

The logo consists of the lowercase letters "viu" in white, centered within a solid orange circle.

viu

**Universidad
Internacional
de Valencia**

Uso de modelos de visión por computador basados en Transformers en tiempo real en dispositivos móviles

Titulación:
Máster en Inteligencia
Artificial
Curso académico
2023 – 2024

Alumno/a: La Casa Nieto,
José Jesús
D.N.I: 77249230G

Director/a de TFM: Enrique
Mas Candela

Convocatoria:
Segunda convocatoria

Dedicatoria

*Dedicado a mi familia y amigos,
en especial a mis padres y mi abuelo Indalecio,
por todo, por ser mi ejemplo de vida.*

Agradecimientos

*A mi tutor, Enrique Mas Candela,
por su implicación y confianza en el proyecto*

*Al profesorado y compañeros del máster que han ayudado
en mi formación durante este curso.*

Resumen

Este trabajo de fin de máster se centra en la implementación y optimización de modelos de visión por computador utilizando arquitecturas basadas en Transformers, con el objetivo de aplicarlos en dispositivos móviles. A lo largo del proyecto, se han explorado dos enfoques principales: la creación de modelos desde cero y la aplicación de transfer learning utilizando modelos preentrenados. Se han implementado técnicas avanzadas de optimización, como pruning, cuantización y binarización, para reducir la complejidad computacional y mejorar la eficiencia de los modelos, facilitando así su despliegue en plataformas con recursos limitados.

Además, se ha realizado una profunda revisión del estado del arte en el ámbito de la visión por computador, lo que ha permitido identificar las principales tendencias y desafíos en la aplicación de modelos basados en Transformers. Se ha prestado especial atención a las implicaciones de utilizar estas arquitecturas en dispositivos móviles, así como a las limitaciones inherentes al proceso de entrenamiento y optimización, incluyendo las restricciones de memoria en GPUs. Los resultados obtenidos demuestran que, a pesar de las dificultades, es posible aplicar con éxito técnicas de optimización y conversión a formatos móviles, logrando así un avance significativo en la implementación de modelos de visión por computador en dispositivos de menor capacidad.

Por último, se presentan propuestas para trabajos futuros que buscan profundizar en la investigación y optimización de estos modelos. Se sugiere investigar arquitecturas híbridas y adaptativas que puedan ajustarse a diferentes condiciones de operación, así como realizar evaluaciones prácticas en entornos del mundo real. La exploración de aplicaciones específicas en campos emergentes, como la salud y la seguridad, también se propone como un área de investigación que podría beneficiar significativamente del desarrollo de modelos eficientes y precisos en dispositivos móviles.

Abstract

This master's thesis focuses on the implementation and optimization of computer vision models using Transformer-based architectures, with the goal of applying them on mobile devices. Throughout the project, two main approaches have been explored: building models from scratch and applying transfer learning using pre-trained models. Advanced optimization techniques such as pruning, quantization, and binarization have been implemented to reduce computational complexity and enhance model efficiency, thereby facilitating their deployment on resource-constrained platforms.

Additionally, an in-depth review of the state of the art in computer vision has been conducted, allowing for the identification of key trends and challenges in the application of Transformer-based models. Special attention has been paid to the implications of using these architectures on mobile devices, as well as the inherent limitations in the training and optimization processes, including memory constraints on GPUs. The results obtained demonstrate that, despite the difficulties, it is possible to successfully apply optimization techniques and convert models to mobile-friendly formats, achieving significant advancements in deploying computer vision models on lower-capacity devices.

Finally, proposals for future work are presented, aimed at furthering research and optimization of these models. It is suggested to investigate hybrid and adaptive architectures that can adjust to varying operating conditions, as well as to conduct practical evaluations in real-world environments. The exploration of specific applications in emerging fields, such as healthcare and security, is also proposed as an area of research that could greatly benefit from the development of efficient and accurate models on mobile devices.

Índice general

Dedicatoria	I
Agradecimientos	III
Resumen	V
Abstract	VII
Índice general	IX
Índice de figuras	XIII
1. Introducción	1
1.1. Contexto y Relevancia	1
1.2. Objetivos del Trabajo	2
1.3. Motivación y Justificación	3
1.4. Estructura del Trabajo	3
2. Estado del arte	5
2.1. Antecedentes generales	6
2.1.1. Visión por Computador	6
2.1.2. Modelos basados en Transformers	8
2.2. Revisión de la literatura	10
2.2.1. Modelos de Visión por Computador tradicionales	10
2.2.2. Transformers en Visión por Computador	11
2.2.3. Aplicaciones en tiempo real	12
2.3. Comparación y discusión	14
2.3.1. Comparación de Enfoques	14
2.3.2. Ventajas y Desventajas	14
2.4. Conclusiones del estado del arte	17

3. Marco teórico	19
3.1. Conceptualización Avanzada y Definiciones	19
3.1.1. Visión por Computador en Dispositivos Móviles	19
3.1.2. Transformers en Visión por Computador: Principios y Funcionalidades	22
3.2. Modelos y Algoritmos Relevantes	23
3.2.1. Vision Transformers (ViT)	23
3.2.2. Transformers Híbridos	24
3.2.3. Optimización para Dispositivos Móviles	25
3.2.4. Modelos Ligados a Aplicaciones Específicas (e.g. YO-LO, DETR)	26
3.3. Teorías y Principios Subyacentes	27
3.3.1. Teoría de la Atención y su Aplicación en Transformers	28
3.3.2. Aprendizaje Profundo en Dispositivos Móviles	29
3.3.3. Comprensión y Cuantización de Modelos	30
3.3.4. Pruning y Técnicas de Reducción de Parámetros	31
3.4. Estado actual de la tecnología	33
4. Materiales	37
4.1. Hardware	37
4.1.1. Estación de Trabajo	37
4.2. Software	38
4.2.1. Framework de Deep Learning	38
4.2.2. Herramientas de Optimización	38
4.2.3. Entorno de Desarrollo	39
4.3. Conjuntos de Datos	39
4.3.1. Conjunto de Datos Principal	39
4.3.2. Conjunto de Datos Adicional	40
5. Métodos	41
5.1. Implementación de Modelos Basados en Transformers	41
5.1.1. Selección de Modelos	41
5.1.2. Preprocesamiento de Datos	41
5.1.3. Entrenamiento de los Modelos	42
5.2. Optimización para Dispositivos Móviles	42
5.2.1. Conversión a Formato Móvil	43
5.2.2. Cuantización	43
5.2.3. Pruning	43
5.2.4. Modelos Binarios	43

<i>ÍNDICE GENERAL</i>	XI
6. Resultados	45
6.1. Arquitectura desde cero (From Scratch)	45
6.1.1. Descripción del proceso	46
6.1.2. Entrenamiento del modelo	57
6.2. Arquitectura con modelos preentrenados (Transfer Learning) . .	60
6.2.1. Aplicación de Transfer Learning	60
6.3. Optimización de modelos para dispositivos móviles	61
6.3.1. Aplicación de las técnicas de optimización	62
6.3.2. Introducción del modelo optimizado en un dispositivo móvil	69
7. Conclusiones y trabajos futuros	73
7.1. Resumen de los principales hallazgos	73
7.2. Implicaciones del uso de Vision Transformer en dispositivos móviles	73
7.3. Limitaciones del trabajo	74
7.4. Propuestas de trabajos futuros	74
Bibliografía	77

Índice de figuras

2.1. Detector de bordes de Canny [28].	6
2.2. Arquitectura Redes Neuronales Convolucionales (CNNs) [29]. . .	7
2.3. Arquitectura modelo basado en Transformer [30].	9
2.4. Arquitectura VGG [31].	10
2.5. Arquitectura Vision Transformer (ViT) [32].	11
2.6. Arquitectura EfficientNet-B0 [33].	12
2.7. Gráfica latencia-precisión modelos visión por computador [34].	13
3.1. Arquitectura Convolutional Vision Transformer (CvT) [35]. . .	24
3.2. Detección de objetos con YOLO [36].	27
3.3. Arquitectura MobileNet [37].	29
3.4. Arquitectura BitNet [38].	32
6.1. Vista preliminar del conjunto de datos SIDD Small Dataset. .	46
6.2. Dimensiones de los tensores para el entrenamiento.	47
6.3. Dimensiones de las imágenes aleatorias para su visualización. .	47
6.4. Visualización de imágenes GT y ruido.	48
6.5. Visualización de imagen con ruido aleatoria.	48
6.6. Sección superior de imagen con ruido aleatoria.	49
6.7. Sección superior de imagen con ruido aleatoria separada en parches.	50
6.8. Imagen aleatoria completa con ruido separada en parches. .	50
6.9. Mapa de características de la imagen aleatoria.	52
6.10. Mapa de características aplanado.	53
6.11. Resumen de la arquitectura del bloque TransformerEncoder- Block.	55
6.12. Resumen de la arquitectura completa del modelo ViT from scratch.	56
6.13. Arquitectura completa del modelo ViT from scratch con decoder.	57
6.14. Error durante el entrenamiento del modelo por falta de capa- cidad GPU.	59

6.15. Arquitectura completa del modelo ViT transfer learning con decoder.	61
6.16. Arquitectura del modelo ViT con pruning aplicado.	63
6.17. Arquitectura del modelo ViT con cuantización aplicada.	65
6.18. Arquitectura del modelo binario ViT.	68
6.19. Modelos para implementación en dispositivos móviles.	71

“El único error real es aquel del que no aprendemos nada.” - Henry Ford

Capítulo 1

Introducción

1.1. Contexto y Relevancia

En la última década, la visión por computador ha emergido como una de las disciplinas más influyentes dentro del campo de la inteligencia artificial (IA). Esta tecnología se centra en dotar a las máquinas de la capacidad para interpretar y entender el contenido visual del mundo de manera similar a como lo hace el ser humano. Desde la detección de objetos y el reconocimiento facial hasta la segmentación semántica y el seguimiento de objetos, las aplicaciones de visión por computador son vastas y abarcan diversos sectores, incluyendo la seguridad, la automoción, la medicina y el entretenimiento.

La revolución en visión por computador ha sido impulsada por avances significativos en el aprendizaje profundo. Las redes neuronales convolucionales (CNN) han dominado durante mucho tiempo el campo, proporcionando mejoras incrementales en tareas de procesamiento de imágenes gracias a su capacidad para aprender representaciones jerárquicas de los datos visuales. Sin embargo, a medida que los requisitos de precisión y eficiencia aumentan, la comunidad de investigación ha comenzado a explorar arquitecturas alternativas que puedan superar las limitaciones inherentes a las CNN tradicionales.

En este contexto, los Transformers, originalmente diseñados para tareas de procesamiento de lenguaje natural, han emergido como una solución prometedora. Su capacidad para modelar relaciones de largo alcance y su flexibilidad en la arquitectura han demostrado ser valiosas para el análisis de datos visuales. Sin embargo, su aplicación en dispositivos móviles, donde las restricciones de hardware y energía son más severas, presenta desafíos únicos que merecen una investigación detallada.

En el capítulo de resultados, se presentan los dos enfoques principales

CAPÍTULO 1. INTRODUCCIÓN

utilizados para el desarrollo de modelos de Vision Transformers aplicados a dispositivos móviles. Primero, se detalla la creación de una arquitectura desde cero, explicando el proceso completo de diseño y entrenamiento del modelo. Posteriormente, se analiza la implementación de un modelo preentrenado mediante técnicas de transfer learning. Además, se describen las técnicas avanzadas de optimización implementadas, tales como pruning, cuantización y binarización, que han permitido reducir la complejidad de los modelos y prepararlos para su despliegue eficiente en dispositivos móviles. Se incluye un análisis del rendimiento de los modelos optimizados en dichas plataformas, lo que proporciona una visión integral de los beneficios y desafíos en este contexto.

Finalmente, en las conclusiones, se destacan los hallazgos clave de la investigación, subrayando las implicaciones prácticas de los Vision Transformers en entornos móviles, así como las limitaciones encontradas, especialmente en relación con las restricciones de memoria y computación de las GPUs utilizadas. El trabajo también ofrece propuestas para investigaciones futuras, sugiriendo la exploración de arquitecturas híbridas y adaptativas que se ajusten mejor a las limitaciones y variabilidad de los dispositivos móviles. Además, se invita a realizar pruebas en escenarios del mundo real para evaluar de forma más precisa el impacto y la viabilidad de los modelos optimizados en aplicaciones prácticas como la salud y la seguridad.

1.2. Objetivos del Trabajo

El principal objetivo de este Trabajo de Fin de Máster es examinar y optimizar el uso de modelos de visión por computador basados en Transformers para aplicaciones en tiempo real en dispositivos móviles. Este trabajo se centra en:

1. **Análisis de modelos basados en Transformers:** Evaluar cómo los Transformers, que han mostrado un rendimiento destacado en tareas de visión por computador, pueden ser adaptados para su uso en dispositivos con recursos limitados.
2. **Optimización para dispositivos móviles:** Investigar técnicas y estrategias para mejorar la eficiencia de los modelos Transformer en términos de consumo de memoria y procesamiento, permitiendo su ejecución fluida en plataformas móviles.
3. **Implementación de técnicas avanzadas de optimización:** Aplicar técnicas como pruning, cuantización y binarización para reducir la

1.3. MOTIVACIÓN Y JUSTIFICACIÓN

complejidad computacional de los modelos, facilitando su adaptación a las restricciones de hardware de los dispositivos móviles.

4. **Despliegue de modelos en plataformas móviles:** Convertir los modelos optimizados a formatos compatibles con dispositivos móviles, como TorchScript y ONNX, y evaluar su rendimiento práctico en términos de velocidad de ejecución y eficiencia energética.

1.3. Motivación y Justificación

La integración de capacidades avanzadas de visión por computador en dispositivos móviles tiene el potencial de transformar numerosas áreas de la vida cotidiana. Desde aplicaciones en realidad aumentada y aumentos en la interacción usuario-dispositivo, hasta sistemas de asistencia y seguridad en entornos móviles, la demanda de soluciones rápidas, precisas y eficientes es cada vez mayor.

Los modelos basados en Transformers representan una evolución significativa respecto a los métodos convencionales. Su capacidad para capturar dependencias globales y su flexibilidad en la configuración de la arquitectura ofrecen ventajas potenciales en tareas complejas de visión por computador. Sin embargo, la implementación efectiva en dispositivos móviles requiere una adaptación cuidadosa para abordar los desafíos de recursos limitados.

El trabajo propuesto no solo tiene relevancia académica al contribuir a la comprensión de cómo los Transformers pueden ser aplicados en un nuevo contexto, sino también un impacto práctico al ofrecer soluciones que podrían mejorar la eficiencia y funcionalidad de las aplicaciones móviles en el mundo real. Esta investigación es relevante tanto para el desarrollo de nuevas tecnologías como para la mejora de las existentes, proporcionando una base para futuras innovaciones en el campo.

1.4. Estructura del Trabajo

El presente trabajo está estructurado en varias secciones que abordan los aspectos clave del uso de Transformers en visión por computador en dispositivos móviles. En el **Estado del Arte** (Capítulo 2), se revisarán los antecedentes generales, la literatura existente sobre modelos tradicionales y basados en Transformers, así como sus aplicaciones en tiempo real. El **Marco teórico** (Capítulo 3) proporcionará una base sólida en los conceptos y teorías fundamentales, incluyendo los modelos y algoritmos relevantes, la teoría de la atención y el estado actual de la tecnología. En las secciones de **Materiales**

CAPÍTULO 1. INTRODUCCIÓN

(Capítulo 4) y **Métodos** (Capítulo 5), se detallarán los recursos y técnicas empleadas en el estudio, mientras que los **Resultados** (Capítulo 6) presentarán los hallazgos clave del análisis y la experimentación. Por último, en el capítulo **Conclusiones y trabajos futuros** (Capítulo 7) se valoran una serie de conclusiones sobre los resultados obtenidos en el proyecto y se realizan propuestas para mejoras e implementaciones de los modelos en diversas áreas de trabajo.

Esta estructura está diseñada para proporcionar una visión completa y detallada del tema, permitiendo una comprensión profunda tanto de los fundamentos teóricos como de las aplicaciones prácticas de los modelos de visión por computador basados en Transformers en dispositivos móviles.

A través de este trabajo, se pretende contribuir a la comprensión y mejora de los modelos de visión por computador basados en Transformers en el entorno específico de los dispositivos móviles, ofreciendo tanto una revisión crítica del estado actual como propuestas para futuras investigaciones y desarrollos.

Capítulo 2

Estado del arte

La visión por computador ha experimentado una evolución significativa en las últimas décadas, con la introducción de redes neuronales profundas que han revolucionado el campo. Recientemente, los modelos basados en Transformers, conocidos por su éxito en el procesamiento de lenguaje natural, han comenzado a mostrar resultados prometedores en tareas de visión por computador. Esta sección revisará los desarrollos más recientes en el uso de Transformers para visión por computador, con un énfasis particular en su aplicación en dispositivos móviles y en tiempo real.

En los últimos años, la visión por computador ha evolucionado drásticamente gracias a los avances en las redes neuronales profundas, que han permitido mejoras significativas en la precisión y eficiencia de diversas aplicaciones. Entre estos avances, los modelos de Transformers, inicialmente diseñados para tareas de procesamiento de lenguaje natural (NLP) [5], han demostrado un potencial notable en el ámbito de la visión por computador. La capacidad de estos modelos para manejar grandes volúmenes de datos y capturar dependencias a largo plazo ha abierto nuevas posibilidades para el análisis y procesamiento de imágenes.

Revisar la literatura existente es crucial para comprender el progreso y las tendencias actuales en el uso de transformers para visión por computador. Esta revisión no solo proporciona un marco contextual para el desarrollo de nuevos modelos y aplicaciones, sino que también identifica las áreas donde se requiere mayor investigación. Además, dada la creciente importancia de las aplicaciones en tiempo real en dispositivos móviles, es fundamental explorar cómo estos modelos pueden ser adaptados y optimizados para operar eficientemente en entornos con recursos limitados.

Los objetivos de esta revisión del estado del arte son:

- Proporcionar una visión general de los avances recientes en modelos de

CAPÍTULO 2. ESTADO DEL ARTE

visión por computador basados en transformers.

- Analizar las aplicaciones y adaptaciones de estos modelos en el contexto de dispositivos móviles y entornos de tiempo real.
- Identificar los desafíos actuales y las posibles direcciones futuras en esta área de investigación.

A través de esta revisión, buscamos establecer una base sólida que guíe futuras investigaciones y desarrollos en el uso de Transformers para visión por computador, especialmente en aplicaciones que requieren procesamiento eficiente en tiempo real en dispositivos móviles.

2.1. Antecedentes generales

2.1.1. Visión por Computador

La visión por computador es una disciplina dentro de la inteligencia artificial que se dedica a replicar y automatizar las capacidades del sistema visual humano en las máquinas. Este área se centra en el desarrollo de algoritmos y técnicas que permitan a las máquinas interpretar, procesar y comprender imágenes y videos.

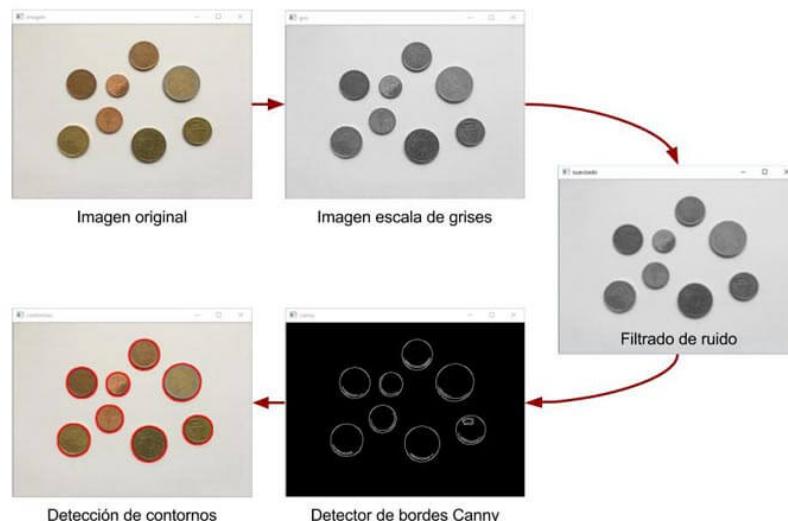


Figura 2.1: Detector de bordes de Canny [28].

2.1. ANTECEDENTES GENERALES

En sus primeras etapas, la visión por computador dependía de métodos geométricos y análisis de imágenes para la detección de bordes, formas y texturas. Los algoritmos utilizados en esta época, como el detector de bordes de Canny [51] y el operador Sobel [52], eran fundamentales para identificar los contornos de los objetos dentro de una imagen. La detección de formas, empleando métodos como la transformada de Hough [53], se utilizaba para identificar formas geométricas específicas en una imagen, mientras que el análisis de texturas se basaba en el análisis de patrones de repetición para clasificar y segmentar diferentes texturas.

El campo experimentó una revolución con la introducción de las redes neuronales convolucionales (CNNs) en la década de 1980 por Yann LeCun [54] y su popularización en la década de 2010. Las CNNs permitieron la extracción automática de características y el reconocimiento de patrones con alta precisión. Componentes clave de las CNNs incluyen las capas convolucionales, que aplican filtros a las imágenes para detectar características locales, y las capas de pooling, que reducen la dimensionalidad de las características extraídas. Además, las capas densas *fully connected* permiten la combinación y clasificación final de estas características aprendidas. Modelos como AlexNet [55], que ganó el concurso ImageNet en 2012, VGGNet [56], que introdujo redes más profundas con pequeños filtros de convolución en 2014, y ResNet [57], que introdujo conexiones residuales en 2015, marcaron hitos importantes en la evolución de las CNNs.

ARQUITECTURA DE UNA CNN

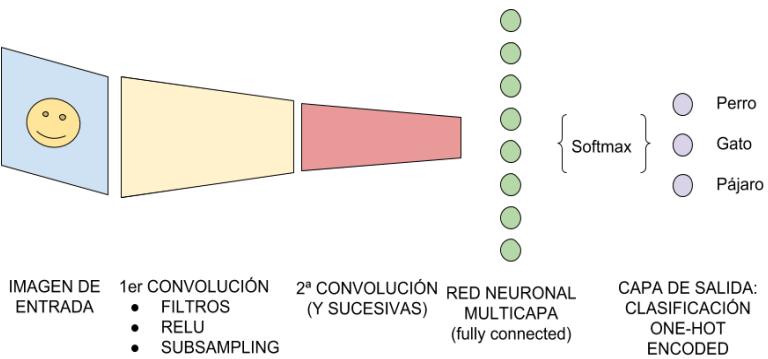


Figura 2.2: Arquitectura Redes Neuronales Convolucionales (CNNs) [29].

Más recientemente, los Transformers, originalmente diseñados para el procesamiento de lenguaje natural, han comenzado a ser adaptados para tareas

CAPÍTULO 2. ESTADO DEL ARTE

de visión por computador [11], ofreciendo nuevas posibilidades para el análisis de datos visuales. Las aplicaciones clave de la visión por computador incluyen el reconocimiento de imágenes y objetos, utilizado en seguridad, automóviles autónomos y aplicaciones móviles; el análisis de video, que abarca vigilancia, deportes y análisis de tráfico; y la visión médica, que se aplica en el diagnóstico asistido por IA y el análisis de imágenes médicas.

2.1.2. Modelos basados en Transformers

Los Transformers son una arquitectura de redes neuronales para tareas de procesamiento de lenguaje natural. Se destacan por su capacidad para manejar dependencias a largo plazo en secuencias de datos mediante un mecanismo de autoatención que permite procesar todos los elementos de una secuencia simultáneamente.

El mecanismo de autoatención de los transformers permite que cada token (unidad de datos) en una secuencia considere todos los otros tokens, ponderando su importancia relativa. Esto se realiza mediante matrices de consulta (Q), clave (K) y valor (V). Cada token en la secuencia se multiplica por estas matrices, produciendo tres representaciones distintas. La atención se calcula multiplicando Q por la transpuesta de K , seguido de una normalización mediante *softmax*. Los valores resultantes se multiplican por V para obtener la salida final de atención. Este enfoque permite capturar relaciones globales y dependencias a largo plazo en los datos, procesando todas las posiciones de la secuencia en paralelo y mejorando la eficiencia computacional.

Los Transformers utilizan *embeddings* para convertir tokens en vectores de alta dimensión que capturan su significado contextual. En el caso de la visión por computador, esto se traduce en la conversión de píxeles o parches de imagen en vectores. Los parches de imagen se apllanan y se proyectan a una dimensión fija utilizando una capa lineal, creando un *embedding* para cada parche. Estos *embeddings* incluyen información posicional para retener el orden espacial de los parches.

La arquitectura de los Transformers está compuesta por múltiples capas que aplican el mecanismo de atención y capas *feed-forward* (de avance directo) para transformar los *embeddings* a través de la red. Cada capa consiste en una subcapa de autoatención seguida de una red neuronal *feed-forward*. La salida de cada subcapa se normaliza y se suma a la entrada original mediante una conexión residual, mejorando la estabilidad y el flujo de gradiente durante el entrenamiento.

En el ámbito de la visión por computador, se han realizado adaptaciones significativas de los Transformers. Los Vision Transformers (ViT), introducidos por Dosovitskiy et al. en 2020, dividen las imágenes en parches y los

2.1. ANTECEDENTES GENERALES

tratan como una secuencia, similar a los tokens en el procesamiento de lenguaje natural. Cada parche se convierte en un *embedding* que pasa por el modelo Transformer, combinándose finalmente para producir una representación de la imagen completa. Los Transformers híbridos combinan elementos de CNNs y Transformers para aprovechar las fortalezas de ambos enfoques. Las CNNs extraen características locales que luego son procesadas por la estructura de Transformers para capturar dependencias globales. Ejemplos de estos modelos incluyen DeiT (Data-efficient Image Transformers) [58], que optimiza el entrenamiento de Transformers para visión por computador utilizando técnicas de distilación de conocimiento, y Swin Transformers (Shifted Window Transformers) [59], que utilizan ventanas deslizantes para aplicar el mecanismo de atención, permitiendo una mejor captura de características locales y globales.

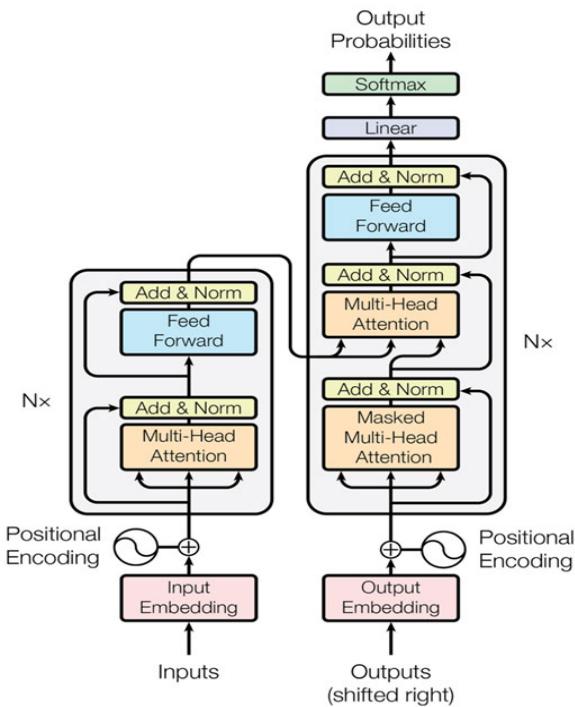


Figura 2.3: Arquitectura modelo basado en Transformer [30].

Los Transformers presentan ventajas significativas, como la capacidad de capturar dependencias a largo plazo y procesar secuencias enteras en paralelo. Sin embargo, también enfrentan desafíos importantes, incluyendo requerimientos computacionales elevados y la necesidad de grandes cantidades de datos para entrenar eficazmente. La adaptación y optimización para dispositi-

tivos móviles y aplicaciones en tiempo real son áreas activas de investigación debido a las limitaciones de recursos.

En conclusión, los antecedentes generales presentados aquí proporcionan el contexto necesario para entender el uso de Transformers en visión por computador. Lo más destacado en la progresión de este campo es la evolución desde las técnicas tradicionales hasta las CNNs y los Transformers. La descripción de los Transformers y sus adaptaciones para tareas de visión disponen de una revisión más detallada en las siguientes secciones del estado del arte.

2.2. Revisión de la literatura

2.2.1. Modelos de Visión por Computador tradicionales

La evolución de los modelos de visión por computador ha sido una trayectoria de innovación constante, iniciada por métodos clásicos y evolucionando hacia técnicas más sofisticadas y eficientes. Uno de los hitos más significativos en esta evolución fue la introducción de las Redes Neuronales Convolucionales (CNNs).

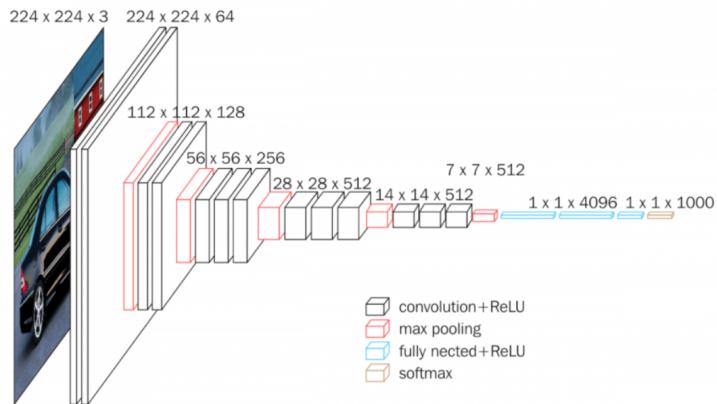


Figura 2.4: Arquitectura VGG [31].

En los primeros años, la visión por computador se basaba en técnicas geométricas y análisis de imágenes. Estos métodos utilizaban algoritmos de detección de bordes, como el detector de Canny y el operador Sobel, para identificar contornos dentro de una imagen. La transformada de Hough se empleaba para detectar formas geométricas, mientras que el análisis de texturas se basaba en patrones de repetición dentro de las imágenes. Sin embargo,

2.2. REVISIÓN DE LA LITERATURA

estas técnicas eran limitadas en su capacidad para manejar la variabilidad y complejidad de las imágenes del mundo real.

La verdadera revolución llegó con las CNNs. Propuestas inicialmente por Yann LeCun en la década de 1980 y popularizadas en 2012 con la victoria de AlexNet en la competencia ImageNet, las CNNs cambiaron radicalmente el panorama. Las CNNs permitieron la extracción automática de características de las imágenes y el reconocimiento de patrones con alta precisión. La arquitectura de las CNNs incluye capas convolucionales para detectar características locales, capas de pooling para reducir la dimensionalidad y capas fully connected para la clasificación final. A lo largo de los años, las CNNs han evolucionado con arquitecturas más profundas y complejas, como VGG-Net, que utilizó pequeñas convoluciones (3x3) para mejorar la precisión, y ResNet, que introdujo conexiones residuales para entrenar redes extremadamente profundas sin sufrir problemas de degradación del gradiente.

2.2.2. Transformers en Visión por Computador

Los transformers, originalmente diseñados para el procesamiento de lenguaje natural (NLP), han comenzado a ser adaptados para el campo de la visión por computador, ofreciendo nuevas posibilidades y ventajas. El mecanismo de autoatención de los transformers, que permite manejar dependencias a largo plazo en secuencias de datos, ha demostrado ser altamente efectivo también en el análisis de imágenes.

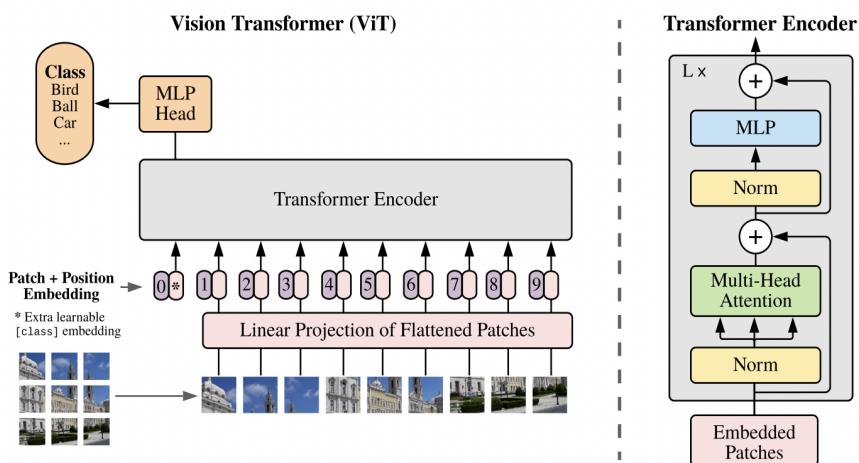


Figura 2.5: Arquitectura Vision Transformer (ViT) [32].

Uno de los desarrollos más importantes en esta área es el Vision Transfor-

CAPÍTULO 2. ESTADO DEL ARTE

mer (ViT), introducido por Dosovitskiy et al. en 2020. Los ViT dividen las imágenes en parches y tratan estos parches como una secuencia de tokens, similar a los tokens en NLP. Cada parche se convierte en un embedding, que luego pasa por un modelo transformer para capturar las relaciones globales entre diferentes partes de la imagen. Este enfoque ha demostrado ser efectivo en diversas tareas de visión por computador, alcanzando resultados competitivos con las CNNs en varios benchmarks.

Además de los ViT, han surgido variantes híbridas que combinan elementos de CNNs y transformers. Estas arquitecturas híbridas buscan aprovechar las fortalezas de ambos enfoques: las CNNs para la extracción de características locales y los transformers para capturar dependencias globales. Ejemplos notables incluyen los DeiT (Data-efficient Image Transformers), que optimizan el entrenamiento de transformers para visión por computador mediante técnicas de distilación de conocimiento, y los Swin Transformers (Shifted Window Transformers), que utilizan ventanas deslizantes para aplicar el mecanismo de atención, permitiendo una mejor captura de características locales y globales.

2.2.3. Aplicaciones en tiempo real

El uso de modelos de visión por computador en aplicaciones en tiempo real, especialmente en dispositivos móviles, presenta un conjunto único de desafíos y oportunidades. La implementación de estos modelos en dispositivos con recursos limitados requiere una optimización cuidadosa tanto del modelo como del hardware.

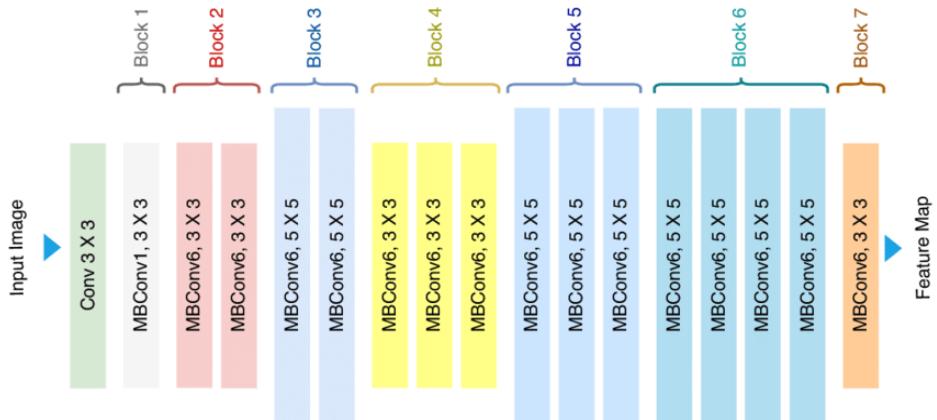


Figura 2.6: Arquitectura EfficientNet-B0 [33].

Varios estudios han explorado la implementación de modelos de visión por computador en dispositivos móviles. Estos estudios se han centrado en

2.2. REVISIÓN DE LA LITERATURA

reducir la complejidad computacional de los modelos mediante técnicas como la poda de redes neuronales, la cuantización de pesos y el diseño de arquitecturas más eficientes. Por ejemplo, MobileNets [8] y EfficientNets [60] son arquitecturas diseñadas específicamente para aplicaciones en dispositivos con recursos limitados. Estas arquitecturas utilizan convoluciones separables y otros trucos de diseño para reducir el número de parámetros y operaciones necesarias, manteniendo al mismo tiempo un rendimiento competitivo.

Un desafío importante en las aplicaciones en tiempo real es el balance entre precisión y latencia. Los modelos deben ser lo suficientemente precisos para cumplir con los requisitos de la aplicación, pero también lo suficientemente rápidos para procesar imágenes en tiempo real sin causar retrasos significativos. Esto es especialmente crítico en aplicaciones como la conducción autónoma, donde los retrasos en el procesamiento pueden tener consecuencias graves.

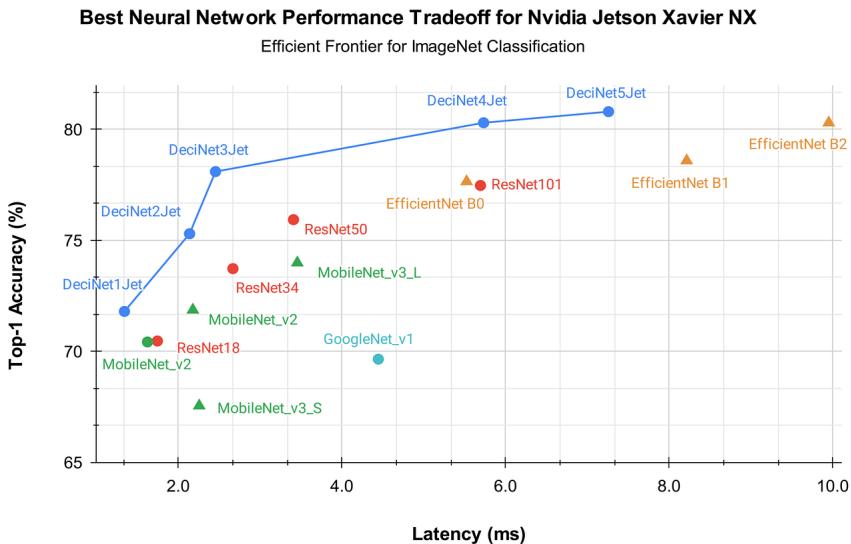


Figura 2.7: Gráfica latencia-precisión modelos visión por computador [34].

La investigación en esta área también ha explorado el uso de técnicas de compresión de modelos y la implementación de inferencias distribuidas, donde partes del modelo se ejecutan en el dispositivo móvil y otras partes en la nube. Esta combinación permite aprovechar la potencia computacional de la nube mientras se minimiza la latencia.

En resumen, la revisión de la literatura muestra un progreso significativo en el campo de la visión por computador, desde los métodos tradicionales

hasta las modernas arquitecturas basadas en Transformers. Las aplicaciones en tiempo real en dispositivos móviles siguen siendo un área activa de investigación, con un enfoque en la optimización de modelos y la eficiencia computacional para superar los desafíos de recursos limitados y latencia.

2.3. Comparación y discusión

En la evolución de la visión por computador, tanto las Redes Neuronales Convolucionales (CNNs) como los Transformers han demostrado ser enfoques poderosos, cada uno con sus propias ventajas y desafíos. Esta sección comparará estos diferentes enfoques, discutiendo sus fortalezas y debilidades, y cómo se han abordado los problemas de eficiencia y precisión.

2.3.1. Comparación de Enfoques

Las CNNs y los Transformers abordan la visión por computador de maneras fundamentalmente diferentes. Las CNNs se basan en operaciones convolucionales que aplican filtros a pequeñas regiones de la imagen para extraer características locales. Este enfoque ha demostrado ser altamente eficaz en la detección de patrones locales y ha sido la base de muchos avances significativos en el campo.

Por otro lado, los Transformers utilizan un mecanismo de autoatención que permite capturar dependencias globales en los datos. Mientras que las CNNs se enfocan en características locales y construyen una comprensión global a través de múltiples capas, los Transformers pueden considerar toda la imagen simultáneamente, lo que les permite capturar relaciones a largo plazo de manera más efectiva.

2.3.2. Ventajas y Desventajas

Ventajas de las Redes Neuronales Convolucionales (CNNs)

- **Eficiencia computacional:** Las operaciones convolucionales son altamente eficientes y se pueden acelerar utilizando hardware especializado como GPUs y TPUs.
- **Eficacia en características locales:** Las CNNs son extremadamente buenas para capturar características locales, lo que las hace ideales para tareas como la detección de objetos y el reconocimiento de patrones.

2.3. COMPARACIÓN Y DISCUSIÓN

- **Madurez del campo:** La investigación en CNNs está bien establecida, con numerosas técnicas y herramientas disponibles para mejorar su rendimiento.

Desventajas de las Redes Neuronales Convolucionales (CNNs)

- **Limitaciones en capturar dependencias globales:** Las CNNs pueden tener dificultades para capturar dependencias a largo plazo y relaciones globales debido a su naturaleza local.
- **Eficiencia en recursos:** Aunque son eficientes, las CNNs profundas pueden ser costosas en términos de memoria y poder computacional, especialmente en dispositivos con recursos limitados.

Ventajas de los Transformers

- **Captura de dependencias globales:** Los Transformers pueden capturar dependencias a largo plazo y relaciones globales de manera más efectiva que las CNNs.
- **Flexibilidad:** Los Transformers son altamente flexibles y pueden adaptarse a una variedad de tareas y tipos de datos.
- **Paralelización:** El mecanismo de autoatención permite la paralelización de la secuencia completa, mejorando la eficiencia en el entrenamiento y la inferencia.

Desventajas de los Transformers

- **Requerimientos computacionales:** Los Transformers requieren una gran cantidad de datos y poder computacional para entrenar eficazmente, lo que puede ser un obstáculo en entornos con recursos limitados.
- **Complejidad de implementación:** Implementar Transformers puede ser más complejo en comparación con las CNNs, especialmente en aplicaciones en tiempo real.
- **Problemas de eficiencia y precisión:** Uno de los principales desafíos tanto para las CNNs como para los Transformers es equilibrar la precisión con la eficiencia, especialmente en aplicaciones en tiempo real y en dispositivos móviles. Diversas estrategias se han desarrollado para abordar estos problemas.

CAPÍTULO 2. ESTADO DEL ARTE

Optimización de CNNs

- **Pruning:** Esta técnica [43] implica eliminar pesos innecesarios en la red, reduciendo la complejidad del modelo sin comprometer significativamente la precisión.
- **Cuantización:** Consiste [42] en reducir la precisión de los pesos del modelo, lo que disminuye el uso de memoria y mejora la velocidad de inferencia.
- **Arquitecturas eficientes:** Modelos como MobileNets y EfficientNets están diseñados específicamente para ser eficientes en términos de recursos, utilizando técnicas como convoluciones separables en profundidad y escalado compuesto.

Optimización de Transformers

- **Distilación de conocimiento:** Los modelos más pequeños son entrenados para imitar el comportamiento de modelos más grandes, manteniendo un buen rendimiento con menor complejidad.
- **Transformers híbridos:** La combinación de CNNs y Transformers puede aprovechar las fortalezas de ambos, utilizando CNNs para la extracción de características locales y Transformers para capturar dependencias globales.
- **Arquitecturas eficientes:** Modelos como los Vision Transformers (ViT) y los Swin Transformers utilizan técnicas innovadoras para mejorar la eficiencia, como el uso de ventanas deslizantes y la reducción de la resolución de entrada.

En conclusión, tanto las CNNs como los Transformers ofrecen enfoques poderosos para la visión por computador, cada uno con sus propias ventajas y desafíos. Las CNNs son altamente eficientes y efectivas para capturar características locales, mientras que los Transformers destacan en la captura de relaciones globales y dependencias a largo plazo. Los esfuerzos continuos para optimizar estos modelos y adaptarlos a aplicaciones en tiempo real y dispositivos móviles están llevando a avances significativos en la eficiencia y precisión, allanando el camino para una adopción más amplia de estas tecnologías en el futuro.

2.4. Conclusiones del estado del arte

La revisión de la literatura sobre visión por computador y los modelos basados en Transformers revela un panorama dinámico y en constante evolución. Los avances en las Redes Neuronales Convolucionales (CNNs) han sido fundamentales para el progreso en el campo de la visión por computador, permitiendo la extracción automática de características y el reconocimiento de patrones con alta precisión. Las CNNs han demostrado ser eficaces en una variedad de aplicaciones, desde el reconocimiento de imágenes y objetos hasta el análisis de vídeo y la visión médica.

La introducción de los Transformers, originalmente diseñados para el procesamiento de lenguaje natural, ha abierto nuevas posibilidades en la visión por computador. Los Vision Transformers (ViT) y sus variantes han mostrado que es posible capturar relaciones globales y dependencias a largo plazo en las imágenes, superando algunas de las limitaciones inherentes a las CNNs. Los Transformers híbridos, que combinan elementos de CNNs y Transformers, representan un enfoque prometedor que aprovecha las fortalezas de ambos paradigmas.

En el contexto de aplicaciones en tiempo real, especialmente en dispositivos móviles, se han realizado avances significativos para mejorar la eficiencia y reducir los requerimientos computacionales. Las técnicas de optimización como la poda, la cuantización y la distilación de conocimiento han permitido implementar modelos de visión por computador más eficientes, sin comprometer significativamente la precisión.

A pesar de los avances significativos, existen varias brechas en la literatura que presentan oportunidades para futuras investigaciones:

- **Eficiencia computacional y optimización:** Aunque se han logrado mejoras en la eficiencia de los modelos, la implementación en dispositivos móviles y otros entornos con recursos limitados sigue siendo un desafío.
- **Adaptación de Transformers para Visión por Computador:** Si bien los Transformers han demostrado ser prometedores en la visión por computador, aún existen desafíos relacionados con su complejidad y requerimientos computacionales. Explorar arquitecturas de Transformers más ligeras y eficientes, así como técnicas de entrenamiento más efectivas.
- **Robustez y generalización:** Asegurar que los modelos de visión por computador sean robustos y puedan generalizar bien a diferentes condiciones y contextos es una área crítica.

CAPÍTULO 2. ESTADO DEL ARTE

- **Aplicaciones especializadas:** Existen áreas específicas de aplicación, como la visión médica, donde los modelos de visión por computador pueden tener un impacto significativo. La adaptación y optimización de modelos para estas aplicaciones especializadas, considerando las características únicas de los datos y los requerimientos del dominio, representan una oportunidad importante.
- **Integración con tecnologías emergentes:** La integración de modelos de visión por computador con tecnologías emergentes como la Internet de las Cosas (IoT), la realidad aumentada (AR) y la realidad virtual (VR) puede abrir nuevas oportunidades de investigación y aplicación.

En conclusión, la revisión del estado del arte destaca el progreso significativo realizado en el campo de la visión por computador, desde los métodos tradicionales hasta las modernas arquitecturas basadas en Transformers. A medida que la tecnología continúa avanzando, la investigación futura tiene el potencial de abordar las brechas identificadas y explorar nuevas oportunidades, impulsando aún más el campo de la visión por computador hacia aplicaciones más eficientes y efectivas en una variedad de dominios.

Capítulo 3

Marco teórico

El marco teórico de esta investigación se fundamenta en los principios de la visión por computador y los modelos de Transformers. A lo largo de esta sección, se explorarán los conceptos esenciales y las teorías que sustentan el uso de Transformers en tareas de visión por computador. Este marco teórico proporciona el contexto conceptual necesario para comprender los fundamentos de la investigación y cómo se alinean con los objetivos del estudio.

3.1. Conceptualización Avanzada y Definiciones

Este apartado se centra en ofrecer una comprensión profunda y técnica de los conceptos clave que sustentan el uso de modelos de visión por computador basados en Transformers en dispositivos móviles. A diferencia del estado del arte, que proporciona una visión general de los antecedentes, aquí se exploran los conceptos desde una perspectiva avanzada, considerando los desafíos únicos y las oportunidades que surgen al aplicar estas tecnologías en un contexto móvil.

3.1.1. Visión por Computador en Dispositivos Móviles

La visión por computador (VC) en dispositivos móviles es una extensión de la inteligencia artificial que emula la percepción visual humana mediante el análisis de imágenes y videos en tiempo real. Sin embargo, a diferencia de las implementaciones tradicionales que se ejecutan en potentes servidores, la visión por computador en dispositivos móviles enfrenta una serie de desafíos técnicos debido a las limitaciones inherentes del hardware, como la capacidad de procesamiento, la memoria, y el consumo energético. A pesar

CAPÍTULO 3. MARCO TEÓRICO

de estos desafíos, la integración de VC en dispositivos móviles ha permitido el desarrollo de aplicaciones innovadoras y prácticas que impactan diversas áreas de la vida cotidiana y la industria.

Para operar eficazmente en dispositivos móviles, la visión por computador debe superar varios obstáculos:

- **Capacidad de procesamiento limitada:** Los dispositivos móviles cuentan con CPUs y GPUs menos potentes, lo que obliga a emplear modelos de visión por computador más eficientes y optimizados. Esto implica, por ejemplo, el uso de redes neuronales más ligeras y técnicas de optimización que permiten reducir la carga computacional sin comprometer significativamente la precisión.
- **Restricciones de memoria:** Dado que la memoria RAM y el almacenamiento en dispositivos móviles son limitados, los modelos deben ser compactos y diseñados para consumir menos memoria, utilizando técnicas como la compresión de modelos o la poda de redes neuronales.
- **Consumo energético:** El procesamiento intensivo puede agotar rápidamente la batería de un dispositivo móvil. Por lo tanto, los modelos de visión por computador deben ser optimizados para reducir el consumo energético, utilizando, por ejemplo, operaciones de bajo consumo o activaciones más simples.
- **Latencia y tiempo real:** Las aplicaciones móviles suelen requerir procesamiento en tiempo real, lo que impone límites estrictos en la latencia permitida para la ejecución de los modelos. Esto es crítico en aplicaciones como la realidad aumentada o la conducción asistida, donde la rapidez de la respuesta es esencial.

Para superar estos desafíos, la visión por computador en dispositivos móviles debe adaptarse, empleando técnicas como la compresión de modelos, la optimización de redes neuronales para hardware específico y la utilización de arquitecturas ligeras que mantengan un equilibrio entre precisión y eficiencia.

A pesar de los desafíos mencionados, la visión por computador en dispositivos móviles ha permitido el desarrollo de una amplia gama de aplicaciones que han transformado tanto la experiencia del usuario como múltiples industrias:

- **Reconocimiento facial:** Una de las aplicaciones más extendidas es el reconocimiento facial, utilizado en seguridad biométrica, desbloqueo de

3.1. CONCEPTUALIZACIÓN AVANZADA Y DEFINICIONES

dispositivos, y autenticación en aplicaciones. Esta tecnología analiza y compara rasgos faciales en tiempo real, permitiendo una identificación rápida y segura.

- **Realidad Aumentada (AR):** En aplicaciones de AR, la visión por computador permite superponer objetos digitales en el entorno físico del usuario. Esto se aplica en juegos, en herramientas de diseño, y en aplicaciones de navegación que integran información visual sobre el entorno real.
- **Reconocimiento de objetos y texto:** Aplicaciones como Google Lens utilizan la visión por computador para identificar y proporcionar información sobre objetos o textos capturados con la cámara del móvil. Esto incluye traducción instantánea de textos, identificación de productos para compras, o reconocimiento de especies vegetales.
- **Conducción asistida y sistemas de seguridad:** En automóviles equipados con dispositivos móviles o sistemas integrados, la visión por computador se utiliza para asistir en la conducción mediante el reconocimiento de señales de tráfico, detección de peatones, y monitoreo de la fatiga del conductor, mejorando así la seguridad vial.
- **Medicina y salud:** En el ámbito de la salud, la visión por computador en dispositivos móviles se aplica en la telemedicina para realizar diagnósticos preliminares a partir de imágenes capturadas por el usuario, como el análisis de lesiones cutáneas o la monitorización de signos vitales.
- **Comercio electrónico:** La VC en móviles permite experiencias de compra más interactivas, como la prueba virtual de ropa o maquillaje, donde los usuarios pueden ver cómo les quedarían los productos sin necesidad de tenerlos físicamente, utilizando modelos 3D generados en tiempo real.

Estas aplicaciones demuestran el potencial transformador de la visión por computador en dispositivos móviles, al permitir que tecnologías avanzadas estén al alcance de los usuarios en su vida diaria. La continua optimización de los modelos para hacer frente a las limitaciones de los dispositivos móviles es clave para la expansión de estas aplicaciones y la creación de nuevas soluciones que aprovechen al máximo las capacidades de la VC.

3.1.2. Transformers en Visión por Computador: Principios y Funcionalidades

Los Transformers, originalmente desarrollados para tareas de procesamiento de lenguaje natural (NLP), han demostrado ser igualmente poderosos en aplicaciones de visión por computador. Su éxito radica en la capacidad de modelar relaciones a largo plazo entre diferentes partes de una imagen, utilizando mecanismos de atención que ponderan la importancia de cada región en el contexto global.

En visión por computador, los Transformers introducen varias innovaciones clave:

- **Mecanismo de Atención (Attention Mechanism):** A diferencia de las Convolutional Neural Networks (CNNs) tradicionales, los Transformers no dependen de filtros convolucionales locales. En su lugar, utilizan un mecanismo de atención que evalúa la relevancia de todas las partes de una imagen simultáneamente, permitiendo capturar relaciones espaciales complejas y mejorar la interpretación de características globales.
- **Representación espacial de la imagen:** Los Transformers dividen la imagen en parches (pequeños fragmentos), los cuales se procesan como si fueran palabras en una frase, similar a cómo funcionan en NLP. Este enfoque permite a los Transformers manejar imágenes con diferentes resoluciones de manera más flexible.
- **Adaptación a dispositivos móviles:** Implementar Transformers en dispositivos móviles implica resolver problemas como la alta complejidad computacional y el consumo de memoria. Los modelos como los Vision Transformers (ViT) han sido modificados para reducir su tamaño y aumentar la eficiencia, utilizando técnicas como el distillation, la cuantización de pesos, y la fusión de capas.

En un modelo Transformer, la entrada se convierte en una serie de *embeddings* que representan cada elemento de la secuencia. Estos *embeddings* pasan a través de múltiples capas de atención y *feed-forward*, donde se procesan y transforman para capturar las relaciones contextuales entre los elementos. La salida final del modelo es una representación enriquecida que puede utilizarse para tareas específicas, como la clasificación de imágenes en Visión por Computador o la generación de texto en NLP.

Los conceptos vistos sobre Visión por Computador y modelos de Transformers proporciona una comprensión fundamental de cómo se procesan y

3.2. MODELOS Y ALGORITMOS RELEVANTES

analizan los datos visuales y secuenciales. Al definir estos conceptos clave y explicar los componentes principales de los Transformers, se establece una base sólida para la investigación y el desarrollo en estas áreas.

Los transformers han abierto nuevas posibilidades en visión por computador, permitiendo el desarrollo de modelos más robustos y generalizables, capaces de realizar tareas como la detección de objetos, segmentación de imágenes y reconocimiento facial con una precisión sin precedentes. Sin embargo, su implementación en dispositivos móviles aún está en una fase de desarrollo activo, con investigaciones enfocadas en mejorar su eficiencia sin sacrificar rendimiento.

3.2. Modelos y Algoritmos Relevantes

En este apartado se presentan los modelos y algoritmos más relevantes que han sido desarrollados o adaptados para aplicaciones de visión por computador en dispositivos móviles. Estos modelos representan el estado del arte en el uso de Transformers para el procesamiento de imágenes y videos, y se han optimizado para superar las limitaciones de los dispositivos móviles, como la capacidad de procesamiento, la memoria, y el consumo energético. Además, se exploran los modelos específicos que han demostrado ser particularmente eficaces en aplicaciones móviles.

3.2.1. Vision Transformers (ViT)

Los Vision Transformers (ViT) [11] han marcado un hito en el campo de la visión por computador al aplicar con éxito la arquitectura de Transformers, originalmente diseñada para procesamiento de lenguaje natural, a tareas de visión. A diferencia de las Convolutional Neural Networks (CNNs), que dominaban el campo, los ViT no dependen de convoluciones locales sino de un mecanismo de atención global que permite modelar relaciones a largo plazo entre diferentes partes de una imagen.

Características principales de los ViT

- **Entrada basada en parches:** Las imágenes se dividen en parches cuadrados, que luego se linealizan y se procesan como si fueran tokens en un modelo de lenguaje.
- **Atención global:** El mecanismo de atención permite que el modelo considere la relación entre todos los parches simultáneamente, captu-

rando características globales que son difíciles de modelar con CNNs tradicionales.

- **Escalabilidad:** Los ViT han demostrado que, con suficiente cantidad de datos, pueden superar a las CNNs en varias tareas de visión por computador, particularmente en clasificación de imágenes.

Aplicaciones en dispositivos móviles: Aunque los ViT tienen un alto costo computacional, se han desarrollado versiones más ligeras y eficientes, como los MobileViT, que están optimizados para funcionar en hardware con recursos limitados, manteniendo un buen equilibrio entre precisión y eficiencia.

3.2.2. Transformers Híbridos

Los Transformers Híbridos [61] combinan la capacidad de los Transformers para modelar relaciones globales con la eficiencia de las CNNs para capturar características locales. Estas arquitecturas híbridas han sido desarrolladas para aprovechar las fortalezas de ambos enfoques, especialmente en escenarios donde se requiere un alto rendimiento en tiempo real, como en dispositivos móviles.

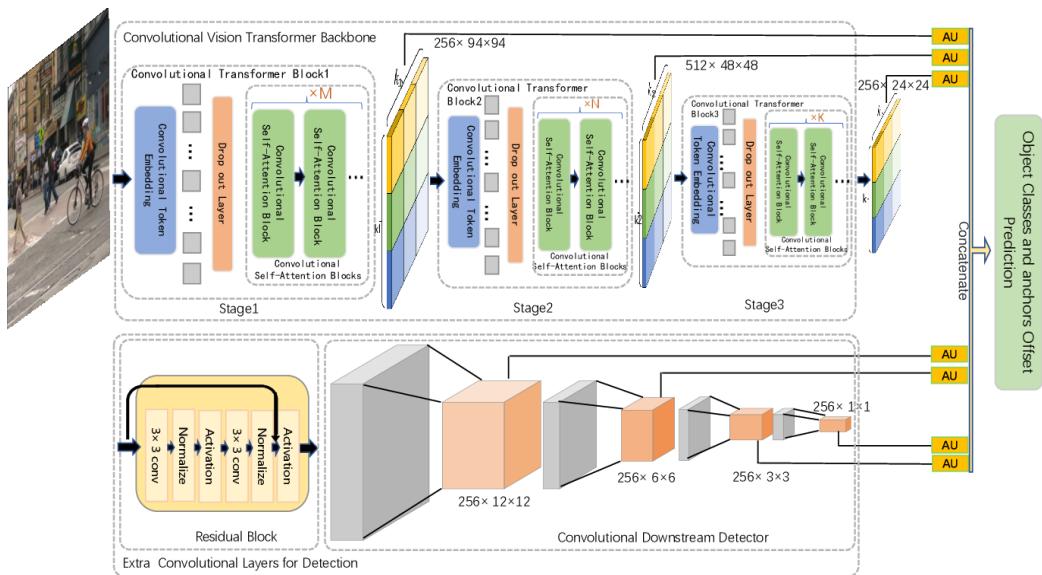


Figura 3.1: Arquitectura Convolutional Vision Transformer (CvT) [35].

--- 3.2. MODELOS Y ALGORITMOS RELEVANTES

Ejemplos de arquitecturas híbridas

- **Convolutional Vision Transformer (CvT)**: Este modelo [35] introduce convoluciones en el proceso de atención de los ViT, permitiendo una mejor captura de características locales y una reducción en el costo computacional.
- **DeiT (Data-efficient Image Transformer)**: DeiT [58] optimiza los ViT para ser entrenados de manera más eficiente, incluso con menos datos, lo que es crucial en escenarios donde la recolección de datos es limitada o costosa.
- **Swin Transformer**: Este modelo [59] divide la imagen en ventanas no superpuestas, y dentro de cada ventana se aplica el mecanismo de atención, permitiendo un procesamiento más eficiente y escalable en términos de resolución.

Relevancia para dispositivos móviles: Los Transformers Híbridos son especialmente útiles en aplicaciones móviles, donde la capacidad de procesamiento es limitada. Al combinar la eficiencia de las CNNs con la capacidad de los Transformers para capturar relaciones a largo plazo, estos modelos ofrecen una solución más balanceada y adecuada para aplicaciones en tiempo real.

3.2.3. Optimización para Dispositivos Móviles

La implementación de modelos de visión por computador basados en Transformers en dispositivos móviles requiere una optimización exhaustiva para garantizar un rendimiento eficiente sin comprometer la precisión. Las técnicas de optimización son esenciales para adaptar estos modelos a las limitaciones de hardware de los dispositivos móviles. Algunas estrategias clave incluyen:

Técnicas de optimización

- **Cuantización**: Esta técnica [42] reduce la precisión de los parámetros del modelo, como los pesos y las activaciones, de 32 bits de punto flotante a 16 o incluso 8 bits. La cuantización disminuye significativamente el tamaño del modelo y el consumo de energía, sin afectar gravemente la precisión.

- **Poda (Prunning):** La poda [43] implica eliminar partes del modelo que tienen poca o ninguna contribución al resultado final, como neuronas o conexiones menos importantes. Esto reduce la complejidad del modelo y mejora la velocidad de inferencia.
- **Distilación:** En esta técnica [62], un modelo grande (maestro) entrena a un modelo más pequeño (alumno), transfiriéndole conocimientos. El modelo alumno, que es más ligero, puede operar eficientemente en dispositivos móviles.

Implementación en hardware específico: Además de las optimizaciones de software, se emplean técnicas que aprovechan las capacidades del hardware específico de los dispositivos móviles, como las unidades de procesamiento de tensor (TPU) de Google o las Neural Processing Units (NPU) de algunos smartphones. Estas unidades están diseñadas para ejecutar operaciones de aprendizaje profundo de manera más eficiente que las CPU o GPU tradicionales.

3.2.4. Modelos Ligados a Aplicaciones Específicas (e.g. YOLO, DETR)

En este subapartado se exploran los modelos de visión por computador basados en Transformers que han sido específicamente adaptados o diseñados para aplicaciones móviles concretas, como la detección de objetos en tiempo real, la segmentación de imágenes, o el reconocimiento facial.

Modelos relevantes

- **YOLO (You Only Look Once):** Aunque originalmente no es un modelo basado en Transformers, YOLO [63] ha sido modificado y combinado con Transformers para mejorar la detección de objetos en tiempo real en dispositivos móviles. Su capacidad para procesar imágenes rápidamente lo hace ideal para aplicaciones como la vigilancia o la asistencia en la conducción.
- **DETR (Detection Transformer):** DETR [16] utiliza Transformers para redefinir el proceso de detección de objetos, eliminando la necesidad de procesos como el anclaje de cajas. Su enfoque basado en atención permite una detección más precisa y robusta, aunque originalmente más costosa en términos computacionales. Adaptaciones de DETR han sido desarrolladas para funcionar eficientemente en dispositivos móviles.

3.3. TEORÍAS Y PRINCIPIOS SUBYACENTES

- **MobileNetV3 con bloques Transformer:** MobileNetV3 [8] es una arquitectura ligera optimizada para dispositivos móviles que ha sido combinada con bloques de Transformers para mejorar la capacidad de modelado a largo plazo, manteniendo la eficiencia computacional. Este modelo es ideal para aplicaciones como el reconocimiento facial o la clasificación de imágenes con recursos limitados.

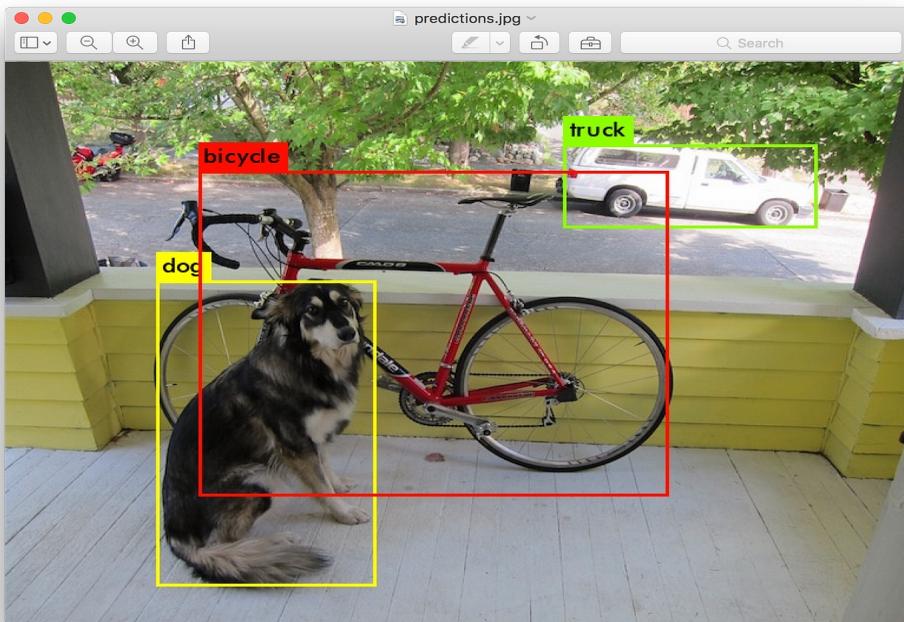


Figura 3.2: Detección de objetos con YOLO [36].

Impacto en aplicaciones móviles: Estos modelos han sido fundamentales para llevar aplicaciones avanzadas de visión por computador a dispositivos móviles, permitiendo experiencias como la realidad aumentada, la navegación asistida, y la interacción inteligente con el entorno, todo en tiempo real.

3.3. Teorías y Principios Subyacentes

En este apartado se examinan las teorías fundamentales y los principios técnicos que sustentan el desarrollo y la optimización de modelos de visión por computador basados en Transformers, especialmente en el contexto de

CAPÍTULO 3. MARCO TEÓRICO

dispositivos móviles. Se abordan conceptos clave como la teoría de la atención, el aprendizaje profundo en entornos con recursos limitados, y las técnicas avanzadas de optimización de modelos que son esenciales para lograr un rendimiento eficiente en hardware móvil.

3.3.1. Teoría de la Atención y su Aplicación en Transformers

La teoría de la atención [5] es un principio clave en el desarrollo de Transformers, tanto en procesamiento de lenguaje natural (NLP) como en visión por computador. Esta teoría se basa en la idea de que, al procesar información, no todos los elementos tienen la misma relevancia. En lugar de tratar todos los datos de manera uniforme, un modelo puede atender de manera diferenciada a las partes más importantes de la entrada, enfocándose en las que son más relevantes para la tarea específica. Esta teoría ha revolucionado el procesamiento de lenguaje natural y ha sido adaptada exitosamente para la visión por computador.

Aplicación en Transformers

- **Atención escalonada:** En los Transformers, la atención se aplica de manera escalonada a diferentes partes de la entrada, permitiendo que el modelo determine qué porciones de la imagen o secuencia de texto son más relevantes para la tarea de predicción. Esto es particularmente útil en tareas de visión por computador, donde ciertas características espaciales o patrones pueden ser más significativos que otros.
- **Self-Attention:** Los mecanismos de *self-attention* permiten que cada parte de la entrada (como un parche de una imagen) influya en todas las demás partes, creando una representación rica y contextualizada de la información. Esto es especialmente poderoso en visión por computador, ya que permite capturar relaciones a largo plazo dentro de la imagen, algo que las CNNs tradicionales, con su enfoque en las características locales, no pueden hacer de manera eficiente.
- **Atención Multi-Cabeza (Multi-Headed Attention):** Este mecanismo permite al modelo enfocarse simultáneamente en diferentes partes de la entrada desde distintas perspectivas, mejorando su capacidad para capturar características complejas y multifacéticas de las imágenes.

3.3. TEORÍAS Y PRINCIPIOS SUBYACENTES

Implementar mecanismos de atención en dispositivos móviles implica desafíos adicionales, como la necesidad de optimizar las operaciones de atención para minimizar el consumo de memoria y el costo computacional. Las versiones simplificadas y las técnicas de atención eficientes, como la atención de ventana local usada en los Swin Transformers, se han desarrollado para abordar estas limitaciones sin sacrificar el rendimiento.

3.3.2. Aprendizaje Profundo en Dispositivos Móviles

El aprendizaje profundo, aunque tradicionalmente dependiente de hardware de alto rendimiento como GPUs, ha evolucionado para ser aplicable en dispositivos móviles. Este subapartado explora las adaptaciones y técnicas específicas que permiten ejecutar modelos de aprendizaje profundo en entornos con recursos limitados.

El aprendizaje profundo ha transformado la visión por computador, pero su implementación en dispositivos móviles enfrenta desafíos significativos debido a las limitaciones de hardware, energía y latencia. Los dispositivos móviles, como smartphones y tablets, no disponen de la misma potencia de procesamiento y memoria que los servidores con GPUs dedicadas, lo que significa que los modelos de aprendizaje profundo, tradicionalmente grandes y complejos, deben ser adaptados o rediseñados para ajustarse a estos entornos más restringidos. Por ejemplo, mientras que en servidores es posible manejar modelos con millones de parámetros y operaciones intensivas, en dispositivos móviles esto puede resultar inviable sin una adecuada optimización.

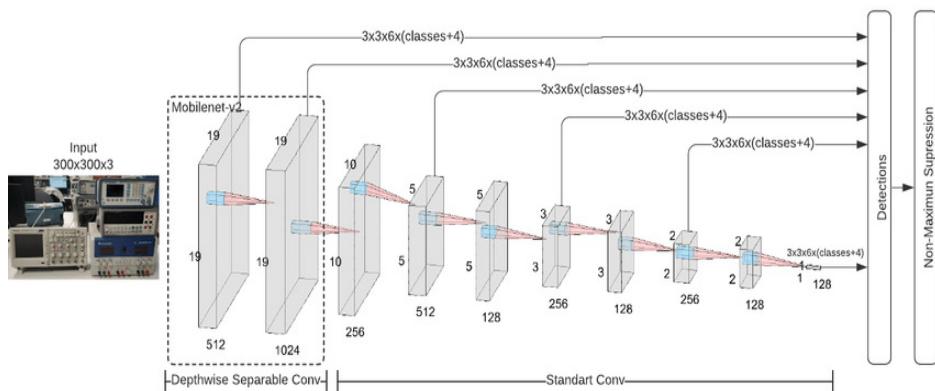


Figura 3.3: Arquitectura MobileNet [37].

El consumo energético es otro desafío crucial. Dado que los dispositivos móviles operan con baterías, es fundamental que los modelos de aprendizaje profundo sean optimizados para minimizar el consumo de energía, sin

CAPÍTULO 3. MARCO TEÓRICO

comprometer la precisión o la experiencia del usuario. Esto es especialmente relevante en aplicaciones como la realidad aumentada o el reconocimiento facial, donde los modelos deben estar activos durante largos períodos, haciendo imprescindible una gestión eficiente de la energía para evitar un drenaje rápido de la batería.

Además, la latencia, o el tiempo que tarda un modelo en procesar una entrada y generar una salida, debe mantenerse lo más baja posible, especialmente en aplicaciones que requieren respuestas en tiempo real. En escenarios como la navegación asistida, un retraso en el procesamiento puede tener consecuencias negativas, afectando la seguridad y la utilidad de la aplicación.

Para superar estos desafíos, existen diferentes propuestas:

- **Modelos ligeros:** Las arquitecturas como MobileNet [37] y EfficientNet [60] se han desarrollado específicamente para dispositivos móviles, optimizando el rendimiento en tareas de visión por computador con una menor cantidad de parámetros y operaciones.
- **Inferencia distribuida:** Dividir el procesamiento entre el dispositivo móvil y la nube puede reducir la carga local, permitiendo que solo las partes críticas de la inferencia se realicen en el dispositivo, mientras que las tareas más complejas se delegan a servidores remotos.

Estas adaptaciones han tenido un impacto significativo en diversas aplicaciones móviles. En el caso de la realidad aumentada, los modelos ligeros permiten que las aplicaciones funcionen en tiempo real, ofreciendo experiencias inmersivas sin una latencia notable. Para el reconocimiento facial, la optimización de la inferencia y el uso de modelos ligeros hacen posible que estos sistemas se ejecuten directamente en dispositivos móviles, mejorando la seguridad y privacidad del usuario. Además, los asistentes virtuales, que dependen de modelos de aprendizaje profundo para reconocer comandos de voz o gestos, se benefician de estas optimizaciones al ofrecer respuestas rápidas y precisas sin agotar la batería del dispositivo.

3.3.3. Comprensión y Cuantización de Modelos

La cuantización es una técnica crucial para optimizar los modelos de aprendizaje profundo en dispositivos con recursos limitados, como los móviles. Este proceso implica reducir la precisión de los parámetros del modelo, como los pesos y las activaciones, lo que disminuye el tamaño del modelo y la cantidad de operaciones necesarias para realizar inferencias.

3.3. TEORÍAS Y PRINCIPIOS SUBYACENTES

- **Cuantización Post-Entrenamiento:** Se aplica [42] después de que el modelo ha sido entrenado con precisión completa. Este enfoque convierte los parámetros del modelo de 32 bits a 16 o 8 bits, reduciendo su tamaño y el consumo energético.
- **Cuantización Aware Training (QAT):** [64] Durante el entrenamiento del modelo, se simula la cuantización para permitir que el modelo aprenda a ser robusto a las operaciones de baja precisión. Este método generalmente produce mejores resultados en términos de precisión que la cuantización post-entrenamiento.

La cuantización es particularmente útil en aplicaciones como el reconocimiento de voz y facial, donde la inferencia rápida es esencial y la precisión del modelo puede ser ajustada para optimizar la experiencia del usuario.

3.3.4. Pruning y Técnicas de Reducción de Parámetros

La optimización de modelos de aprendizaje profundo es fundamental para su implementación en dispositivos móviles, donde los recursos son limitados. Entre las técnicas más importantes para lograr esta optimización se encuentran el pruning [43], la reducción de parámetros, y la compresión de modelos. Estas técnicas no solo permiten reducir el tamaño y la complejidad de las redes neuronales, sino que también facilitan su almacenamiento y ejecución eficiente en hardware con capacidades limitadas.

Pruning y reducción de parámetros

El pruning, o poda de modelos, es un proceso mediante el cual se eliminan partes de la red neuronal que tienen poca o ninguna influencia en la predicción final. Esto se puede hacer a nivel de pesos individuales, neuronas, o incluso capas enteras, dependiendo del enfoque utilizado.

- **Pruning basado en magnitud:** Elimina los pesos del modelo que tienen valores muy pequeños, bajo la suposición de que estos tienen una influencia mínima en el resultado.
- **Pruning estructurado:** A diferencia del pruning no estructurado, que elimina pesos individuales, este enfoque elimina componentes enteros como canales de convolución o neuronas completas, lo que facilita la implementación de modelos más rápidos y compactos.

- **Pruning dinámico:** En lugar de aplicar pruning estático durante o después del entrenamiento, el pruning dinámico ajusta el tamaño del modelo durante la inferencia en función de las necesidades de la tarea, permitiendo un compromiso entre precisión y eficiencia en tiempo real.

Comprensión de modelos

Además del pruning, la compresión de modelos [66] juega un papel crucial en la optimización para dispositivos móviles. Técnicas como la codificación de Huffman y la descomposición de matrices permiten reducir aún más el tamaño del modelo, facilitando su almacenamiento y ejecución en dispositivos con capacidades limitadas.

La **codificación de Huffman** [65] es una técnica de compresión sin pérdida que se utiliza para reducir la cantidad de bits necesarios para representar los pesos del modelo. Esta técnica asigna códigos más cortos a los valores de pesos que ocurren con mayor frecuencia, y códigos más largos a los que ocurren con menos frecuencia, lo que reduce el tamaño total del modelo sin afectar su precisión.

La **descomposición de matrices** [67], por otro lado, descompone las matrices de pesos en productos de matrices más pequeñas, lo que reduce el número de parámetros y, por lo tanto, el tamaño del modelo. Esto no solo facilita el almacenamiento del modelo, sino que también puede acelerar su ejecución al reducir el número de operaciones matemáticas necesarias durante la inferencia.

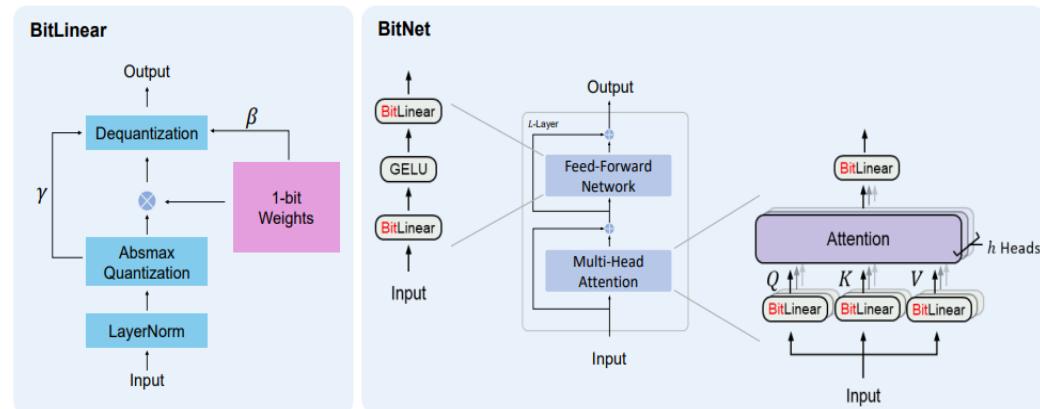


Figura 3.4: Arquitectura BitNet [38].

3.4. ESTADO ACTUAL DE LA TECNOLOGÍA

Aplicaciones en dispositivos móviles

La implementación exitosa de pruning y compresión de modelos en dispositivos móviles ha permitido una amplia gama de aplicaciones avanzadas. Por ejemplo, en sistemas de reconocimiento facial y de voz, la reducción de parámetros y la compresión del modelo son esenciales para asegurar que las aplicaciones funcionen de manera eficiente en tiempo real, sin agotar rápidamente los recursos del dispositivo. De manera similar, en aplicaciones de realidad aumentada, estas técnicas permiten que los modelos de visión por computador se ejecuten sin problemas, proporcionando experiencias inmersivas sin comprometer el rendimiento del dispositivo.

3.4. Estado actual de la tecnología

El campo de la visión por computador en dispositivos móviles ha experimentado avances significativos en los últimos años, impulsados por el rápido desarrollo de hardware especializado y la innovación en modelos de aprendizaje profundo, particularmente aquellos basados en Transformers. En este apartado, se examinan los principales desarrollos tecnológicos que han permitido la integración efectiva de estas tecnologías en dispositivos móviles, así como las tendencias actuales y los desafíos que persisten en este ámbito.

Hardware especializado para Inteligencia Artificial en dispositivos móviles

Uno de los pilares del estado actual de la tecnología es la evolución del hardware especializado en inteligencia artificial (IA) en dispositivos móviles. Los fabricantes de chips han desarrollado unidades de procesamiento neuronal (NPU, por sus siglas en inglés) y unidades de procesamiento gráfico (GPU) cada vez más potentes, diseñadas específicamente para manejar las cargas de trabajo de aprendizaje profundo de manera eficiente. Estas unidades permiten la ejecución de modelos complejos en tiempo real sin comprometer la duración de la batería o la experiencia del usuario.

Por ejemplo, los procesadores móviles de última generación, como los de la serie Qualcomm Snapdragon o los Apple A-series, incorporan NPUs dedicadas que aceleran las tareas de IA, como la visión por computador, permitiendo que aplicaciones como el reconocimiento facial, la realidad aumentada y la fotografía computacional funcionen de manera más fluida y eficiente.

CAPÍTULO 3. MARCO TEÓRICO

Modelos de aprendizaje profundo optimizados

Junto con el avance del hardware, ha habido un progreso significativo en el desarrollo de modelos de aprendizaje profundo optimizados para dispositivos móviles. Modelos como MobileNet, EfficientNet y las versiones reducidas de Vision Transformers (ViT) han sido diseñados para maximizar el rendimiento en entornos con recursos limitados. Estos modelos equilibran cuidadosamente la precisión y la eficiencia, permitiendo que tareas intensivas en cómputo, como la detección de objetos y la segmentación de imágenes, se ejecuten localmente en el dispositivo sin la necesidad de procesar los datos en la nube.

Además, las técnicas de compresión y optimización de modelos, como el pruning, la cuantización y la descomposición de matrices, han sido ampliamente adoptadas para reducir el tamaño de los modelos y acelerar su ejecución. Estas técnicas permiten que incluso los modelos de última generación puedan ser desplegados en dispositivos móviles, llevando la potencia de la IA a la palma de la mano del usuario.

Ecosistemas de desarrollo y herramientas de software

El estado actual de la tecnología también se ve reflejado en el desarrollo de ecosistemas de software robustos que facilitan la implementación de visión por computador en dispositivos móviles. Herramientas y frameworks como TensorFlow Lite, Core ML de Apple, y PyTorch Mobile han sido fundamentales para llevar modelos de aprendizaje profundo al entorno móvil. Estos frameworks están diseñados para simplificar el proceso de optimización y despliegue de modelos, ofreciendo soporte para las últimas técnicas de compresión y aceleración de hardware.

Las plataformas de desarrollo de IA están comenzando a ofrecer herramientas específicas para la creación de modelos ligeros y optimizados. Por ejemplo, TensorFlow Lite Model Maker permite a los desarrolladores entrenar y ajustar modelos de aprendizaje profundo de manera eficiente, optimizándolos automáticamente para su uso en dispositivos móviles.

Aplicaciones en el mundo real

La combinación de hardware especializado, modelos optimizados y herramientas de desarrollo ha permitido la aparición de aplicaciones avanzadas en el mundo real. Aplicaciones de realidad aumentada, como las utilizadas en juegos o en la navegación interior, han mejorado notablemente gracias a la capacidad de los dispositivos móviles para procesar imágenes y datos en tiempo real. Del mismo modo, los sistemas de seguridad basados en re-

3.4. ESTADO ACTUAL DE LA TECNOLOGÍA

conocimiento facial y los asistentes virtuales que dependen de la visión por computador han alcanzado un nuevo nivel de precisión y eficiencia.

Desafíos y perspectivas futuras

A pesar de estos avances, persisten varios desafíos en la integración de modelos de visión por computador en dispositivos móviles. La necesidad de equilibrar la precisión del modelo con la eficiencia sigue siendo un tema central, especialmente a medida que los usuarios demandan aplicaciones más sofisticadas que requieren capacidades de IA más avanzadas. Además, la seguridad y la privacidad de los datos son preocupaciones crecientes, dado que muchos de estos modelos procesan información personal sensible en el dispositivo.

Mirando hacia el futuro, se anticipa que la investigación continuará enfocándose en la creación de modelos aún más compactos y eficientes, capaces de manejar tareas más complejas con una mínima latencia y consumo energético. Asimismo, la evolución de la computación en el borde (edge computing) y la inferencia distribuida puede ofrecer nuevas soluciones para superar las limitaciones actuales, permitiendo que los dispositivos móviles no solo sean más inteligentes, sino también más autónomos.

Capítulo 4

Materiales

En esta sección se detallan los recursos y herramientas utilizados para el desarrollo de este proyecto. Estos incluyen tanto los recursos de hardware como los de software, así como los conjuntos de datos empleados para el entrenamiento y evaluación de los modelos de visión por computador basados en Transformers en dispositivos móviles.

4.1. Hardware

Dado que el objetivo de este trabajo es evaluar la viabilidad de modelos de visión por computador basados en Transformers en tiempo real en dispositivos móviles, se utilizaron tanto dispositivos móviles como estaciones de trabajo más potentes para el entrenamiento y optimización de los modelos. Los detalles específicos de los equipos son los siguientes:

4.1.1. Estación de Trabajo

Para el entrenamiento de los modelos, se empleó una estación de trabajo de alta capacidad con las siguientes especificaciones técnicas:

- **Procesador:** AMD Ryzen 9 5900X, con múltiples núcleos (12 núcleos) para optimizar el entrenamiento en paralelo de los modelos.
- **GPU:** NVIDIA RTX 3070, con capacidad de procesamiento masivo de operaciones matriciales y soporte para bibliotecas de deep learning como CUDA y cuDNN, fundamentales para la aceleración del entrenamiento de redes neuronales.

- **Memoria RAM:** 32 GB DDR4, para garantizar la gestión de grandes volúmenes de datos durante las fases de preprocesamiento y entrenamiento.
- **Almacenamiento:** SSD de 1 TB, lo que facilita la lectura y escritura rápida de datos durante el entrenamiento y experimentación.

Además, se emplearon dispositivos adicionales para el monitoreo del uso energético y el consumo de recursos del dispositivo móvil durante las pruebas, como medidores de consumo de batería y herramientas para la monitorización del rendimiento del hardware.

4.2. Software

El desarrollo y experimentación de este trabajo requerían un conjunto específico de software, incluyendo frameworks de deep learning, bibliotecas de optimización para dispositivos móviles y entornos de desarrollo adecuados.

4.2.1. Framework de Deep Learning

Se utilizó PyTorch [39], uno de los frameworks más avanzados y populares en la comunidad de aprendizaje profundo. Este framework ofrece compatibilidad con herramientas de optimización y despliegue en dispositivos móviles, como TensorFlow Lite (en caso de utilizar la librería de TensorFlow) y PyTorch Mobile, permitiendo una transición fluida desde el desarrollo en estaciones de trabajo hasta la implementación en dispositivos móviles. Las versiones utilizadas fueron:

- **PyTorch:** Versión CUDA 12.1 para Linux, con soporte para la exportación a ONNX y posterior conversión a ONNX Runtime para dispositivos móviles.

4.2.2. Herramientas de Optimización

Para asegurar un rendimiento eficiente de los modelos en dispositivos móviles, se emplearon diversas herramientas de optimización:

- **TorchScript:** Herramienta [40] utilizada para convertir modelos entrenados en PyTorch en una representación más eficiente que se puede ejecutar independientemente de Python, permitiendo la optimización para dispositivos móviles mediante técnicas como la cuantización y el trazado de gráficos computacionales.

- **ONNX Runtime:** Utilizado para convertir y ejecutar modelos entrenados en PyTorch [41], facilitando la compatibilidad con múltiples dispositivos móviles a través de optimizaciones específicas para hardware ARM. PyTorch tiene soporte nativo para exportar modelos a ONNX.
- **PyTorch Quantization Toolkit:** Biblioteca [42] empleada para aplicar técnicas de cuantización, como la cuantización estática y dinámica, reduciendo el tamaño y los requisitos computacionales de los modelos mientras se mantiene un nivel aceptable de precisión.
- **PyTorch Pruning:** Extensión [43] que permite realizar pruning en los modelos entrenados, reduciendo el número de parámetros y mejorando la eficiencia computacional sin una pérdida considerable en la precisión del modelo.

4.2.3. Entorno de Desarrollo

El código fue desarrollado en un entorno de programación basado en Ubuntu 20.04 [44], utilizando el IDE VSCode [45] como el entorno de desarrollo integrado principal. Este entorno facilitó la depuración y el despliegue del código en el dispositivo móvil mediante extensiones específicas para PyTorch.

4.3. Conjuntos de Datos

Los conjuntos de datos empleados en este proyecto fueron seleccionados tanto por su relevancia en tareas de visión por computador como por su diversidad y aplicabilidad en escenarios del mundo real. Estos conjuntos de datos fueron utilizados tanto para el entrenamiento como para la evaluación de los modelos.

4.3.1. Conjunto de Datos Principal

El principal conjunto de datos utilizado fue SIDD (Smartphone Image Denoising Dataset) [46], ampliamente reconocido en la comunidad de visión por computador por su enfoque en la reducción de ruido en imágenes capturadas por dispositivos móviles. Este conjunto de datos contiene más de 30,000 pares de imágenes con ruido y sus respectivas imágenes limpias de referencia, capturadas en diversas condiciones de iluminación y con diferentes teléfonos inteligentes. Las características relevantes de este conjunto de datos incluyen:

CAPÍTULO 4. MATERIALES

- **Imágenes de alta diversidad:** Las imágenes incluyen una amplia variedad de escenas y condiciones de iluminación, lo que permite entrenar modelos de eliminación de ruido que sean robustos y generalizables a diferentes entornos de captura.
- **Anotaciones precisas:** Las imágenes están acompañadas de sus respectivas versiones limpias (sin ruido), lo que facilita el entrenamiento supervisado de modelos de reducción de ruido basados en aprendizaje profundo.

4.3.2. Conjunto de Datos Adicional

Para evaluar el rendimiento de los modelos en escenarios específicos de desenfoque por movimiento en dispositivos móviles, se utilizó un conjunto de datos adicional, el GOPRO dataset [47], que incluye imágenes capturadas en condiciones de movimiento y desenfoque. Este conjunto de datos cuenta con las siguientes características:

- **Resolución alta y variada:** Imágenes de alta resolución capturadas con cámaras GoPro, que permiten evaluar la capacidad de los modelos para procesar imágenes de alta calidad y adaptarse a diferentes resoluciones durante el preprocesamiento.
- **Desenfoque por movimiento:** Imágenes capturadas durante situaciones de movimiento rápido, lo que simula condiciones de uso en aplicaciones móviles como grabación de video, deportes o videovigilancia, donde el desenfoque es común.

Este conjunto de datos permitió realizar pruebas de robustez y determinar si los modelos optimizados podían manejar el desenfoque por movimiento sin sacrificar la precisión.

Capítulo 5

Métodos

Este apartado describe de manera detallada el proceso de implementación, optimización y evaluación de los modelos de visión por computador basados en Transformers en dispositivos móviles. La metodología se estructuró en varias fases: desde la selección de los modelos hasta su optimización para dispositivos móviles y su evaluación en condiciones reales.

5.1. Implementación de Modelos Basados en Transformers

La implementación de los modelos se dividió en varias etapas clave, comenzando con la selección y modificación de arquitecturas basadas en Transformers y pasando por el preprocesamiento de datos y el entrenamiento de los modelos en estaciones de trabajo.

5.1.1. Selección de Modelos

Se optó por implementar Vision Transformers [11] que han demostrado ser altamente efectivos en tareas de visión por computador. Se caracteriza por dividir la imagen en pequeños parches, los cuales son tratados como *tokens* de entrada para el Transformer. La arquitectura ViT es puramente basada en Transformers, sin el uso de convoluciones, lo que la distingue de modelos tradicionales de visión por computador.

5.1.2. Preprocesamiento de Datos

Antes del entrenamiento, se realizaron varias técnicas de preprocesamiento [48] de los datos para preparar las imágenes del conjunto de datos:

CAPÍTULO 5. MÉTODOS

- **Redimensionamiento:** Todas las imágenes fueron redimensionadas a 224x224 píxeles, lo que garantiza la consistencia del tamaño de los *tokens* utilizados en Vision Transformers.
- **Normalización:** Se aplicó la normalización de los valores de los píxeles entre 0 y 1 utilizando los valores de media y desviación estándar propios del conjunto de datos, como ocurre en ImageNet.
- **Augmentación de Datos:** Para mejorar la robustez del modelo, se emplearon técnicas de augmentación de datos, como rotaciones, recortes, reflejos horizontales y cambios en el brillo, lo que aumenta la variabilidad del conjunto de entrenamiento.

5.1.3. Entrenamiento de los Modelos

Los modelos seleccionados fueron entrenados [50] en la estación de trabajo descrita previamente, con los siguientes parámetros:

- **Función de pérdida:** Se utilizó la función de pérdida categorical cross-entropy para tareas de clasificación, y IoU (Intersection over Union) para la detección de objetos.
- **Optimizador:** Se empleó el optimizador Adam con una tasa de aprendizaje inicial de 0.001, junto con una estrategia de decay para reducirla progresivamente durante el entrenamiento.
- **Batch size:** El tamaño de lote se estableció en 32, optimizado para equilibrar la capacidad de procesamiento de la GPU con la velocidad de convergencia del modelo.
- **Épocas:** Los modelos fueron entrenados durante un total de 100 épocas, con validación periódica en cada 10 época para prevenir el sobreajuste (overfitting).

Los modelos entrenados se almacenaron en formatos estándar para su posterior conversión y optimización en dispositivos móviles.

5.2. Optimización para Dispositivos Móviles

Uno de los principales desafíos de este trabajo fue adaptar los modelos basados en Transformers, que típicamente requieren recursos computacionales elevados, para que pudieran ejecutarse de manera eficiente en dispositivos móviles con recursos limitados. Para ello, se implementaron varias técnicas de optimización.

5.2. OPTIMIZACIÓN PARA DISPOSITIVOS MÓVILES

5.2.1. Conversión a Formato Móvil

Los modelos fueron convertidos a formatos optimizados para su ejecución en dispositivos móviles, utilizando las siguientes herramientas:

- **TorchScript**: Esta herramienta [40] permitió convertir los modelos de PyTorch a un formato compacto y eficiente, optimizado para su uso en dispositivos con hardware limitado, como teléfonos móviles, permitiendo además su ejecución sin la necesidad de un intérprete de Python.
- **ONNX Runtime**: Para los modelos entrenados en PyTorch, se utilizó ONNX (Open Neural Network Exchange) [41] para exportar el modelo y hacerlo compatible con múltiples plataformas, incluyendo dispositivos móviles con arquitectura ARM, facilitando la optimización y portabilidad.

5.2.2. Cuantización

Una de las técnicas más importantes aplicadas fue la cuantización [42], que reduce la precisión de los pesos del modelo desde 32 bits de coma flotante a 16 bits o 8 bits sin perder significativamente la precisión en las predicciones.

- **Cuantización híbrida**: En algunos casos, solo ciertos componentes del modelo fueron cuantizados, mientras que las operaciones más sensibles a la precisión mantuvieron su formato original de 32 bits.

5.2.3. Pruning

Para reducir la complejidad de los modelos y mejorar la eficiencia en dispositivos móviles, se aplicó el pruning [43] o poda de parámetros, que consiste en eliminar conexiones no esenciales dentro de la red neuronal. Esta técnica redujo tanto el tamaño del modelo como la cantidad de operaciones requeridas, sin comprometer significativamente su precisión.

5.2.4. Modelos Binarios

En algunas configuraciones, se implementaron modelos binarios mediante la técnica de binarización de pesos [7], donde los pesos del modelo se limitan a valores binarios (-1, 1), lo que reduce drásticamente los requerimientos de almacenamiento y procesamiento, a cambio de una ligera pérdida de precisión.

Capítulo 6

Resultados

En este capítulo se presentan los resultados obtenidos a partir de los experimentos realizados con modelos Vision Transformer (ViT) aplicados a tareas de visión por computador, tanto en su versión entrenada desde cero como a través de técnicas de transfer learning. Se evaluarán los rendimientos de ambos enfoques en función de métricas clave de desempeño, destacando las diferencias en cuanto a precisión y eficiencia.

Además, se abordarán diversas técnicas de optimización enfocadas a la reducción del tamaño del modelo y la mejora de su eficiencia en dispositivos móviles, con el objetivo de facilitar su implementación en entornos con recursos limitados. En particular, se expondrá cómo se puede introducir un modelo optimizado en un dispositivo móvil, analizando el proceso de conversión a formatos específicos y las consideraciones necesarias para su integración.

Finalmente, se proporcionará una comparativa entre los enfoques, permitiendo extraer conclusiones respecto a la eficacia de los modelos desarrollados y su viabilidad en aplicaciones móviles en tiempo real.

6.1. Arquitectura desde cero (From Scratch)

En este apartado se analizan los resultados obtenidos al implementar una arquitectura de Vision Transformer (ViT) diseñada desde cero [49], replicando en código la propuesta presentada en el artículo científico “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale” [11]. Esta implementación no utiliza pesos preentrenados, sino que el modelo se entrena directamente sobre el conjunto de datos seleccionado. A continuación, se detallan los principales aspectos del modelo, incluyendo su diseño, entrenamiento y las métricas de desempeño obtenidas a lo largo de las épocas.

El análisis de los resultados se centrará en la capacidad del modelo para

CAPÍTULO 6. RESULTADOS

generalizar a los datos, observando su comportamiento tanto en términos de precisión como de eficiencia computacional.

6.1.1. Descripción del proceso

El primer paso en la implementación del modelo Vision Transformer from scratch es la descarga y preparación del conjunto de datos. En este caso, se ha utilizado el SIDD Small Dataset, un conjunto de imágenes en formato sRGB, que contiene imágenes limpias (GT) y con ruido (Noisy).

El código comienza estableciendo las rutas donde se almacenarán los datos y, si estos no están disponibles localmente, procede a descargarlos de una fuente en línea. Una vez descargado el archivo comprimido, el código lo descomprime en la ubicación adecuada.

Posteriormente, el código recorre las carpetas del conjunto de datos para localizar las imágenes correspondientes a las versiones GT y Noisy de cada caso. Estas imágenes se almacenan en listas separadas para su uso posterior en el entrenamiento del modelo. Además, se verifica que tanto las imágenes GT como las Noisy estén presentes en cada carpeta antes de agregarlas a las listas.

Finalmente, se muestra una vista preliminar de los primeros elementos de las listas, así como el número total de imágenes cargadas, garantizando que los datos están listos para el proceso de entrenamiento.

```
GT Image Paths: [PosixPath('data/sidd_small_dataset/SIDD_Small_sRGB_Only/Data/0140_006_IP_00800_00800_3200_L/GT_SRGB_010.PNG'), PosixPath('data/sidd_small_dataset/SIDD_Small_sRGB_Only/Data/0140_006_IP_00800_00800_3200_L/NOISY_SRGB_010.PNG')], PosixPath('data/sidd_small_dataset/SIDD_Small_sRGB_Only/Data/0140_006_IP_00800_00800_3200_L/NOISY_SRGB_010.PNG')]  
Noisy Image Paths: [PosixPath('data/sidd_small_dataset/SIDD_Small_sRGB_Only/Data/0140_006_IP_00800_00800_3200_L/NOISY_SRGB_010.PNG')], PosixPath('data/sidd_small_dataset/SIDD_Small_sRGB_Only/Data/0140_006_IP_00800_00800_3200_L/NOISY_SRGB_010.PNG')]  
GT Image Paths size: 160  
Noisy Image Paths size: 160
```

Figura 6.1: Vista preliminar del conjunto de datos SIDD Small Dataset.

Se define y prepara el conjunto de datos para ser utilizado en el entrenamiento del modelo. Para ello, se crea una clase personalizada llamada SIDDIImageDataset que extiende la clase Dataset de PyTorch. Esta clase permite cargar las imágenes de las listas de rutas de imágenes GT y Noisy, y aplicar transformaciones opcionales, como el redimensionamiento y la conversión a tensores.

- **Transformaciones:** Se definen transformaciones que incluyen el redimensionamiento de las imágenes a un tamaño estándar de 224x224 píxeles y la conversión de las imágenes a tensores.
- **DataLoader:** Se utiliza un DataLoader que facilita el manejo de los datos en lotes (batches) durante el entrenamiento, permitiendo cargar un conjunto de 32 imágenes en cada iteración de manera aleatoria.

6.1. ARQUITECTURA DESDE CERO (FROM SCRATCH)

Finalmente, el código ejecuta una iteración de prueba para verificar que tanto las imágenes GT como las Noisy se cargan correctamente en un batch, mostrando la forma (dimensiones) de los tensores.

```
Batch de imágenes ruidosas: torch.Size([32, 3, 224, 224])
Batch de imágenes GT: torch.Size([32, 3, 224, 224])
```

Figura 6.2: Dimensiones de los tensores para el entrenamiento.

Se decide a seleccionar y visualizar imágenes aleatorias desde el conjunto de datos de imágenes limpias (GT) y con ruido. Para ello, se selecciona de manera aleatoria una imagen del conjunto, tanto la versión sin ruido (GT) como la correspondiente imagen con ruido. A continuación, se obtienen y muestran los metadatos básicos de estas imágenes, como sus dimensiones. Esto permite observar la calidad de las imágenes y verificar que ambas versiones corresponden entre sí.

```
Random image path: data/sidd_small_dataset/SIDD_Small_sRGB_Only/Data/0194_009_IP_01600_04000_3200_N/GT_SRGB_010.PNG
Image height: 3024
Image width: 4032

Random image path: data/sidd_small_dataset/SIDD_Small_sRGB_Only/Data/0194_009_IP_01600_04000_3200_N/NOISY_SRGB_010.PNG
Image height: 3024
Image width: 4032
```

Figura 6.3: Dimensiones de las imágenes aleatorias para su visualización.

Finalmente, se visualizan ambas imágenes para hacer una comparación visual directa (Figura 6.4). Esto es especialmente útil para evaluar subjetivamente cómo el ruido afecta a la imagen y sirve como una validación preliminar de la estructura y formato del conjunto de datos antes de pasar a fases más avanzadas, como el entrenamiento del modelo.

A continuación, se procede a preparar las imágenes para poder introducirlas al modelo para su entrenamiento. Para ello, se sigue la recomendación del artículo científico, convirtiendo las imágenes en pequeños batches que se utilizarán en el modelo ViT con sus respectivas posiciones. Primero, se visualiza una imagen específica seleccionada aleatoriamente del lote de imágenes ruidosas. Se verifican las dimensiones de la imagen seleccionada, lo que proporciona información sobre la estructura de los datos (número de canales, altura y ancho).

Luego, se utiliza la biblioteca de visualización matplotlib para mostrar la imagen. Es importante notar que se realiza una permutación de los ejes de la imagen antes de la visualización, lo que es necesario para que la imagen se muestre correctamente en el formato que espera la función de visualización. Este paso es crucial, ya que las imágenes en PyTorch suelen tener el formato de tensor en el que el primer eje representa el número de canales, seguido de

CAPÍTULO 6. RESULTADOS

la altura y el ancho, mientras que la visualización requiere que el formato sea de altura por ancho por canales.

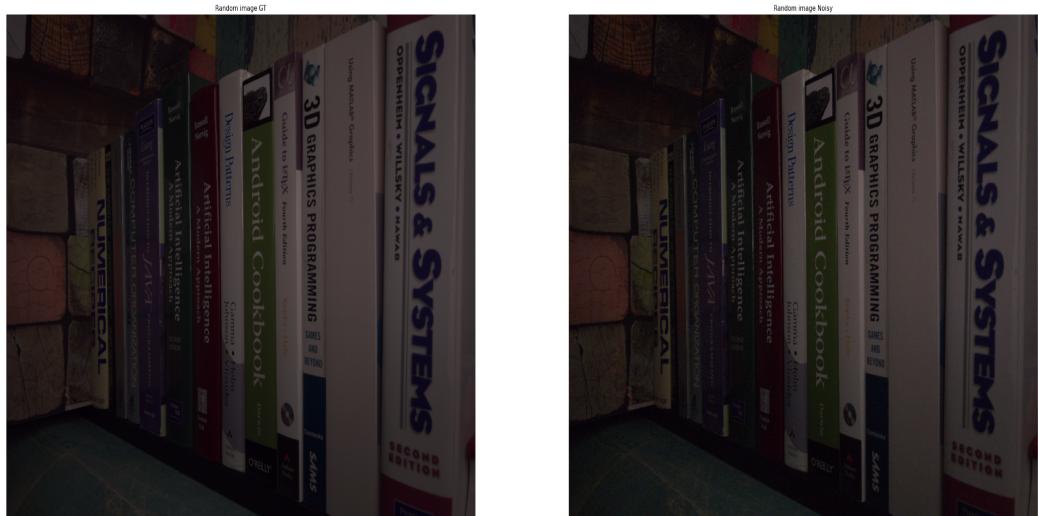


Figura 6.4: Visualización de imágenes GT y ruido.

La visualización permite al investigador inspeccionar cómo se ve la imagen con ruido, lo que es útil para validar visualmente los datos antes de proceder con el entrenamiento del modelo.



Figura 6.5: Visualización de imagen con ruido aleatoria.

Para entender cómo se separa la imagen en parches para introducir al modelo, se procede a mostrar cada paso necesario de forma visual. Primero,

6.1. ARQUITECTURA DESDE CERO (FROM SCRATCH)

se reorganiza la imagen para que los canales de color se ubiquen al final de la estructura de datos, lo que la convierte en un formato adecuado para la visualización.

Se establece un tamaño de parche de 16 píxeles, lo que indica que se va a visualizar una porción de la imagen que abarca las primeras 16 filas de píxeles. Este enfoque permite observar de manera más detallada la variación de los valores de píxel en esta área específica de la imagen, lo que puede ser útil para entender mejor las características del ruido presente en la imagen.

Finalmente, se utiliza una función de visualización para mostrar esta sección de la imagen. Al enfocarse en una pequeña región, se pueden identificar patrones de ruido que pueden ser relevantes para el proceso de denoising que se está llevando a cabo en el modelo.

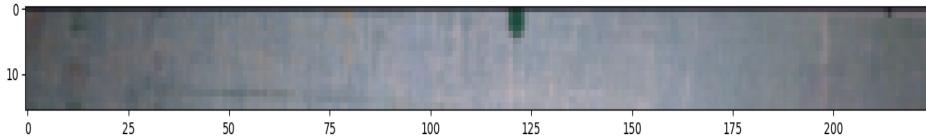


Figura 6.6: Sección superior de imagen con ruido aleatoria.

Se divide la imagen en parches y se visualiza la parte superior de la imagen en una serie de subgráficas. Se establecen varios parámetros para la visualización de estos parches.

Primero, se define el tamaño de la imagen y el tamaño de los parches, junto con el número total de parches en una fila. Se incluye una aserción para asegurarse de que el tamaño de la imagen sea divisible por el tamaño del parche, lo que garantiza que no habrá problemas al dividir la imagen en secciones iguales.

Luego, se crea una serie de subgráficas donde cada columna corresponde a un parche de la imagen. Esto permite una visualización clara y estructurada de cómo se ve cada sección de la parte superior de la imagen.

A continuación, el código itera sobre el número de parches en la fila superior y visualiza cada uno en las subgráficas creadas previamente. Cada parche se etiqueta numéricamente para facilitar la identificación, mientras que se eliminan las marcas de los ejes para evitar distracciones en la visualización. Esta técnica es útil para estudiar la composición de la imagen y entender cómo se distribuyen los datos en pequeñas regiones, lo que es relevante para el entrenamiento y la interpretación de modelos de visión por computador, como el Vision Transformer.

CAPÍTULO 6. RESULTADOS

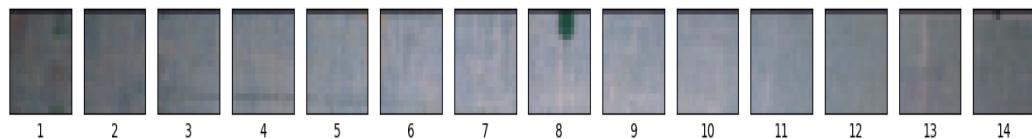


Figura 6.7: Sección superior de imagen con ruido aleatoria separada en parches.

Una vez comprendido cuál es el procedimiento visto para una sección de la imagen, se procede a dividir y visualizar en un formato de cuadrícula la imagen al completo (Figura 6.8). Permite observar la imagen completa como una colección de sus partes.



Figura 6.8: Imagen aleatoria completa con ruido separada en parches.

Aquí están los elementos clave:

- **Definición de parámetros:** Se establece el tamaño de la imagen (224x224 píxeles) y el tamaño de los parches (16x16 píxeles). Se calcula el número total de parches tanto en filas como en columnas, y se incluye una asercción para garantizar que el tamaño de la imagen sea divisible por el tamaño de los parches.

6.1. ARQUITECTURA DESDE CERO (FROM SCRATCH)

- **Creación de subgráficas:** Se generan subgráficas en una cuadrícula que refleja la división de la imagen en parches. El número de filas y columnas se determina según la cantidad de parches calculada previamente.
- **Iteración sobre parches:** A través de bucles anidados, se itera sobre la altura y el ancho de la imagen, y se extraen y visualizan los parches correspondientes en las subgráficas. Esto permite que cada subgráfica represente un pequeño segmento de la imagen.
- **Etiquetas y presentación:** Se configuran las etiquetas para cada subgráfica, que indican la posición del parche (número de fila y columna). Se eliminan las marcas de los ejes para que la atención se centre en los parches. Finalmente, se establece un título para la visualización que indica que se está mostrando la imagen en forma de parches.

Este tipo de visualización es esencial para entender cómo el modelo Vision Transformer procesa las imágenes al dividirlas en segmentos más pequeños, lo que facilita la captura de características relevantes de manera más eficiente.

Ahora, se procede a transformar los parches generados anteriormente en embeddings a través de una capa convolucional. De esta manera, se generará un mapa de características que el modelo comprenderá mejor y mejorará el proceso de generación de nuevas imágenes. A continuación se describen los elementos clave:

- **Configuración de la capa convolucional:** Se define una capa convolucional 2D con parámetros específicos que son relevantes para el modelo Vision Transformer (ViT). Esta capa está diseñada para recibir imágenes de entrada (en este caso, imágenes en color con 3 canales) y generar mapas de características (embeddings) de un tamaño específico. Los parámetros incluyen:
 - **in-channels:** Número de canales de entrada (3 para imágenes RGB).
 - **out-channels:** Número de mapas de características que se generarán (768, basado en el tamaño D del modelo ViT-Base).
 - **kernel-size y stride:** Definen el tamaño de los parches y el paso con el que se aplican.
- **Propagación a través de la capa convolucional:** La imagen se pasa a través de la capa convolucional, añadiendo primero una dimensión de lote (batch dimension) para que la forma del tensor sea compatible

CAPÍTULO 6. RESULTADOS

con la capa. El resultado es un tensor que contiene los embeddings generados para cada parche de la imagen.

- **Visualización de mapas de características:** Se seleccionan índices aleatorios de los mapas de características generados para visualizarlos. Se utiliza “random.sample” para obtener un subconjunto de índices y luego se crean subgráficas para mostrar cada uno de los mapas de características. Esto permite observar cómo la capa convolucional ha extraído diferentes características de la imagen original en forma de matrices de activación.
- **Visualización de resultados:** Para cada mapa de características seleccionado, se visualiza su representación, eliminando la dimensión del lote y asegurando que no se rastreen los gradientes al pasar a formato numpy para la visualización. Esto proporciona una comprensión visual de cómo el modelo está procesando la imagen a nivel de parches y qué características están siendo aprendidas en el proceso.

Esta parte del código es crucial para entender cómo el modelo ViT transforma imágenes en características que pueden ser utilizadas en etapas posteriores del proceso de generación de nuevas imágenes.

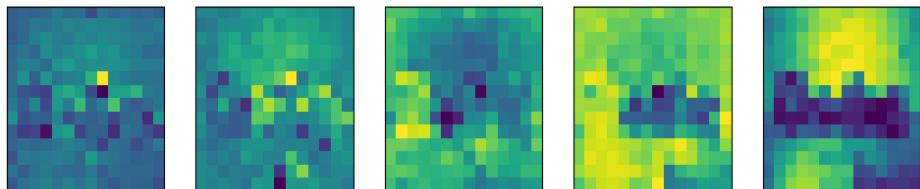


Figura 6.9: Mapa de características de la imagen aleatoria.

Una vez analizados los mapas de características, se aplana los embeddings de los parches y reorganizan las dimensiones para preparar los datos de entrada que se introducirán en el Transformer. Después de generar los mapas de características mediante la capa convolucional, el siguiente paso es aplinar esos mapas. Esto se realiza con la función `torch.nn.Flatten()`, que convierte las dimensiones de altura y ancho de los mapas de características en un vector unidimensional, preparando así los datos como secuencias de embeddings. Estos embeddings representan cada parche de la imagen y contienen la información aprendida por la convolución, lo cual es crucial para que el modelo Transformer procese las imágenes de manera eficiente.

6.1. ARQUITECTURA DESDE CERO (FROM SCRATCH)

Tras aplanar los mapas de características, se reorganizan las dimensiones del tensor para adecuarlos a la forma requerida por el modelo Vision Transformer: (tamaño del batch, número de parches, dimensión del embedding). Esta reorganización asegura que cada parche de la imagen esté representado como un vector que contiene las características aprendidas, facilitando su procesamiento por la capa de atención del Transformer.

Además, se realiza una visualización del mapa de características aplanado para obtener una representación gráfica de los resultados. Este gráfico ayuda a comprender cómo las características de la imagen se transforman en parches y se reestructuran para ser utilizadas en el modelo Transformer. También se muestra la imagen original, permitiendo comparar el proceso de conversión de características con el aspecto visual inicial de la imagen. Este paso es esencial para preparar las imágenes de entrada que serán utilizadas por el Transformer en el proceso de aprendizaje.



Figura 6.10: Mapa de características aplanado.

Se define una clase PatchEmbedding, que implementa la capa de embeddings de parches en un Vision Transformer. La capa utiliza una convolución para dividir una imagen en parches de tamaño fijo y generar embeddings, que luego se aplanan y reorganizan para que el modelo Transformer pueda procesarlos. El método forward realiza la pasada hacia adelante, asegurando que las dimensiones de la imagen sean correctas y devolviendo una secuencia de embeddings.

Ahora, se implementan dos conceptos importantes en Vision Transformers: el class token y el position embedding. Primero, se crea el class token, que es un parámetro aprendible del mismo tamaño que los embeddings de los parches. Este token se concatena al inicio de la secuencia de parches para capturar la información global de la imagen. Luego, se crea el position embedding, una representación aprendible que indica la posición de cada parche en la imagen, lo que permite al modelo entender el orden espacial de los parches. Finalmente, se suma el position embedding a los embeddings de parches y del class token, completando el proceso de preparación de entradas para el modelo Transformer.

La siguiente parte del código implementa un bloque de atención multi-cabeza (MultiHead Self-Attention, MSA), un componente clave en la arquitectura del Vision Transformer (ViT). Se define la clase MultiHeadSelfAttentionBlock, que hereda de nn.Module. En el constructor, se inicializan la normalización de capas (LayerNorm), que estabiliza y acelera el entrenamiento,

CAPÍTULO 6. RESULTADOS

y la capa de atención multi-cabeza (MultiheadAttention). Esta última permite al modelo captar diferentes aspectos de las relaciones entre los parches de imagen al tener múltiples “cabezas” de atención, extrayendo características complementarias.

En el método forward, la entrada se normaliza y se pasa a la capa de atención, utilizando el mismo tensor como consulta, clave y valor, lo que es típico en la auto-atención. Finalmente, se crea una instancia del bloque de atención y se pasa la secuencia de embeddings de parches y posiciones a través de él. Se imprimen las formas de entrada y salida para verificar que la transformación se ha realizado correctamente y que las dimensiones son las esperadas, asegurando la coherencia dimensional a lo largo del procesamiento del modelo.

Se implementa un bloque de perceptrón multicapa (MLP) como parte de la arquitectura del Vision Transformer (ViT). Se define la clase MLPBlock, que hereda de nn.Module. En el constructor de la clase, se inicializa una capa de normalización (LayerNorm) que ayuda a estabilizar el proceso de entrenamiento al normalizar las activaciones.

Luego, se crea un modelo MLP mediante una secuencia de capas. Este modelo incluye una capa lineal que transforma las características de entrada del tamaño del embedding (768) a un tamaño más grande (3072), seguida de una función de activación GELU (Gaussian Error Linear Unit) que introduce no linealidades, y una capa de dropout que se utiliza para prevenir el sobreajuste. Después, se añade otra capa lineal que reduce nuevamente las dimensiones al tamaño del embedding original, junto con otra capa de dropout.

En el método forward, se normalizan las entradas y se pasan a través del MLP. Finalmente, se crea una instancia del bloque MLP y se alimenta la salida del bloque de atención multi-cabeza (MSA) a través de él, imprimiendo las formas de entrada y salida para confirmar que las dimensiones se mantienen consistentes durante el procesamiento.

Por último, se implementa un bloque de codificador Transformer mediante la clase TransformerEncoderBlock, que hereda de nn.Module. En el constructor, se inicializan dos componentes clave: el bloque de atención multi-cabeza (msa-block) y el bloque de perceptrón multicapa (mlp-block). Ambos utilizan parámetros que corresponden a la arquitectura ViT-Base, como el tamaño del embedding (768), el número de cabezas de atención (12) y el tamaño de la MLP (3072).

En el método forward, se aplica la atención multi-cabeza y se añade una conexión residual (skip connection), que ayuda a evitar la pérdida de información durante el paso de los datos a través del bloque. A continuación, se pasa la salida a través del bloque MLP y se aplica otra conexión residual.

6.1. ARQUITECTURA DESDE CERO (FROM SCRATCH)

Finalmente, se crea una instancia de TransformerEncoderBlock, y se utiliza la función summary de la biblioteca torchinfo para obtener un resumen de la arquitectura del bloque. Este resumen incluye información sobre el tamaño de entrada y salida, el número de parámetros y la cantidad de parámetros que se pueden entrenar, facilitando la comprensión de la complejidad del modelo y sus características.

Layer (type (var_name))	Input Shape	Output Shape	Param #	Trainable
TransformerEncoderBlock (TransformerEncoderBlock)	[1, 197, 768]	[1, 197, 768]	--	True
└ MultiHeadSelfAttentionBlock (msa_block)	[1, 197, 768]	[1, 197, 768]	--	True
└ LayerNorm (layer_norm)	[1, 197, 768]	[1, 197, 768]	1,536	True
└ MultiheadAttention (multihead_attn)	--	[1, 197, 768]	2,362,368	True
└ MLPBlock (mlp_block)	[1, 197, 768]	[1, 197, 768]	--	True
└ LayerNorm (layer_norm)	[1, 197, 768]	[1, 197, 768]	1,536	True
└ Sequential (mlp)	[1, 197, 768]	[1, 197, 768]	--	True
└ Linear (0)	[1, 197, 768]	[1, 197, 3072]	2,362,368	True
└ GELU (1)	[1, 197, 3072]	[1, 197, 3072]	--	--
└ Dropout (2)	[1, 197, 3072]	[1, 197, 3072]	--	--
└ Linear (3)	[1, 197, 3072]	[1, 197, 768]	2,360,064	True
└ Dropout (4)	[1, 197, 768]	[1, 197, 768]	--	--
Total params:	7,087,872			
Trainable params:	7,087,872			
Non-trainable params:	0			
Total mult-adds (Units.MEGABYTES):	4.73			
Input size (MB):	0.61			
Forward/backward pass size (MB):	8.47			
Params size (MB):	18.90			
Estimated Total Size (MB):	27.98			

Figura 6.11: Resumen de la arquitectura del bloque TransformerEncoderBlock.

Se define la clase ViT (Vision Transformer), que implementa un modelo de Transformer para imágenes. En el constructor, se establecen parámetros como el tamaño de la imagen, el número de capas del Transformer, el tamaño de los embeddings y las tasas de dropout, asegurando que el tamaño de la imagen sea divisible por el tamaño del parche.

Se crean embeddings de clase y de posición como parámetros aprendibles, junto con un dropout para los embeddings. Luego, se instancian los bloques de patch embedding y los bloques del codificador Transformer.

En el método forward, se procesa la entrada: se genera una token de clase, se crean los embeddings de parches y se concatenan. Se suman los embeddings de posición y se aplica el dropout antes de pasar la salida a través de los bloques del Transformer. También hay una sección comentada para un cabezal de clasificador, que se puede activar para tareas de clasificación en visión por computador.

Haciendo uso de la clase ViT explicada, se obtiene un resumen visual del modelo ViT (Vision Transformer) utilizando la biblioteca torchinfo. Primero, se define un tensor de embeddings de clase, que se expande para que coincida

CAPÍTULO 6. RESULTADOS

con un tamaño de lote de 32. Esto demuestra cómo se puede adaptar un embedding de clase para múltiples imágenes.

Luego, se genera un tensor de imagen aleatorio con la misma forma que una imagen de entrada y se crea una instancia del modelo ViT. Al pasar este tensor aleatorio a la instancia de ViT, se activan las capas del modelo, preparándolo para la inferencia.

Para mostrar un resumen detallado del modelo ViT, se utiliza `summary` de `torchinfo`, que incluye la forma de entrada y salida, el número de parámetros y cuántos de ellos son entrenables. Esto proporciona una visión clara de la arquitectura y la complejidad del modelo, facilitando la comprensión de sus componentes y su estructura.

Layer (type (var_name))	Input Shape	Output Shape	Param #	Trainable
VIT (ViT)				
-PatchEmbedding (patch_embedding)				
└Conv2d (patcher)	[1, 3, 224, 224]	[1, 196, 768]	--	True
└Flatten (flatten)	[1, 3, 224, 224]	[1, 768, 14, 14]	590,592	True
-Dropout (embedding_dropout)	[1, 768, 14, 14]	[1, 768, 196]	--	--
-Sequential (transformer_encoder)	[1, 197, 768]	[1, 197, 768]	--	--
└TransformerEncoderBlock (0)	[1, 197, 768]	[1, 197, 768]	--	True
└MultiHeadSelfAttentionBlock (msa_block)	[1, 197, 768]	[1, 197, 768]	2,363,904	True
└MLPBlock (mlp_block)	[1, 197, 768]	[1, 197, 768]	4,723,968	True
└TransformerEncoderBlock (1)	[1, 197, 768]	[1, 197, 768]	--	True
└MultiHeadSelfAttentionBlock (msa_block)	[1, 197, 768]	[1, 197, 768]	2,363,904	True
└MLPBlock (mlp_block)	[1, 197, 768]	[1, 197, 768]	4,723,968	True
└TransformerEncoderBlock (2)	[1, 197, 768]	[1, 197, 768]	--	True
└MultiHeadSelfAttentionBlock (msa_block)	[1, 197, 768]	[1, 197, 768]	2,363,904	True
└MLPBlock (mlp_block)	[1, 197, 768]	[1, 197, 768]	4,723,968	True
└TransformerEncoderBlock (3)	[1, 197, 768]	[1, 197, 768]	--	True
└MultiHeadSelfAttentionBlock (msa_block)	[1, 197, 768]	[1, 197, 768]	2,363,904	True
└MLPBlock (mlp_block)	[1, 197, 768]	[1, 197, 768]	4,723,968	True
└TransformerEncoderBlock (4)	[1, 197, 768]	[1, 197, 768]	--	True
└MultiHeadSelfAttentionBlock (msa_block)	[1, 197, 768]	[1, 197, 768]	2,363,904	True
└MLPBlock (mlp_block)	[1, 197, 768]	[1, 197, 768]	4,723,968	True
└TransformerEncoderBlock (5)	[1, 197, 768]	[1, 197, 768]	--	True
└MultiHeadSelfAttentionBlock (msa_block)	[1, 197, 768]	[1, 197, 768]	2,363,904	True
└MLPBlock (mlp_block)	[1, 197, 768]	[1, 197, 768]	4,723,968	True
└TransformerEncoderBlock (6)	[1, 197, 768]	[1, 197, 768]	--	True
└MultiHeadSelfAttentionBlock (msa_block)	[1, 197, 768]	[1, 197, 768]	2,363,904	True
└MLPBlock (mlp_block)	[1, 197, 768]	[1, 197, 768]	4,723,968	True
└TransformerEncoderBlock (7)	[1, 197, 768]	[1, 197, 768]	--	True
└MultiHeadSelfAttentionBlock (msa_block)	[1, 197, 768]	[1, 197, 768]	2,363,904	True
└MLPBlock (mlp_block)	[1, 197, 768]	[1, 197, 768]	4,723,968	True
└TransformerEncoderBlock (8)	[1, 197, 768]	[1, 197, 768]	--	True
└MultiHeadSelfAttentionBlock (msa_block)	[1, 197, 768]	[1, 197, 768]	2,363,904	True
└MLPBlock (mlp_block)	[1, 197, 768]	[1, 197, 768]	4,723,968	True
└TransformerEncoderBlock (9)	[1, 197, 768]	[1, 197, 768]	--	True
└MultiHeadSelfAttentionBlock (msa_block)	[1, 197, 768]	[1, 197, 768]	2,363,904	True
└MLPBlock (mlp_block)	[1, 197, 768]	[1, 197, 768]	4,723,968	True
└TransformerEncoderBlock (10)	[1, 197, 768]	[1, 197, 768]	--	True
└MultiHeadSelfAttentionBlock (msa_block)	[1, 197, 768]	[1, 197, 768]	2,363,904	True
└MLPBlock (mlp_block)	[1, 197, 768]	[1, 197, 768]	4,723,968	True
└TransformerEncoderBlock (11)	[1, 197, 768]	[1, 197, 768]	--	True
└MultiHeadSelfAttentionBlock (msa_block)	[1, 197, 768]	[1, 197, 768]	2,363,904	True
└MLPBlock (mlp_block)	[1, 197, 768]	[1, 197, 768]	4,723,968	True
Total params: 85,797,120				
Trainable params: 85,797,120				
Non-trainable params: 0				
Total mult-adds (Units.MEGABYTES): 172.46				
Input size (MB): 0.60				
Forward/backward pass size (MB): 102.88				
Params size (MB): 229.19				
Estimated Total Size (MB): 332.66				

Figura 6.12: Resumen de la arquitectura completa del modelo ViT from scratch.

6.1. ARQUITECTURA DESDE CERO (FROM SCRATCH)

6.1.2. Entrenamiento del modelo

En esta fase, se ha añadido un decoder al modelo de Vision Transformer previamente descrito, el cual tiene la función de generar imágenes a color de 224x224 píxeles. El objetivo de este decoder es producir una nueva imagen que coincida con las dimensiones de entrada del modelo, que en este caso son imágenes ruidosas. La salida del modelo será una imagen generada que se asemeje lo más posible a la imagen original sin ruido.

Layer (type [var_name])	Input Shape	Output Shape	Param #	Trainable
ViT (vit)	[1, 3, 224, 224]	[1, 3, 224, 224]	152,064	True
PatchEmbedding (patch_embedding)	[1, 3, 224, 224]	[1, 196, 768]	--	True
Conv2d (patcher)	[1, 3, 224, 224]	[1, 196, 768]	590,592	True
Flatten (flatten)	[1, 768, 14, 14]	[1, 768, 14, 14]	--	--
Dropout (embedding_dropout)	[1, 197, 768]	[1, 197, 768]	--	--
Sequential (transformer_encoder)	[1, 197, 768]	[1, 197, 768]	--	True
TransformerEncoderBlock (0)	[1, 197, 768]	[1, 197, 768]	--	True
MultiHeadSelfAttentionBlock (msa_block)	[1, 197, 768]	[1, 197, 768]	2,363,904	True
MLPBlock (mlp_block)	[1, 197, 768]	[1, 197, 768]	4,723,968	True
TransformerEncoderBlock (1)	[1, 197, 768]	[1, 197, 768]	--	True
MultiHeadSelfAttentionBlock (msa_block)	[1, 197, 768]	[1, 197, 768]	2,363,904	True
MLPBlock (mlp_block)	[1, 197, 768]	[1, 197, 768]	4,723,968	True
TransformerEncoderBlock (2)	[1, 197, 768]	[1, 197, 768]	--	True
MultiHeadSelfAttentionBlock (msa_block)	[1, 197, 768]	[1, 197, 768]	2,363,904	True
MLPBlock (mlp_block)	[1, 197, 768]	[1, 197, 768]	4,723,968	True
TransformerEncoderBlock (3)	[1, 197, 768]	[1, 197, 768]	--	True
MultiHeadSelfAttentionBlock (msa_block)	[1, 197, 768]	[1, 197, 768]	2,363,904	True
MLPBlock (mlp_block)	[1, 197, 768]	[1, 197, 768]	4,723,968	True
TransformerEncoderBlock (4)	[1, 197, 768]	[1, 197, 768]	--	True
MultiHeadSelfAttentionBlock (msa_block)	[1, 197, 768]	[1, 197, 768]	2,363,904	True
MLPBlock (mlp_block)	[1, 197, 768]	[1, 197, 768]	4,723,968	True
TransformerEncoderBlock (5)	[1, 197, 768]	[1, 197, 768]	--	True
MultiHeadSelfAttentionBlock (msa_block)	[1, 197, 768]	[1, 197, 768]	2,363,904	True
MLPBlock (mlp_block)	[1, 197, 768]	[1, 197, 768]	4,723,968	True
TransformerEncoderBlock (6)	[1, 197, 768]	[1, 197, 768]	--	True
MultiHeadSelfAttentionBlock (msa_block)	[1, 197, 768]	[1, 197, 768]	2,363,904	True
MLPBlock (mlp_block)	[1, 197, 768]	[1, 197, 768]	4,723,968	True
TransformerEncoderBlock (7)	[1, 197, 768]	[1, 197, 768]	--	True
MultiHeadSelfAttentionBlock (msa_block)	[1, 197, 768]	[1, 197, 768]	2,363,904	True
MLPBlock (mlp_block)	[1, 197, 768]	[1, 197, 768]	4,723,968	True
TransformerEncoderBlock (8)	[1, 197, 768]	[1, 197, 768]	--	True
MultiHeadSelfAttentionBlock (msa_block)	[1, 197, 768]	[1, 197, 768]	2,363,904	True
MLPBlock (mlp_block)	[1, 197, 768]	[1, 197, 768]	4,723,968	True
TransformerEncoderBlock (9)	[1, 197, 768]	[1, 197, 768]	--	True
MultiHeadSelfAttentionBlock (msa_block)	[1, 197, 768]	[1, 197, 768]	2,363,904	True
MLPBlock (mlp_block)	[1, 197, 768]	[1, 197, 768]	4,723,968	True
TransformerEncoderBlock (10)	[1, 197, 768]	[1, 197, 768]	--	True
MultiHeadSelfAttentionBlock (msa_block)	[1, 197, 768]	[1, 197, 768]	2,363,904	True
MLPBlock (mlp_block)	[1, 197, 768]	[1, 197, 768]	4,723,968	True
TransformerEncoderBlock (11)	[1, 197, 768]	[1, 197, 768]	--	True
MultiHeadSelfAttentionBlock (msa_block)	[1, 197, 768]	[1, 197, 768]	2,363,904	True
MLPBlock (mlp_block)	[1, 197, 768]	[1, 197, 768]	4,723,968	True
Sequential (decoder)	[1, 768]	[1, 3, 224, 224]	--	True
LayerNorm (0)	[1, 768]	[1, 768]	1,536	True
Linear (1)	[1, 768]	[1, 150528]	115,756,032	True
Unflatten (2)	[1, 150528]	[1, 3, 224, 224]	--	--

Total params: 201,554,688
Trainable params: 201,554,688
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 288.22
====
Input size (MB): 0.60
Forward/backward pass size (MB): 104.09
Params size (MB): 692.22
Estimated Total Size (MB): 796.90

Figura 6.13: Arquitectura completa del modelo ViT from scratch con decoder.

Este enfoque permite que el modelo no solo clasifique, sino que también realice una tarea de generación de imágenes, donde la calidad de la salida se mide en función de su similitud con la imagen de referencia (ground truth).

CAPÍTULO 6. RESULTADOS

Cuanto más parecida sea la imagen generada a la imagen etiquetada sin ruido, mejor será el rendimiento del modelo.

A continuación, voy a hacer una descripción del código utilizado para la implementación del decoder al modelo de vision transformer generado from scratch de la sección anterior. Se define la clase ViT para implementar la arquitectura Vision Transformer, permitiendo el entrenamiento del modelo para generar imágenes a partir de entradas con ruido. El constructor de la clase inicializa varios parámetros esenciales que guiarán el aprendizaje del modelo.

Primero, se establece que el tamaño de la imagen debe ser divisible por el tamaño de los parches. Esto es crucial, ya que la imagen se dividirá en parches para su procesamiento. Se calcula el número total de parches que se generarán a partir de la imagen, y se crea un embedding de clase que será aprendible, así como un embedding de posición. Estos embeddings son fundamentales, ya que añaden información contextual a cada parche y ayudan al modelo a entender la relación entre ellos.

La capa de embedding de parches se configura a través de la clase *PathEmbedding*, que transforma las imágenes de entrada en parches, mapeando las dimensiones de entrada en una representación de embedding del tamaño especificado. A continuación, se crea una secuencia de bloques de codificador Transformer a través de la clase *TransformerEncoderBlock*, que consta de varias capas que aplican atención y MLP (perceptrón multicapa), formando el núcleo del modelo.

Ahora es cuando se incluye el decoder para la generación de imágenes sin ruido. Está compuesto por una capa de normalización y una capa lineal. Esta última mapea el embedding resultante a un tamaño que corresponde a la imagen generada. Finalmente, se utiliza *Unflatten* para reestructurar la salida en forma de imagen (C, H, W), donde C es el número de canales, y H y W son las dimensiones de la imagen.

En el método forward, el modelo recibe como entrada un tensor de imágenes. Primero, se determina el tamaño del batch y se crea un token de clase que se expande para igualar el tamaño del batch. Luego, se generan los embeddings de los parches y se concatenan con el token de clase. A continuación, se añaden los embeddings de posición a la concatenación y se aplica un dropout para regularizar el entrenamiento.

Los embeddings resultantes se pasan a través del codificador Transformer, donde se realizan las operaciones de atención y se genera una representación más abstracta de las imágenes. Finalmente, se procesa a través del decoder para producir la imagen final.

Para validar la implementación, se crea un tensor de imagen aleatorio con las mismas dimensiones que una imagen de entrada y se pasa a una instancia

6.1. ARQUITECTURA DESDE CERO (FROM SCRATCH)

del modelo ViT. Este proceso permite comprobar que la arquitectura del modelo se configura correctamente y que puede manejar las entradas de la forma esperada. En la Figura 6.13 se puede ver la arquitectura completa del modelo, una vez se le ha implementado el decoder para la generación de imágenes.

La principal diferencia entre el modelo Vision Transformer (ViT) sin decoder y el modelo con decoder radica en la estructura de salida y el propósito del modelo. El modelo sin decoder produce una representación de características de la imagen de entrada, transformando la imagen de un tamaño de entrada de 224×224 a una salida de 197 tokens de características con 768 dimensiones cada uno. En este caso, el enfoque se centra en extraer información significativa de las imágenes, lo que implica un número total de parámetros de aproximadamente 85,8 millones.

Por otro lado, el modelo con decoder está diseñado para generar imágenes de salida de 224×224 , lo que implica una arquitectura más compleja que incluye un componente adicional que permite la reconstrucción de imágenes. Este modelo también utiliza el mismo proceso de codificación inicial, pero luego transforma la representación en una salida que conserva la estructura espacial de la imagen original. Esto lo hace más adecuado para tareas de generación de imágenes o de denoising. En comparación con el modelo sin decoder, el modelo con decoder presenta una mayor carga computacional debido a su mayor complejidad en el manejo de los datos de salida.

El entrenamiento del modelo Vision Transformer (ViT) se llevó a cabo utilizando una configuración que incluyó la definición del modelo, la función de pérdidas, el optimizador y el número de épocas. Se estableció el modelo utilizando la clase ‘ViT’, y se eligió la función de pérdida de error cuadrático medio (MSELoss) para evaluar la calidad de las predicciones en comparación con las imágenes originales. El optimizador seleccionado fue Adam, con una tasa de aprendizaje de 1×10^{-4} . Se fijó el número de épocas en 5, lo que permitió evaluar el rendimiento del modelo en un tiempo relativamente corto.

```
RuntimeError: CUDA out of memory. Tried to allocate 74.00 MiB. GPU 0 has a total capacity of 7.78 GiB of which 51.31 MiB is free. Including non-PyTorch memory, this process has 7.26 GiB memory in use.
```

Figura 6.14: Error durante el entrenamiento del modelo por falta de capacidad GPU.

Durante el entrenamiento, el modelo se trasladó a la GPU para aprovechar su capacidad de procesamiento. El bucle de entrenamiento se diseñó para iterar sobre las épocas y los lotes de datos del dataloader, realizando un paso hacia adelante para obtener las predicciones y calcular la pérdida. A continuación, se llevó a cabo el paso hacia atrás para actualizar los pesos del

CAPÍTULO 6. RESULTADOS

modelo. En cada época, se registró la pérdida promedio para monitorizar el progreso. Sin embargo, a pesar de esta configuración, no se pudo completar el entrenamiento del modelo debido a una falta de capacidad de GPU, lo que impidió su entrenamiento adecuado.

6.2. Arquitectura con modelos preentrenados (Transfer Learning)

La técnica de Transfer Learning se ha empleado en este proyecto para aprovechar las capacidades de modelos preentrenados y mejorar la eficiencia y efectividad del proceso de entrenamiento. En lugar de entrenar un modelo desde cero, se ha optado por utilizar un modelo de Vision Transformer que ya ha sido previamente entrenado en una gran cantidad de datos. Este enfoque permite transferir los conocimientos adquiridos durante el entrenamiento inicial a la tarea específica de denoising de imágenes. Es una técnica muy común en los entrenamientos de modelos de inteligencia artificial.

6.2.1. Aplicación de Transfer Learning

En la implementación del Transfer Learning, se ha creado una clase *ViT-TransferLearning* que extiende *nn.Module*, permitiendo modificar la arquitectura de un modelo de Vision Transformer (ViT) preentrenado. En el constructor de la clase, se carga el modelo ViT de tipo vit-b-16 ya entrenado en un amplio conjunto de datos. Este modelo cuenta con capas que han aprendido representaciones generales de imágenes, las cuales se pueden aprovechar para tareas específicas, como el denoising de imágenes.

Una de las modificaciones clave en este proceso es la eliminación de la capa de clasificación preentrenada, que en este caso es la cabeza del modelo *self.encoder.heads*. Esta capa se reemplaza por una identidad *nn.Identity()*, permitiendo que el modelo conserve sus capacidades de representación sin realizar la clasificación. En su lugar, se añade un nuevo componente llamado decoder, que se encarga de generar las imágenes a partir de los embeddings extraídos del modelo preentrenado. El decoder consta de una capa de normalización y una capa lineal que transforma los embeddings en un formato compatible con el tamaño de las imágenes de salida, seguido de una operación de unflatten para dar forma a los datos en el formato adecuado (canales, altura, y ancho). Este enfoque no solo reduce el tiempo de entrenamiento al aprovechar las características previamente aprendidas, sino que también mejora el rendimiento en la tarea específica de reconstrucción de imágenes limpias a

6.3. OPTIMIZACIÓN DE MODELOS PARA DISPOSITIVOS MÓVILES

partir de imágenes ruidosas, ajustando el modelo a las particularidades del conjunto de datos utilizado en el proyecto.

Layer (type (var_name))	Input Shape	Output Shape	Param #	Trainable
ViTTransferLearning (ViTTransferLearning)	[1, 3, 224, 224]	[1, 3, 224, 224]	--	True
—VisionTransformer (encoder)	[1, 3, 224, 224]	[1, 768]	768	True
—Conv2d (conv_proj)	[1, 3, 224, 224]	[1, 768, 14, 14]	590,592	True
—Encoder (encoder)	[1, 197, 768]	[1, 197, 768]	151,296	True
—Dropout (dropout)	[1, 197, 768]	[1, 197, 768]	--	--
—Sequential (layers)	[1, 197, 768]	[1, 197, 768]	85,054,464	True
—LayerNorm (ln)	[1, 197, 768]	[1, 197, 768]	1,536	True
—Identity (heads)	[1, 768]	[1, 768]	--	--
—Sequential (decoder)	[1, 768]	[1, 3, 224, 224]	--	True
—LayerNorm (8)	[1, 768]	[1, 768]	1,536	True
—Linear (1)	[1, 768]	[1, 158528]	115,756,032	True
—Unflatten (2)	[1, 158528]	[1, 3, 224, 224]	--	--
Total params: 201,556,224				
Trainable params: 201,556,224				
Non-trainable params: 0				
Total mult-adds (Units.MEGABYTES): 288.22				
Input size (MB): 0.60				
Forward/backward pass size (MB): 105.38				
Params size (MB): 692.22				
Estimated Total Size (MB): 798.12				

Figura 6.15: Arquitectura completa del modelo ViT transfer learning con decoder.

El modelo ViT encoder-decoder from scratch se entrena desde cero con parámetros inicializados aleatoriamente, lo que requiere un conjunto de datos amplio y diverso para un rendimiento óptimo, totalizando aproximadamente 201.55 millones de parámetros. En cambio, el modelo con transfer learning utiliza una arquitectura preentrenada, permitiendo un entrenamiento más rápido y efectivo, especialmente con datos limitados. Aunque ambos modelos tienen un tamaño de parámetros similar, el transfer learning se beneficia de representaciones más ricas, lo que puede mejorar el rendimiento en tareas específicas.

Lamentablemente, no se ha podido entrenar el modelo de transfer learning debido a limitaciones de memoria en la GPU, similar a lo ocurrido con el modelo ViT encoder-decoder from scratch. A pesar de los esfuerzos realizados para optimizar la arquitectura y reducir el tamaño de los lotes de entrenamiento, la capacidad de memoria disponible no fue suficiente para manejar el tamaño del modelo y los datos, lo que impidió completar el proceso de entrenamiento de manera efectiva.

6.3. Optimización de modelos para dispositivos móviles

En la actualidad, los modelos transformers han demostrado un rendimiento sobresaliente en diversas tareas de visión por computadora. Sin embargo,

CAPÍTULO 6. RESULTADOS

su implementación en dispositivos móviles presenta desafíos significativos debido a las limitaciones inherentes en términos de recursos de computación, memoria y duración de la batería. Estos dispositivos requieren soluciones que no solo mantengan la precisión del modelo, sino que también garanticen una ejecución eficiente y rápida.

La optimización de modelos implica una serie de estrategias y técnicas diseñadas para reducir la complejidad de los modelos sin comprometer su rendimiento. Entre las técnicas más comunes se encuentran el pruning, la cuantización y los modelos binarios, cada una de las cuales aborda diferentes aspectos del modelo, como el tamaño, la velocidad y el consumo de energía. Estas técnicas no solo permiten la reducción del tamaño del modelo, sino que también mejoran la velocidad de inferencia, lo que es esencial para aplicaciones en tiempo real que requieren decisiones instantáneas.

Además, la transición de un modelo optimizado a un entorno móvil no se limita a la optimización en sí misma. También es crucial realizar la conversión del modelo a formatos que sean compatibles con los dispositivos móviles, como TorchScript y ONNX Runtime.

En este capítulo, se explorarán en detalle las técnicas de optimización aplicadas a los modelos de Vision Transformers, su conversión a formatos móviles, así como los aspectos clave a considerar al integrar estos modelos en aplicaciones móviles. Se buscará proporcionar una visión comprensiva de cómo estas optimizaciones no solo mejoran la eficiencia del modelo, sino que también permiten su uso práctico en el mundo real, ampliando así el alcance y la utilidad de la inteligencia artificial en dispositivos de uso cotidiano.

6.3.1. Aplicación de las técnicas de optimización

Para optimizar el modelo de Vision Transformer para su implementación en dispositivos móviles, se han aplicado tres técnicas principales: pruning, cuantización y modelos binarios. A continuación, se describen cada una de estas técnicas y se proporciona una explicación del código utilizado para implementarlas.

Pruning

El pruning es una técnica que consiste en eliminar pesos de la red neuronal que no contribuyen significativamente al rendimiento del modelo. Este proceso se basa en la premisa de que muchos de los parámetros de un modelo entrenado son redundantes o irrelevantes, y su eliminación puede llevar a un modelo más ligero y rápido. En este caso, se ha utilizado pruning de tipo

6.3. OPTIMIZACIÓN DE MODELOS PARA DISPOSITIVOS MÓVILES

l1 unstructured, que elimina un porcentaje específico de pesos de las capas lineales del modelo.

Layer (type [var_name])	Input Shape	Output Shape	Param #	Trainable
ViT (ViT)	[1, 3, 224, 224]	[1, 197, 768]	152,064	True
PatchEmbedding (patch_embedding)	[1, 3, 224, 224]	[1, 196, 768]	--	True
Conv2d (patcher)	[1, 3, 224, 224]	[1, 768, 14, 14]	590,592	True
Flatten (flatten)	[1, 768, 14, 14]	[1, 768, 196]	--	--
Dropout (embedding_dropout)	[1, 197, 768]	[1, 197, 768]	--	--
Sequential (transformer_encoder)	[1, 197, 768]	[1, 197, 768]	--	True
TransformerEncoderBlock (0)	[1, 197, 768]	[1, 197, 768]	--	True
MultiHeadSelfAttentionBlock (msa_block)	[1, 197, 768]	[1, 197, 768]	2,363,904	True
MLPBlock (mlp_block)	[1, 197, 768]	[1, 197, 768]	1,892,812	True
TransformerEncoderBlock (1)	[1, 197, 768]	[1, 197, 768]	--	True
MultiHeadSelfAttentionBlock (msa_block)	[1, 197, 768]	[1, 197, 768]	2,363,904	True
MLPBlock (mlp_block)	[1, 197, 768]	[1, 197, 768]	1,892,812	True
TransformerEncoderBlock (2)	[1, 197, 768]	[1, 197, 768]	--	True
MultiHeadSelfAttentionBlock (msa_block)	[1, 197, 768]	[1, 197, 768]	2,363,904	True
MLPBlock (mlp_block)	[1, 197, 768]	[1, 197, 768]	1,892,812	True
TransformerEncoderBlock (3)	[1, 197, 768]	[1, 197, 768]	--	True
MultiHeadSelfAttentionBlock (msa_block)	[1, 197, 768]	[1, 197, 768]	2,363,904	True
MLPBlock (mlp_block)	[1, 197, 768]	[1, 197, 768]	1,892,812	True
TransformerEncoderBlock (4)	[1, 197, 768]	[1, 197, 768]	--	True
MultiHeadSelfAttentionBlock (msa_block)	[1, 197, 768]	[1, 197, 768]	2,363,904	True
MLPBlock (mlp_block)	[1, 197, 768]	[1, 197, 768]	1,892,812	True
TransformerEncoderBlock (5)	[1, 197, 768]	[1, 197, 768]	--	True
MultiHeadSelfAttentionBlock (msa_block)	[1, 197, 768]	[1, 197, 768]	2,363,904	True
MLPBlock (mlp_block)	[1, 197, 768]	[1, 197, 768]	1,892,812	True
TransformerEncoderBlock (6)	[1, 197, 768]	[1, 197, 768]	--	True
MultiHeadSelfAttentionBlock (msa_block)	[1, 197, 768]	[1, 197, 768]	2,363,904	True
MLPBlock (mlp_block)	[1, 197, 768]	[1, 197, 768]	1,892,812	True
TransformerEncoderBlock (7)	[1, 197, 768]	[1, 197, 768]	--	True
MultiHeadSelfAttentionBlock (msa_block)	[1, 197, 768]	[1, 197, 768]	2,363,904	True
MLPBlock (mlp_block)	[1, 197, 768]	[1, 197, 768]	1,892,812	True
TransformerEncoderBlock (8)	[1, 197, 768]	[1, 197, 768]	--	True
MultiHeadSelfAttentionBlock (msa_block)	[1, 197, 768]	[1, 197, 768]	2,363,904	True
MLPBlock (mlp_block)	[1, 197, 768]	[1, 197, 768]	1,892,812	True
TransformerEncoderBlock (9)	[1, 197, 768]	[1, 197, 768]	--	True
MultiHeadSelfAttentionBlock (msa_block)	[1, 197, 768]	[1, 197, 768]	2,363,904	True
MLPBlock (mlp_block)	[1, 197, 768]	[1, 197, 768]	1,892,812	True
TransformerEncoderBlock (10)	[1, 197, 768]	[1, 197, 768]	--	True
MultiHeadSelfAttentionBlock (msa_block)	[1, 197, 768]	[1, 197, 768]	2,363,904	True
MLPBlock (mlp_block)	[1, 197, 768]	[1, 197, 768]	1,892,812	True
TransformerEncoderBlock (11)	[1, 197, 768]	[1, 197, 768]	--	True
MultiHeadSelfAttentionBlock (msa_block)	[1, 197, 768]	[1, 197, 768]	2,363,904	True
MLPBlock (mlp_block)	[1, 197, 768]	[1, 197, 768]	1,892,812	True

Total params: 51,823,248
 Trainable params: 51,823,248
 Non-trainable params: 0
 Total mult-adds (Units.MEGABYTES): 138.49

Input size (MB): 0.68
 Forward/backward pass size (MB): 102.88
 Params size (MB): 93.29
 Estimated Total Size (MB): 196.77

Figura 6.16: Arquitectura del modelo ViT con pruning aplicado.

El código que se ha utilizado para la implementación, comienza importando la funcionalidad necesaria para aplicar pruning a las capas del modelo. Se define la función *apply-pruning*, que recorre todos los módulos del modelo y verifica si son instancias de *nn.Sequential*. Dentro de estos módulos, se identifica cada submódulo que sea una capa lineal (*nn.Linear*). Para estas capas, se aplica el pruning utilizando *prune.l1-unstructured*, que elimina un porcentaje específico de pesos según el valor del argumento *amount* que se le pase a la función. Después de aplicar el pruning, se devuelve el modelo modificado.

Luego, se utiliza esta función para podar un modelo ViT, eliminando el 60 % de los pesos de sus capas lineales. Finalmente, se visualiza la arquitectura del modelo.

CAPÍTULO 6. RESULTADOS

tura del modelo podado usando la función `summary`, que muestra detalles sobre el tamaño de entrada, el tamaño de salida, el número de parámetros y si son entrenables. Esta visualización ayuda a entender cómo ha cambiado el modelo tras aplicar el pruning.

El modelo original de Vision Transformer (ViT) cuenta con un total de 85,797,120 parámetros, lo que lo convierte en un modelo considerablemente complejo y con un alto costo computacional. Esta complejidad se deriva principalmente de la arquitectura de sus bloques de codificación, especialmente en los componentes de *MultiHeadSelfAttentionBlock* y *MLPBlock*, que poseen un gran número de parámetros entrenables. Por ejemplo, cada *MLPBlock* en el modelo original tiene alrededor de 4,723,968 parámetros.

Por otro lado, al aplicar la técnica de pruning, el número total de parámetros del modelo se reduce significativamente, lo que se traduce en un modelo más ligero y eficiente. Esto se logra eliminando de manera selectiva aquellos parámetros que tienen un impacto menor en el rendimiento del modelo, lo que permite mantener la precisión mientras se disminuye el tamaño del modelo. En este caso, el modelo pruned presenta una reducción en el número de parámetros de los *MLPBlock* a aproximadamente 1,892,812, lo que representa una disminución sustancial.

La reducción en el número de parámetros no solo facilita un menor uso de memoria, sino que también permite un procesamiento más rápido y eficiente, lo que es especialmente beneficioso para su implementación en dispositivos móviles. Un modelo más ligero es mejor para aplicaciones en tiempo real donde los recursos son limitados y la velocidad de inferencia es prioritaria. En conclusión, la aplicación de pruning en el Vision Transformer no solo optimiza el rendimiento del modelo, sino que también permite que sea más accesible para una variedad de plataformas y dispositivos.

Cuantización

La cuantización es el proceso de reducir la precisión de los números utilizados para representar los pesos y activaciones del modelo. Esta técnica puede resultar en una reducción significativa del tamaño del modelo y un aumento en la velocidad de inferencia, lo cual es especialmente beneficioso en dispositivos móviles. En este caso, se ha realizado una cuantización post-entrenamiento, donde el modelo se calibra utilizando datos de entrada antes de convertirlo a un formato cuantizado.

El código que se ha utilizado para ello, comienza importando las bibliotecas necesarias para trabajar con PyTorch y modelos. Se asume que se tiene un modelo de tipo ViT, que se instancia en la variable `vit-model`. Primero, se mueve el modelo a la CPU, ya que la cuantización suele realizarse en este dis-

6.3. OPTIMIZACIÓN DE MODELOS PARA DISPOSITIVOS MÓVILES

positivo. Se prepara el modelo para la cuantización estableciendo el modo de evaluación con `vit-model.eval()` y llamando a `torch.quantization.prepare(vit-model, inplace=True)`. Esto prepara el modelo para la cuantización, creando los registros necesarios para almacenar la información de los parámetros antes de la conversión.

Layer (type:depth-idx)	Output Shape	Param #
ViT	[1, 197, 768]	152,064
PatchEmbedding: 1-1	[1, 196, 768]	--
└Conv2d: 2-1	[1, 768, 14, 14]	590,592
└Flatten: 2-2	[1, 768, 196]	--
Dropout: 1-2	[1, 197, 768]	--
Sequential: 1-3	[1, 197, 768]	--
└TransformerEncoderBlock: 2-3	[1, 197, 768]	--
└MultiHeadSelfAttentionBlock: 3-1	[1, 197, 768]	2,363,904
└MLPBlock: 3-2	[1, 197, 768]	4,723,968
└TransformerEncoderBlock: 2-4	[1, 197, 768]	--
└MultiHeadSelfAttentionBlock: 3-3	[1, 197, 768]	2,363,904
└MLPBlock: 3-4	[1, 197, 768]	4,723,968
└TransformerEncoderBlock: 2-5	[1, 197, 768]	--
└MultiHeadSelfAttentionBlock: 3-5	[1, 197, 768]	2,363,904
└MLPBlock: 3-6	[1, 197, 768]	4,723,968
└TransformerEncoderBlock: 2-6	[1, 197, 768]	--
└MultiHeadSelfAttentionBlock: 3-7	[1, 197, 768]	2,363,904
└MLPBlock: 3-8	[1, 197, 768]	4,723,968
└TransformerEncoderBlock: 2-7	[1, 197, 768]	--
└MultiHeadSelfAttentionBlock: 3-9	[1, 197, 768]	2,363,904
└MLPBlock: 3-10	[1, 197, 768]	4,723,968
└TransformerEncoderBlock: 2-8	[1, 197, 768]	--
└MultiHeadSelfAttentionBlock: 3-11	[1, 197, 768]	2,363,904
└MLPBlock: 3-12	[1, 197, 768]	4,723,968
└TransformerEncoderBlock: 2-9	[1, 197, 768]	--
└MultiHeadSelfAttentionBlock: 3-13	[1, 197, 768]	2,363,904
└MLPBlock: 3-14	[1, 197, 768]	4,723,968
└TransformerEncoderBlock: 2-10	[1, 197, 768]	--
└MultiHeadSelfAttentionBlock: 3-15	[1, 197, 768]	2,363,904
└MLPBlock: 3-16	[1, 197, 768]	4,723,968
└TransformerEncoderBlock: 2-11	[1, 197, 768]	--
└MultiHeadSelfAttentionBlock: 3-17	[1, 197, 768]	2,363,904
└MLPBlock: 3-18	[1, 197, 768]	4,723,968
└TransformerEncoderBlock: 2-12	[1, 197, 768]	--
└MultiHeadSelfAttentionBlock: 3-19	[1, 197, 768]	2,363,904
└MLPBlock: 3-20	[1, 197, 768]	4,723,968
└TransformerEncoderBlock: 2-13	[1, 197, 768]	--
└MultiHeadSelfAttentionBlock: 3-21	[1, 197, 768]	2,363,904
└MLPBlock: 3-22	[1, 197, 768]	4,723,968
└TransformerEncoderBlock: 2-14	[1, 197, 768]	--
└MultiHeadSelfAttentionBlock: 3-23	[1, 197, 768]	2,363,904
└MLPBlock: 3-24	[1, 197, 768]	4,723,968
Total params: 85,797,120		
Trainable params: 85,797,120		
Non-trainable params: 0		
Total mult-adds (Units.MEGABYTES): 172.46		
====		
Input size (MB): 0.60		
Forward/backward pass size (MB): 102.88		
Params size (MB): 229.19		
Estimated Total Size (MB): 332.66		
====		

Figura 6.17: Arquitectura del modelo ViT con cuantización aplicada.

Después, se realiza una inferencia con datos de entrada generados aleatoriamente para calibrar el modelo. Esto es crucial, ya que permite ajustar los rangos de los parámetros del modelo según los valores que encuentra en las activaciones. Se utiliza `torch.no_grad()` para evitar el cálculo de gradientes, lo que optimiza el uso de memoria y mejora el rendimiento durante la

CAPÍTULO 6. RESULTADOS

inferencia.

Finalmente, se convierte el modelo a su versión cuantizada mediante `torch.quantization.convert(vit-model, inplace=True)`. Esto reduce el tamaño del modelo al representar los pesos con menor precisión, lo que puede resultar en un modelo más ligero y rápido. Se imprime el modelo cuantizado para verificar que la conversión se haya realizado correctamente.

El modelo original utiliza números de punto flotante de 32 bits para representar sus pesos y activaciones, lo que asegura alta precisión pero también implica un mayor consumo de memoria y recursos de procesamiento. En cambio, el modelo cuantizado representa estos parámetros en formatos más compactos, como enteros de 8 bits, reduciendo significativamente el uso de memoria y mejorando la eficiencia en términos de velocidad y consumo energético.

A pesar de que ambos modelos mantienen la misma arquitectura y número de capas, las capas del modelo cuantizado pueden implementar operaciones optimizadas, como `QuantizedLinear` o `QuantizedConv2d`, diseñadas específicamente para manejar la representación cuantizada. Esto permite realizar cálculos más rápidos, haciendo que el modelo cuantizado sea más adecuado para entornos con recursos limitados. Aunque el conteo total de parámetros y la estructura general no muestran cambios evidentes, la cuantización transforma la forma en que se almacenan y procesan los parámetros, resultando en una mejora en la eficiencia operativa del modelo.

Modelos binarios

Los modelos binarios son una forma extrema de cuantización donde tanto los pesos como las activaciones se reducen a valores binarios, es decir, -1 y 1. Esta técnica permite una notable reducción en el tamaño del modelo y mejora la velocidad de inferencia, ya que las operaciones se simplifican a operaciones lógicas. La implementación de modelos binarios requiere la creación de capas personalizadas que binarizan los pesos y las activaciones durante el paso hacia adelante.

El código que se ha utilizado, define una serie de clases y funciones para crear un modelo de Vision Transformer (ViT) utilizando capas binarias. La función `binarize(tensor)` toma un tensor y lo “binariza”, limitando sus valores entre -1 y 1 y aplicando la función signo. Esto convierte los pesos y activaciones a valores de -1 o 1.

- **BinaryLinear:** Esta clase representa una versión binaria de una capa lineal. En su método `forward`, antes de realizar la multiplicación matricial, se binarizan los pesos de la capa lineal utilizando la función

6.3. OPTIMIZACIÓN DE MODELOS PARA DISPOSITIVOS MÓVILES

binarize. Esto permite que la capa utilice solo pesos binarios durante la inferencia.

- **BinaryMultiHeadSelfAttentionBlock**: Esta clase implementa un bloque de atención multi-cabeza en su versión binaria. En su método *forward*, se binarizan las activaciones de entrada antes de pasarlas a la capa de atención. Esto significa que tanto las activaciones de entrada como los pesos de la capa de atención se procesan de forma binaria.
- **BinaryMLPBlock**: Esta clase representa un bloque de perceptrón multicapa (MLP) en su versión binaria. Utiliza instancias de *BinaryLinear* para las capas lineales. En el método *forward*, después de pasar los datos a través de la primera capa, se binarizan las activaciones mediante la función *binarize*. La salida se pasa a través de la segunda capa lineal.
- **BinaryTransformerEncoderBlock**: Esta clase combina el bloque de atención multi-cabeza y el bloque MLP en una única unidad. En su método *forward*, primero aplica la atención, seguido de la normalización de la capa y luego la operación del MLP. Finalmente, normaliza la salida nuevamente.
- **BinaryViT**: Esta es la clase principal que crea el modelo de Vision Transformer utilizando las capas binarias definidas anteriormente. En su constructor, se inicializan los parámetros necesarios, incluidas las incrustaciones de clase y de posición, así como la capa de incrustación de parches. La parte central del modelo es una secuencia de bloques *BinaryTransformerEncoderBlock*. En el método *forward*, se gestionan las incrustaciones y la entrada de parches, se concatenan el token de clase y las incrustaciones de posición, y finalmente se pasa la salida a través del encoder.

Al final del código, se inicializa una instancia de *BinaryViT* y se utiliza la función *summary* de *torchinfo* para visualizar la arquitectura del modelo. Esto proporciona un resumen del modelo, mostrando el tamaño de entrada, el tamaño de salida, el número de parámetros y el tamaño de los núcleos (kernel size), lo que permite entender la complejidad del modelo y su estructura.

El modelo binario reduce tanto los pesos como las activaciones a valores binarios (+1 o -1), lo que disminuye significativamente el uso de memoria y las operaciones aritméticas. Esto hace que sea mucho más eficiente en términos de ejecución en hardware de bajo rendimiento, como dispositivos móviles. Sin embargo, esta simplicidad también implica una pérdida considerable de

CAPÍTULO 6. RESULTADOS

información, lo que puede afectar la precisión del modelo en tareas complejas. Para compensar esta degradación, se utilizan técnicas como el escalado en las capas binarizadas, pero aun así, es difícil igualar el rendimiento del modelo original en escenarios donde la precisión es crítica.

Layer (type:depth-idx)	Input Shape	Output Shape	Param #
BinaryViT	[1, 3, 224, 224]	[1, 197, 768]	152,064
PatchEmbedding: 1-1	[1, 3, 224, 224]	[1, 196, 768]	--
Conv2d: 2-1	[1, 3, 224, 224]	[1, 768, 14, 14]	596,592
Flatten: 2-2	[1, 768, 14, 14]	[1, 768, 196]	--
Dropout: 1-2	[1, 197, 768]	[1, 197, 768]	--
Sequential: 1-3	[1, 197, 768]	[1, 197, 768]	--
BinaryTransformerEncoderBlock: 2-3	[1, 197, 768]	[1, 197, 768]	--
BinaryMultiHeadSelfAttentionBlock: 3-1	[1, 197, 768]	[1, 197, 768]	2,362,368
LayerNorm: 3-2	[1, 197, 768]	[1, 197, 768]	1,536
BinaryMLPBlock: 3-3	[1, 197, 768]	[1, 197, 768]	4,722,432
LayerNorm: 3-4	[1, 197, 768]	[1, 197, 768]	1,536
BinaryTransformerEncoderBlock: 2-4	[1, 197, 768]	[1, 197, 768]	--
BinaryMultiHeadSelfAttentionBlock: 3-5	[1, 197, 768]	[1, 197, 768]	2,362,368
LayerNorm: 3-6	[1, 197, 768]	[1, 197, 768]	1,536
BinaryMLPBlock: 3-7	[1, 197, 768]	[1, 197, 768]	4,722,432
LayerNorm: 3-8	[1, 197, 768]	[1, 197, 768]	1,536
BinaryTransformerEncoderBlock: 2-5	[1, 197, 768]	[1, 197, 768]	--
BinaryMultiHeadSelfAttentionBlock: 3-9	[1, 197, 768]	[1, 197, 768]	2,362,368
LayerNorm: 3-10	[1, 197, 768]	[1, 197, 768]	1,536
BinaryMLPBlock: 3-11	[1, 197, 768]	[1, 197, 768]	4,722,432
LayerNorm: 3-12	[1, 197, 768]	[1, 197, 768]	1,536
BinaryTransformerEncoderBlock: 2-6	[1, 197, 768]	[1, 197, 768]	--
BinaryMultiHeadSelfAttentionBlock: 3-13	[1, 197, 768]	[1, 197, 768]	2,362,368
LayerNorm: 3-14	[1, 197, 768]	[1, 197, 768]	1,536
BinaryMLPBlock: 3-15	[1, 197, 768]	[1, 197, 768]	4,722,432
LayerNorm: 3-16	[1, 197, 768]	[1, 197, 768]	1,536
BinaryTransformerEncoderBlock: 2-7	[1, 197, 768]	[1, 197, 768]	--
BinaryMultiHeadSelfAttentionBlock: 3-17	[1, 197, 768]	[1, 197, 768]	2,362,368
LayerNorm: 3-18	[1, 197, 768]	[1, 197, 768]	1,536
BinaryMLPBlock: 3-19	[1, 197, 768]	[1, 197, 768]	4,722,432
LayerNorm: 3-20	[1, 197, 768]	[1, 197, 768]	1,536
BinaryTransformerEncoderBlock: 2-8	[1, 197, 768]	[1, 197, 768]	--
BinaryMultiHeadSelfAttentionBlock: 3-21	[1, 197, 768]	[1, 197, 768]	2,362,368
LayerNorm: 3-22	[1, 197, 768]	[1, 197, 768]	1,536
BinaryMLPBlock: 3-23	[1, 197, 768]	[1, 197, 768]	4,722,432
LayerNorm: 3-24	[1, 197, 768]	[1, 197, 768]	1,536
BinaryTransformerEncoderBlock: 2-9	[1, 197, 768]	[1, 197, 768]	--
BinaryMultiHeadSelfAttentionBlock: 3-25	[1, 197, 768]	[1, 197, 768]	2,362,368
LayerNorm: 3-26	[1, 197, 768]	[1, 197, 768]	1,536
BinaryMLPBlock: 3-27	[1, 197, 768]	[1, 197, 768]	4,722,432
LayerNorm: 3-28	[1, 197, 768]	[1, 197, 768]	1,536
BinaryTransformerEncoderBlock: 2-10	[1, 197, 768]	[1, 197, 768]	--
BinaryMultiHeadSelfAttentionBlock: 3-29	[1, 197, 768]	[1, 197, 768]	2,362,368
LayerNorm: 3-30	[1, 197, 768]	[1, 197, 768]	1,536
BinaryMLPBlock: 3-31	[1, 197, 768]	[1, 197, 768]	4,722,432
LayerNorm: 3-32	[1, 197, 768]	[1, 197, 768]	1,536
BinaryTransformerEncoderBlock: 2-11	[1, 197, 768]	[1, 197, 768]	--
BinaryMultiHeadSelfAttentionBlock: 3-33	[1, 197, 768]	[1, 197, 768]	2,362,368
LayerNorm: 3-34	[1, 197, 768]	[1, 197, 768]	1,536
BinaryMLPBlock: 3-35	[1, 197, 768]	[1, 197, 768]	4,722,432
LayerNorm: 3-36	[1, 197, 768]	[1, 197, 768]	1,536
BinaryTransformerEncoderBlock: 2-12	[1, 197, 768]	[1, 197, 768]	--
BinaryMultiHeadSelfAttentionBlock: 3-37	[1, 197, 768]	[1, 197, 768]	2,362,368
LayerNorm: 3-38	[1, 197, 768]	[1, 197, 768]	1,536
BinaryMLPBlock: 3-39	[1, 197, 768]	[1, 197, 768]	4,722,432
LayerNorm: 3-40	[1, 197, 768]	[1, 197, 768]	1,536
BinaryTransformerEncoderBlock: 2-13	[1, 197, 768]	[1, 197, 768]	--
BinaryMultiHeadSelfAttentionBlock: 3-41	[1, 197, 768]	[1, 197, 768]	2,362,368
LayerNorm: 3-42	[1, 197, 768]	[1, 197, 768]	1,536
BinaryMLPBlock: 3-43	[1, 197, 768]	[1, 197, 768]	4,722,432
LayerNorm: 3-44	[1, 197, 768]	[1, 197, 768]	1,536
BinaryTransformerEncoderBlock: 2-14	[1, 197, 768]	[1, 197, 768]	--
BinaryMultiHeadSelfAttentionBlock: 3-45	[1, 197, 768]	[1, 197, 768]	2,362,368
LayerNorm: 3-46	[1, 197, 768]	[1, 197, 768]	1,536
BinaryMLPBlock: 3-47	[1, 197, 768]	[1, 197, 768]	4,722,432
LayerNorm: 3-48	[1, 197, 768]	[1, 197, 768]	1,536
Total params: 85,797,120			
Trainable params: 85,797,120			
Non-trainable params: 0			
Total mult-adds (Units.MEGABYTES): 115.79			
Input size (MB): 0.60			
Forward/backward pass size (MB): 30.25			
Params size (MB): 2.51			
Estimated Total Size (MB): 33.37			

Figura 6.18: Arquitectura del modelo binario ViT.

En contraste, el modelo original utiliza representaciones de 32 bits en

6.3. OPTIMIZACIÓN DE MODELOS PARA DISPOSITIVOS MÓVILES

punto flotante, lo que le permite manejar con mayor detalle las complejas relaciones de las características en los datos. Esto se traduce en un mayor uso de memoria y más operaciones computacionales, pero a cambio, logra una precisión mucho mayor que los modelos binarios. Si bien el modelo original puede no ser adecuado para dispositivos con limitaciones de recursos, sigue siendo la opción preferida cuando se prioriza el rendimiento sobre la eficiencia.

6.3.2. Introducción del modelo optimizado en un dispositivo móvil

La optimización del modelo Vision Transformer para su implementación en dispositivos móviles ha requerido la aplicación de varias técnicas, siendo la binarización, la cuantización y el pruning. Sin embargo, la conversión a un formato compatible con dispositivos móviles es clave para facilitar el despliegue eficiente. A través de esta etapa, el objetivo es mantener la precisión del modelo mientras se reduce el uso de recursos, lo que permite un funcionamiento fluido en entornos con limitaciones de hardware y energía, como smartphones o tablets.

Conversión a formato móvil

La conversión del modelo Vision Transformer a un formato compatible con dispositivos móviles es un paso crucial para permitir su implementación eficiente en estos entornos. En este trabajo se han utilizado dos herramientas principales: TorchScript y ONNX Runtime, que permiten exportar y ejecutar modelos de PyTorch en dispositivos móviles de manera optimizada.

En primer lugar, TorchScript ofrece la capacidad de serializar y exportar el modelo de PyTorch, transformando su estructura a un formato que puede ser cargado y ejecutado sin depender del entorno de desarrollo original. El proceso de conversión se realizó utilizando el mecanismo de trazabilidad de TorchScript. Para ello, se definió un modelo Vision Transformer (ViT) y se creó un tensor de imagen aleatorio como entrada. El modelo fue exportado a TorchScript mediante la función `torch.jit.trace`, que sigue el flujo de datos a través del modelo para capturarla en un formato ejecutable. Este modelo convertido fue guardado en un archivo, lo que permite su reutilización y carga posterior para realizar inferencias, replicando el comportamiento del modelo original de manera eficiente en dispositivos móviles. Después de la carga, el modelo puede procesar nuevas imágenes, asegurando que su salida coincida con la del modelo base.

Por otro lado, se ha utilizado ONNX Runtime, un framework optimizado para ejecutar modelos en diversos dispositivos, que facilita la interoperabilidad

CAPÍTULO 6. RESULTADOS

dad entre diferentes entornos y plataformas. La conversión a ONNX (Open Neural Network Exchange) se realizó exportando el modelo de PyTorch mediante la función `torch.onnx.export`. En este proceso, el modelo fue preparado en modo de evaluación, y se utilizó un tensor de entrada de prueba (dummy input) para realizar la exportación, especificando un conjunto de parámetros como el opset version y los nombres de entrada y salida. La exportación a ONNX permitió aprovechar optimizaciones, como la fusión de operadores y el plegado de constantes. Posteriormente, el modelo ONNX fue cargado utilizando ONNX Runtime, y se realizó la inferencia sobre el tensor de entrada convertido a formato NumPy. El resultado de esta inferencia demostró que el modelo ONNX podía replicar la salida del modelo PyTorch con un rendimiento optimizado, garantizando una implementación eficiente para dispositivos móviles.

En conjunto, la combinación de TorchScript y ONNX Runtime garantiza que el modelo Vision Transformer pueda ser ejecutado en dispositivos móviles sin depender de PyTorch, aprovechando el hardware disponible de manera óptima, reduciendo la latencia y el consumo energético, y facilitando la portabilidad a diversas plataformas móviles.

Integración en aplicaciones móviles y consideraciones de despliegue

Una vez convertido el modelo Vision Transformer a formatos compatibles con dispositivos móviles (TorchScript y ONNX), el siguiente paso es integrarlo en una aplicación móvil para su uso en tiempo real. Este proceso implica varias etapas, desde la inclusión del modelo en la aplicación hasta la optimización del rendimiento y el manejo de recursos móviles.

En el caso de modelos exportados con TorchScript, la integración en aplicaciones móviles desarrolladas con Android o iOS es directa, ya que PyTorch proporciona bibliotecas móviles que permiten la ejecución del modelo nativamente en estos dispositivos. Estas bibliotecas, como PyTorch Mobile, son ligeras y ofrecen la funcionalidad necesaria para cargar el modelo preentrenado y realizar inferencias sobre imágenes capturadas desde la cámara del dispositivo o cargadas desde la galería. La inferencia en tiempo real es clave para muchas aplicaciones, por lo que es crucial optimizar la latencia y el consumo de recursos durante la ejecución del modelo. Para esto, se deben aprovechar optimizaciones específicas del hardware, como el uso de GPU o NPU (Unidad de Procesamiento Neuronal), dependiendo del dispositivo móvil.

Por otro lado, si se utiliza ONNX Runtime para ejecutar el modelo, las aplicaciones móviles pueden beneficiarse de su capacidad para funcionar en diferentes plataformas, incluyendo dispositivos basados en Android, iOS, y

6.3. OPTIMIZACIÓN DE MODELOS PARA DISPOSITIVOS MÓVILES

sistemas embebidos. ONNX Runtime soporta varias optimizaciones como el uso de backends específicos (e.g., OpenVINO, TensorRT) que mejoran el rendimiento de los modelos en dispositivos con hardware especializado. Para integrar el modelo ONNX, es necesario agregar la dependencia de ONNX Runtime Mobile al proyecto de la aplicación, cargar el modelo durante el inicio de la aplicación y utilizar los métodos de inferencia proporcionados por ONNX Runtime para procesar imágenes de entrada. Además, se deben considerar factores como la administración de memoria y la capacidad de procesamiento del dispositivo, asegurando que la aplicación maneje eficientemente la inferencia, incluso con lotes de datos variables.

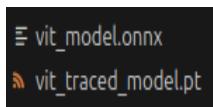


Figura 6.19: Modelos para implementación en dispositivos móviles.

A nivel de despliegue, también es importante evaluar las consideraciones de distribución de la aplicación móvil. Esto incluye:

- **Tamaño del modelo:** Los modelos grandes pueden ocupar mucho espacio en dispositivos móviles. Considera aplicar técnicas de pruning y cuantización para reducir el tamaño del modelo sin sacrificar demasiado rendimiento.
- **Rendimiento:** Asegúrate de medir el rendimiento del modelo en dispositivos reales. Optimiza el modelo si es necesario, utilizando técnicas como la paralelización de operaciones.
- **Uso de recursos:** Los dispositivos móviles tienen recursos limitados. Considera el uso de hilos o procesamiento asíncrono para realizar inferencias sin bloquear la interfaz de usuario.
- **Manejo de errores:** Asegúrate de manejar errores adecuadamente, como fallas al cargar el modelo o problemas con los datos de entrada.
- **Actualizaciones de modelos:** Planifica cómo actualizar tu modelo en la aplicación sin necesidad de que el usuario vuelva a descargar la app. Puedes considerar cargar el modelo desde un servidor.

En resumen, la integración del modelo optimizado en aplicaciones móviles requiere una planificación cuidadosa, asegurando un balance entre rendimiento y eficiencia, y aprovechando las capacidades del hardware móvil para proporcionar una experiencia de usuario fluida y responsiva.

Capítulo 7

Conclusiones y trabajos futuros

En este capítulo se presentan las conclusiones obtenidas a partir del trabajo realizado, así como las limitaciones encontradas y propuestas para futuros trabajos en el ámbito de los modelos Vision Transformer (ViT) en dispositivos móviles.

7.1. Resumen de los principales hallazgos

En este trabajo se han utilizado tanto modelos Vision Transformer (ViT) desarrollados desde cero como aquellos basados en transfer learning. Se logró optimizar ambos tipos de modelos mediante técnicas de pruning, cuantización y binarización, lo que permitió mejorar su eficiencia y rendimiento en dispositivos con recursos limitados. Además, se aplicaron TorchScript y ONNX para facilitar la implementación de los modelos en dispositivos móviles, lo que demuestra el potencial de los modelos Vision Transformer para ser utilizados en entornos de producción donde se requieren soluciones rápidas y eficientes.

7.2. Implicaciones del uso de Vision Transformer en dispositivos móviles

El uso de Vision Transformer en dispositivos móviles representa un avance en el campo de la visión por computadora. La capacidad de estos modelos para captar relaciones complejas en las imágenes permite mejorar la precisión de tareas como la clasificación, detección y segmentación. Al optimizar estos modelos para dispositivos móviles, se pueden implementar en una variedad

de aplicaciones, desde fotografía inteligente hasta sistemas de seguridad, facilitando la ejecución de tareas que anteriormente requerían hardware potente.

En nuestro proyecto, la idea es poder implementar modelos capaces de eliminar el ruido de imágenes realizadas con dispositivos móviles. De esta manera, el propio dispositivo dispondría de un modelo de visión transformer con decoder para generación de imágenes, en las que se les introduzca la imagen realizada por la cámara y sea capaz de generar la imagen sin ruido.

7.3. Limitaciones del trabajo

Una de las principales limitaciones de este trabajo fue la imposibilidad de entrenar el modelo desarrollado desde cero y el modelo de transfer learning, debido a restricciones de memoria en la GPU. Esta limitación afectó la capacidad de evaluar completamente el rendimiento de los modelos optimizados, así como de realizar un análisis exhaustivo de su comportamiento en diferentes configuraciones y conjuntos de datos. Además, aunque se realizaron esfuerzos por aplicar diversas técnicas de optimización, el impacto real de estas en escenarios prácticos aún necesita ser explorado.

7.4. Propuestas de trabajos futuros

Las investigaciones futuras en el ámbito de los modelos Vision Transformer en dispositivos móviles podrían centrarse en las siguientes propuestas:

- **Optimización de modelos:** Continuar explorando y aplicando técnicas avanzadas de optimización, como la reducción de la complejidad computacional y la mejora en la eficiencia energética. Esto podría incluir el desarrollo de arquitecturas híbridas que combinen elementos de modelos clásicos y modernos para maximizar el rendimiento.
- **Mejora de adaptabilidad:** Investigar métodos de ajuste fino que permitan a los modelos adaptarse a diferentes tipos de datos y condiciones de operación en tiempo real. Esto podría implicar el uso de aprendizaje continuo o transfer learning en escenarios dinámicos.
- **Implementación en entornos reales:** Realizar pruebas exhaustivas de los modelos en entornos del mundo real, utilizando dispositivos móviles en situaciones prácticas. Evaluar el rendimiento en condiciones de iluminación variadas, diferentes ángulos de captura y otras variaciones del mundo real ayudará a entender mejor las limitaciones y capacidades de los modelos.

7.4. PROPUESTAS DE TRABAJOS FUTUROS

- **Análisis de rendimiento en el tiempo:** Realizar un seguimiento y análisis del rendimiento de los modelos a lo largo del tiempo para identificar patrones de degradación y proponer soluciones que mantengan su efectividad.

Estas propuestas no solo contribuirán a mejorar el rendimiento y la eficiencia de los modelos Vision Transformer en dispositivos móviles, sino que también facilitarán su adopción en aplicaciones del mundo real, ampliando así el alcance de la visión por computadora en este contexto.

Bibliografía

- [1] Pan, J.; Bulat, A.; Tan, F.; Zhu, X.; Dudziak, L.; Li, H.; Tzimiropoulos, G.; Martinez, B. (2022). *EdgeViTs: Competing Light-weight CNNs on Mobile Devices with Vision Transformers*. The Chinese University of Hong Kong, Samsung AI Cambridge, Queen Mary University of London.
- [2] Li, Y.; Yuan, G.; Wen, Y.; Hu, J.; Evangelidis, G.; Tulyakov, S.; Wang, Y.; Ren, J. (2022). *EfficientFormer: Vision Transformers at MobileNet Speed*. Snap Inc, Northeastern University.
- [3] Chen, Y.; Dai, X.; Chen, D.; Liu, M.; Dong, X.; Yuan, L.; Liu, Z. (2022). *Mobile-Former: Bridging MobileNet and Transformer*. Microsoft, University of Science and Technology of China.
- [4] Mehta, S.; Rastegari, M. (2022). *MobileViT: Light-Weight, General-Purpose, and Mobile-Friendly Vision Transformer*. Apple.
- [5] Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.; Kaiser, L.; Polosukhin, I. (2017). *Attention Is All You Need*. Google, University of Toronto.
- [6] Radford, A.; Kim, J.; Hallacy, C.; Ramesh, A.; Goh, G.; Agarwal, S.; Sastry, G.; Askell, A.; Mishkin, P.; Clark, J.; Krueger, G.; Sutskever, I. (2021). *Learning Transferable Visual Models From Natural Language Supervision*. OpenAI.
- [7] Ma, S.; Wang, H.; Ma, L.; Wang, L.; Wang, W.; Huang, S.; Dong, L.; Wang, R.; Xue, J.; Wei, F. (2024). *The Era of 1-bit LLMs: All Large Language Models are in 1.58 Bits*. Microsoft.
- [8] Howard, A.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; Adam, H. (2017). *MobileNets: Efficient Convolutional Neural Network for Mobile Vision Applications*. Google.

BIBLIOGRAFÍA

- [9] Wadekar, S.; Chaurasia, A. (2022). *MobileViT3: Mobile-Friendly Vision Transformer With Simple and Effective Fusion of Local, Global and Input Features*. Micron Technology.
- [10] Howard, A.; Sandler, M.; Chu, G.; Chen, L.; Chen, B.; Tan, M.; Wang, W.; Zhu, Y.; Pang, R.; Vasudevan, V.; Le, Q.; Adam, H. (2019). *Searching for MobileNetV3*. Google AI, Google Brain.
- [11] Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; Uszkoreit, J.; Houlsby, N. (2020). *An Image is Worth 16x16: Transformers for Image Recognition at Scale*. Google Research, Brain Team.
- [12] Khan, S.; Naseer, M.; Hayat, M.; Zamir, S.; Shahbaz, F.; Shah, M. (2021). *Transformers in Vision: A Survey*. ACM Computing Surveys (CSUR).
- [13] Abdelhamed, A.; Lin, S.; Brown, M. (2018). *A High-Quality Denoising Dataset for Smartphone Cameras*. York University, Microsoft.
- [14] Abdelhamed, A.; Timofte, R.; Brown, M.; Yu, S.; Park, B.; Jeong, J.; Jung, S. (2019). *NTIRE 2019 Challenge on Real Image Denoising: Methods and Results*. Conference on Computer Vision and Pattern Recognition Workshops (CVPRW).
- [15] Nah, S.; Kim, T.; Lee, K. (2016). *Deep Multi-scale Convolutional Neural Network for Dynamic Scene Deblurring*. Seoul National University.
- [16] Carion, N.; Massa, F.; Synnaeve, G.; Usunier, N.; Kirillov, A.; Zagoruyko, S. (2020). *End-to-End Object Detection with Transformers*. Facebook AI.
- [17] Conde, M.; Vasluiianu, F.; Vazquez-Corral, J.; Timofte, R. (2022). *Perceptual Image Enhancement for Smartphone Real-Time Applications*. University of Würzburg, Universitat Autònoma de Barcelona.
- [18] Feng, R.; Li, C.; Chen, H.; Li, S.; Loy, C.; Gu, J. (2021). *Removing Diffraction Image Artifacts in Under-Display Camera via Dynamic Skip Connection Network*. Nanyang Technological University, Tetras AI, Shanghai AI Laboratory.
- [19] Ekman, M. (2022). *Learning Deep Learning: Theory and Practice of Neural Networks, Computer Vision, Natural Language Processing, and Transformers Using TensorFlow*. Boston: Addison-Wesley.

BIBLIOGRAFÍA

- [20] Lakshmanan, V.; Görner, M.; Gillard, R. (2021). *Practical Machine Learning for Computer Vision*. Beijing: O'Reilly.
- [21] Rothman, D. (2024). *Transformers for Natural Language Processing and Computer Vision*. Birmingham-Mumbai: Packt.
- [22] González, R.; Woods, R. (2018). *Digital Image Processing*. New York: Pearson.
- [23] Marcos, A.; Martínez, F.; Pernía, A.; Alba, F.; Castejón, M.; Ordieres, J.; Vergara, E. (2006). *Técnicas y Algoritmos Básicos de Visión Artificial*. Universidad de La Rioja: Logroño.
- [24] Aprende Machine Learning. (2024). *¿Cómo funcionan los Transformers?* Recuperado el 10 de agosto de 2024, desde <https://www.aprendemachinelearning.com/como-funcionan-los-transformers-espanol-nlp-gpt-bert/>
- [25] IBM. (2024). *¿Qué es la visión artificial?* Recuperado el 10 de agosto de 2024, desde <https://www.ibm.com/es-es/topics/computer-vision>
- [26] Encord. (2024). *YOLO Object Detection: Evolution, Algorithm and Applications*. Recuperado el 14 de agosto de 2024, desde <https://encord.com/blog/yolo-object-detection-guide/>
- [27] Hugging Face. (2024). *DETR*. Recuperado el 14 de agosto de 2024, desde <https://huggingface.co/docs/transformers/model-doc/detr>
- [28] Programar fácil. (2024). *Detector de bordes de Canny, cómo contar objetos con OpenCV y Python*. Recuperado el 14 de agosto de 2024, desde <https://programarfácil.com/blog/vision-artificial/detector-de-bordes-canny-opencv/>
- [29] Health Big Data. (2024). *Redes Neuronales Convolucionales*. Recuperado el 14 de agosto de 2024, desde <https://www.juanbarrios.com/redes-neuronales-convolucionales/>
- [30] Instituto de Ingeniería del Conocimiento. (2024). *Transformers en Procesamiento del Lenguaje Natural*. Recuperado el 14 de agosto de 2024, desde <https://www.iic.uam.es/innovacion/transformers-en-procesamiento-del-lenguaje-natural/>
- [31] Linkedin. (2024). *Deep Learning: Aplicación de la Arquitectura VGG*. Recuperado el 14 de agosto de 2024, desde <https://www.linkedin.com/pulse/deep-learningaplicaci>

BIBLIOGRAFÍA

- [32] Juan Sensio. (2024). *Transformers Visuales*. Recuperado el 14 de agosto de 2024, desde <https://juansensio.com/blog/064-vit>
- [33] ResearchGate. (2024). *Architecture of EfficientNet-B0*. Recuperado el 14 de agosto de 2024, desde <https://www.researchgate.net/figure/Architecture-of-EfficientNet-B0-with-MBConv-as-Basic-building-blocks-fig3-356981443>
- [34] Medium. (2024). *Measuring Neural Network Performance: Latency and Throughput on GPU*. Recuperado el 14 de agosto de 2024, desde <https://younsess-elbrag.medium.com/measuring-neural-network-performance-latency-and-throughput-on-gpu-5d54657871f0>
- [35] Jin, W.; Yu, H.; Luo, X. (2024). *CvT-ASSD: Convolutional vision-Transformer Based Attentive Single Shot MultiBox Detector*. University of Shanghai. China.
- [36] Medium. (2024). *Detección de objetos con YOLO: Implementaciones y cómo usarlas*. Recuperado el 14 de agosto de 2024, desde <https://medium.com/@enriqueav/detecci%C3%B3n-de-objetos-con-yolo-implementaciones-y-como-usarlas-c73ca2489246>
- [37] Estrada, J.; Paheding, S.; Yang, X.; Niyaz, Q. (2022). *Deep-Learning-Incorporated Augmented Reality Application for Engineering Lab Training*. Department of Electrical and Computer Engineering. Purdue University Northwest. USA.
- [38] Wang, H.; Ma, S.; Dong, L.; Huang, S.; Wang, H.; Ma, L.; Yang, F.; Wang, R.; Wu, Y.; Wei, F. (2023). *BitNet: Scaling 1-bit Transformers for Large Language Models*. Microsoft Research. University of Chinese Academy of Sciences. China.
- [39] PyTorch. (2024). *Página oficial de PyTorch*. Recuperado el 13 de octubre de 2024, desde <https://pytorch.org/>
- [40] TorchScript. (2024). *Documentación oficial de TorchScript*. Recuperado el 13 de octubre de 2024, desde <https://pytorch.org/docs/stable/jit.html>
- [41] ONNX Runtime. (2024). *Página oficial de ONNX Runtime*. Recuperado el 13 de octubre de 2024, desde <https://onnxruntime.ai/>
- [42] Pytorch Quantization. (2024). *Documentación oficial de PyTorch Quantization*. Recuperado el 13 de octubre de 2024, desde <https://pytorch.org/docs/stable/quantization.html>

BIBLIOGRAFÍA

- [43] PyTorch Pruning. (2024). *Documentación oficial de PyTorch Pruning*. Recuperado el 13 de octubre de 2024, desde <https://pytorch.org/tutorials/intermediate/pruning-tutorial.html>
- [44] Ubuntu v20.04. (2024). *Página oficial de Ubuntu v20.04*. Recuperado el 13 de octubre de 2024, desde <https://releases.ubuntu.com/20.04/>
- [45] Visual Studio Code. (2024). *Página oficial de Visual Studio Code*. Recuperado el 13 de octubre de 2024, desde <https://code.visualstudio.com/>
- [46] SIDD Dataset. (2024). *Página oficial del dataset SIDD*. Recuperado el 13 de octubre de 2024, desde <https://abdokamel.github.io/sidd/>
- [47] GoPro Dataset. (2024). *Página oficial del dataset GoPro*. Recuperado el 13 de octubre de 2024, desde <https://seungjunnah.github.io/Datasets/gopro>
- [48] Datasets y DataLoaders. (2024). *Documentación oficial de Datasets y DataLoaders*. Recuperado el 13 de octubre de 2024, desde <https://pytorch.org/tutorials/beginner/basics/data-tutorial.html>
- [49] Zero To Mastery. (2024). *Zero To Mastery Learn PyTorch for Deep Learning*. Recuperado el 13 de octubre de 2024, desde <https://www.learnpytorch.io/>
- [50] Steiner, A.; Kolesnikov, A.; Zhai, X.; Wightman, R.; Uszkoreit, J.; Beyer, L. (2022). *How to train your ViT? Data, Augmentation, and Regularization in Vision Transformers*. Google Research. Brain Team, Zürich. USA.
- [51] Canny, J. (2009). *A Computational Approach to Edge Detection*. Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA.
- [52] Kanopoulos, N.; Vasanthavada, N.; Baker, R.L. (1988). *Design of an image edge detection filter using the Sobel operator*. Center of Digital Systems Research, Research and Triangle Institute, NC, USA.
- [53] Shehata, A.; Mohammad, S.; Sameer, M.; Ehab, M. (2015). *A Survey on Hough Transform, Theory, Techniques and Applications*. Informatics Department, Electronics Research Institute. Egypt.
- [54] LeCun, Y.; Kavukcuoglu, K.; Farabet, C. (2010). *Convolutional Networks and Applications in Vision*. Computer Science Department, Courant Institute of Mathematical Sciences, New York University. USA.

BIBLIOGRAFÍA

- [55] Krizhevsky, A.; Sutskever, I.; Hinton, G. (2012). *ImageNet Classification with Deep Convolutional Neural Networks*. University of Toronto. USA.
- [56] Simonyan, K.; Zisserman, A. (2015). *Very Deep Convolutional Networks for Large-Scale Image Recognition*. Visual Geometry Group, Department of Engineering Science, University of Oxford. UK.
- [57] He, K.; Zhang, X.; Ren, S.; Sun, J. (2015). *Deep Residual Learning for Image Recognition*. Microsoft Research. USA.
- [58] Touvron, H.; Cord, M.; Douze, M.; Massa, F.; Sablayrolles, A.; Jégou, H. (2021). *Training data-efficient image transformers and distillation through attention*. Facebook AI. Sorbonne University. USA.
- [59] Liu, Z.; Lin, Y.; Cao, Y.; Hu, H.; Wei, Y.; Zhang, Z.; Lin, S.; Guo, B. (2021). *Swin Transformer: Hierarchical Vision Transformer using Shifted Windows*. Microsoft Research. Asia.
- [60] Tan, M.; Le, Q. (2020). *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. Google Research, Brain Team, Mountain View, CA. USA.
- [61] Sartori, E. (2003). *Hybrid Transformers*. Bell Telephone Laboratories, Inc., Whippny, NJ. USA.
- [62] Habib, G.; Jan, T.; Lall, B. (2023). *Knowledge Distillation in Vision Transformers: A Critical Review*. Bharti School of Telecommunication Technology and Management. India.
- [63] Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. (2016). *You Only Look Once: Unified, Real-Time Object Detection*. University of Washington. Allen Institute for AI. Facebook AI Research. USA.
- [64] Tailor, S.; Lane, N.; Fernandez-Marques, J. (2021). *Degree-Quant: Quantization-Aware Training for Graph Neural Networks*. Department of Computer Science and Technology. Department of Computer Science. Cambridge. Oxford.
- [65] Gautam, R.; Murali, S. (2016). *An Optimized Huffman's Coding by the method of Grouping*. Maharaja Institute of Technology. Mysore.
- [66] Tang, Y.; Wang, Y.; Guo, J.; Tu, Z.; Han, K.; Hu, H.; Tao, D. (2024). *A Survey on Transformer Compression*. Huawei Noah's Ark Lab. University of Sydney. China. Australia.

BIBLIOGRAFÍA

- [67] Noach, M.; Goldberg, Y. (2020). *Compressing Pre-trained Language Models by Matrix Decomposition*. Intel AI Lab. Allen Institute for Artificial Intelligence. Israel.



viu

Universidad
Internacional
de Valencia