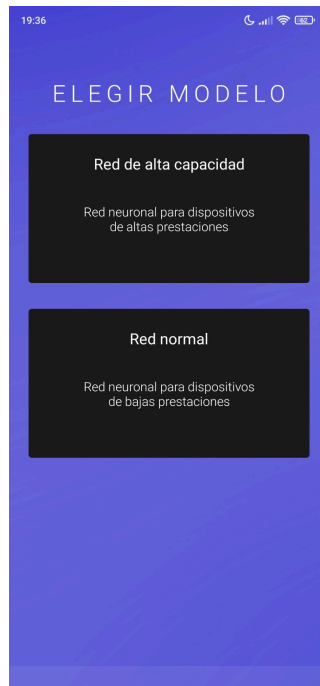


Ilustración 43. Elegir modelo

Al continuar la aplicación da a elegir entre dos modelos para realizar la ejecución como se muestra en la Ilustración 43. Al elegir uno se abre la cámara.

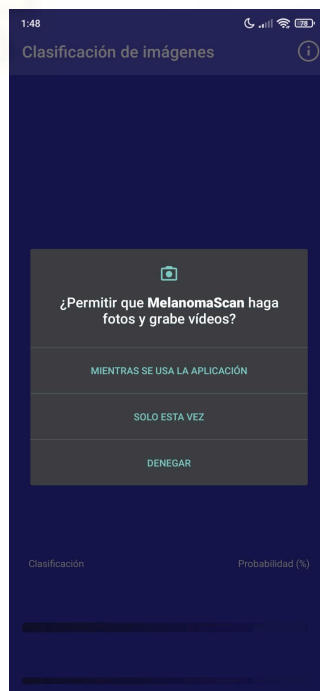


Ilustración 44. Solicitud de permiso

La primera vez que se utiliza la cámara saldrá un mensaje igual o similar al que se muestra en la Ilustración 44, solicitando permiso para permitir el uso de la cámara.

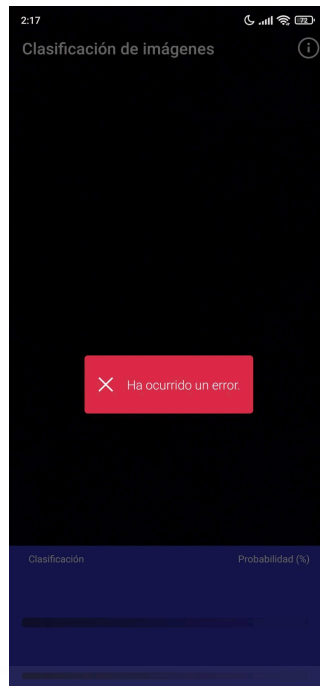


Ilustración 45. Error

En caso de producirse un error durante la ejecución de la red neuronal, se mostrará un mensaje como el de la Ilustración 45.

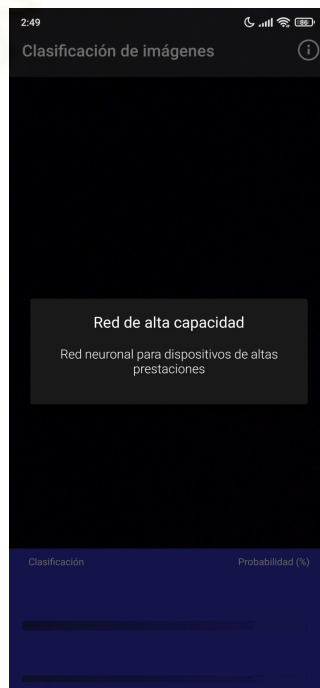



Ilustración 46. Información de la red neuronal

Al pulsar el símbolo  de la esquina superior derecha, se muestra la información acerca de la red neuronal que se está utilizando, como se muestra en la Ilustración 46.

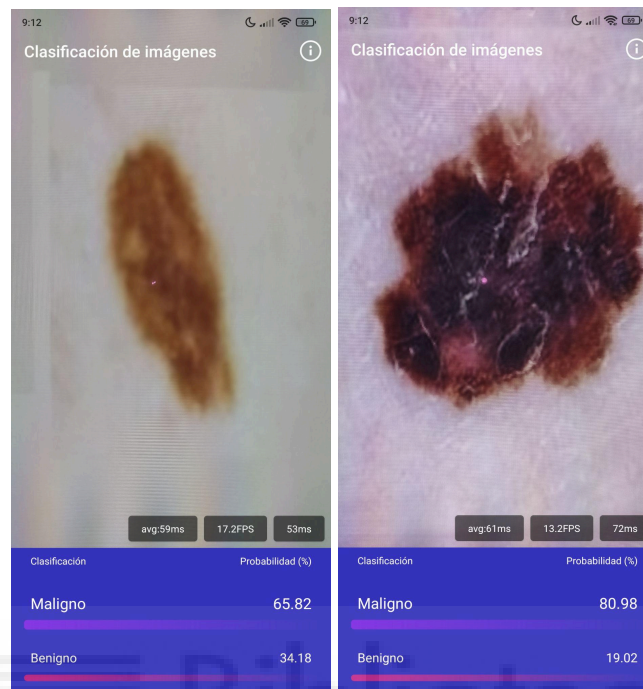


Ilustración 46. Ejemplos de clasificación

La Ilustración 46 muestra dos ejemplos de la aplicación clasificando dos imágenes, mostrando en la parte inferior el porcentaje de certeza que tiene en la clasificación. El primero es un caso benigno (imagen 6300 del conjunto de datos de validación) y el segundo uno maligno (imagen 30 del conjunto de datos de entrenamiento).