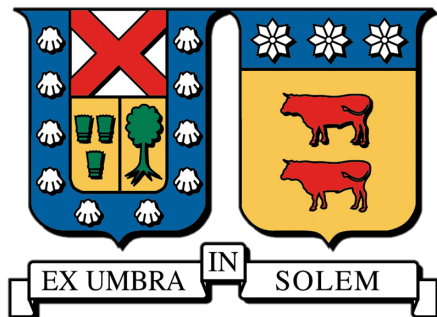


UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE ELECTRÓNICA
VALPARAÍSO - CHILE



**“EVALUACIÓN DE MODELOS DE INFERENCIA LIGEROS PARA
PROCESAMIENTO DE IMÁGENES SATELITALES A BORDO:
BENCHMARKING Y OPTIMIZACIÓN EN PLATAFORMAS CUBESAT”**

DIEGO AGUSTÍN HERRERA LÓPEZ

**MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE INGENIERO CIVIL
ELECTRÓNICO, MENCIÓN COMPUTADORES**

PROFESOR GUÍA: ALEJANDRO WEINSTEIN OPPENHEIMER

PROFESOR CORREFERENTE: ROBERTO SILVA ESPINOZA

AGOSTO - 2026



CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☒ Memoria o trabajo de título ☐ Tesis de Postgrado

Título del trabajo: "Evaluación de modelos de inferencia ligeros para procesamiento de imágenes satelitales a bordo: Benchmarking y optimización en plataformas CubeSat"

Nombre del candidato(a): Diego Agustín Herrera López

Carrera / Grado: Ingeniería Civil Electrónica

Campus: Casa Central Departamento: Electrónica

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Alejandro Weinstein, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente

DEJO CONSTANCIA que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución.

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL (marcar una opción)

☒ El trabajo **NO contiene** información que amerite confidencialidad y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (**embargo**) por (**marcar una opción**):

☐ 6 meses ☐ 12 meses ☐ 2 años ☐ 3 años ☐ 5 años ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: 22-10-2026 Firma: 

Estudiante o Candidato(a):

Fecha: 20-10-2026 Firma: 

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

Resumen

El procesamiento de imágenes satelitales a bordo permite reducir el volumen de información transmitida a Tierra y acelerar la identificación de datos de interés. Sin embargo, su implementación en plataformas CubeSat está condicionada por restricciones de potencia y capacidad computacional. En este contexto, esta memoria evalúa alternativas de inferencia para clasificación binaria de imágenes entre las clases “wildfire” y “nowildfire”, considerando conjuntamente su desempeño predictivo y costo computacional.

Para establecer una referencia común, se realizó un benchmark en CPU de seis modelos y se seleccionaron una CNN ligera, MobileNetV2 y ResNet50. Estos modelos fueron exportados a ONNX y optimizados mediante TensorRT en precisiones FP32, FP16 e INT8 cuando fue técnicamente viable. Las variantes resultantes se evaluaron sobre 6299 imágenes válidas del test split, se caracterizaron en una RTX 3060 Laptop y se desplegaron en una NVIDIA Jetson Xavier NX bajo perfiles nominales de 10 W y 15 W.

A partir de estas evaluaciones, FP16 se identificó como la optimización más consistente, ya que conservó prácticamente el desempeño predictivo de FP32 y, al mismo tiempo, redujo latencia y energía por inferencia en las tres arquitecturas. Por su parte, INT8 produjo beneficios adicionales en la CNN ligera, aunque no fue posible obtener variantes utilizables para MobileNetV2 y ResNet50. En la Xavier NX, CNN INT8 presentó la menor latencia y energía incremental entre las variantes evaluadas. En contraste, ResNet50 FP16 alcanzó el mayor desempeño predictivo global, mientras que CNN FP16 obtuvo el mayor recall para “wildfire”. Asimismo, las mediciones de potencia se mantuvieron dentro del margen considerado en el escenario CubeSat de referencia.

En conjunto, los resultados muestran que no existe una única configuración óptima, sino compromisos entre desempeño predictivo, latencia, throughput y consumo energético. La selección depende de las prioridades de la aplicación y de las restricciones de la plataforma de ejecución.

Palabras clave: CubeSat, edge computing, incendios forestales, TensorRT, Jetson Xavier NX.

Abstract

On-board satellite image processing can reduce the amount of information transmitted to ground stations and accelerate the identification of relevant data. However, its implementation on CubeSat platforms is constrained by limited power and computing resources. In this context, this thesis evaluates inference alternatives for binary image classification between the “wildfire” and “nowildfire” classes, jointly considering predictive performance and computational cost.

To establish a common reference, a CPU benchmark was conducted on six models, from which a lightweight CNN, MobileNetV2, and ResNet50 were selected. These models were exported to ONNX and optimized with TensorRT using FP32, FP16, and INT8 precision when technically feasible. The resulting variants were evaluated on 6299 valid images from the test split, characterized on an RTX 3060 Laptop GPU, and deployed on an NVIDIA Jetson Xavier NX under nominal 10 W and 15 W power profiles.

Based on these evaluations, FP16 was identified as the most consistent optimization, as it preserved almost all of the FP32 predictive performance while reducing latency and energy per inference across the three architectures. INT8 provided additional benefits for the lightweight CNN, although usable variants could not be obtained for MobileNetV2 and ResNet50. On the Xavier NX, the INT8 CNN achieved the lowest latency and incremental energy among the evaluated variants. In contrast, ResNet50 FP16 attained the highest overall predictive performance, while CNN FP16 achieved the highest recall for “wildfire”. In addition, the measured power values remained within the margin considered in the reference CubeSat scenario.

Overall, the results show that there is no single optimal configuration, but rather trade-offs between predictive performance, latency, throughput, and energy consumption. The final selection depends on the priorities of the application and the constraints of the execution platform.

Keywords: CubeSat, edge computing, wildfires, TensorRT, Jetson Xavier NX.

Índice

1	Introducción	1
1.1	Contexto y motivación	1
1.2	Planteamiento del problema	2
1.3	Objetivo general	3
1.4	Objetivos específicos	3
1.5	Metodología general	3
1.6	Alcances y limitaciones	4
1.7	Estructura del documento	5
2	Marco teórico y estado del arte	6
2.1	Procesamiento a bordo y restricciones en plataformas CubeSat	6
2.2	Clasificación de imágenes con redes neuronales convolucionales	7
2.2.1	Operaciones convolucionales	7
2.2.2	Reducción espacial y clasificación	8
2.2.3	Representación de los tensores	9
2.3	Arquitecturas consideradas	10
2.3.1	Redes convolucionales personalizadas	10
2.3.2	MobileNetV2	10
2.3.3	DenseNet121	11
2.3.4	ResNet50	12
2.4	Métricas de evaluación	13
2.4.1	Matriz de confusión y desempeño predictivo	13
2.4.2	Latencia y tasa de procesamiento	14
2.4.3	Potencia y energía	14

2.5	Optimización y despliegue de inferencia	15
2.5.1	ONNX como representación intermedia	15
2.5.2	TensorRT y engines	15
2.5.3	Precisión numérica	16
2.5.4	Cuantización INT8	16
2.5.5	Ejecución sobre GPU	17
2.5.6	Condiciones de ejecución	17
2.6	Estado del arte	17
2.6.1	Inferencia de redes neuronales a bordo	18
2.6.2	Inteligencia artificial aplicada a incendios en sistemas restringidos	18
2.6.3	Posicionamiento del presente trabajo	19
3	Metodología experimental	21
3.1	Diseño experimental y flujo general	21
3.2	Dataset y preparación de entradas	22
3.2.1	Dataset	22
3.2.2	Preparación de entradas para las campañas de inferencia	23
3.3	Modelos evaluados y preprocesamiento	24
3.3.1	Modelos evaluados	24
3.3.2	Preprocesamiento y representación de las salidas	25
3.4	Benchmark inicial en CPU	27
3.4.1	Entorno y condiciones de ejecución	27
3.4.2	Evaluación de desempeño predictivo	28
3.4.3	Medición de latencia	29
3.5	Selección de modelos para optimización	30
3.6	Conversión, validación y optimización con ONNX y TensorRT	31
3.6.1	Exportación a ONNX y validación	32
3.6.2	Construcción de variantes TensorRT	32
3.6.3	Cuantización INT8	33
3.6.4	Variantes resultantes	34
3.7	Metodología experimental en RTX 3060	35
3.7.1	Plataforma y condiciones de ejecución	35

3.7.2	Medición de latencia y flujo de procesamiento completo	35
3.7.3	Carga sostenida, telemetría y energía	36
3.8	Despliegue y metodología en Jetson Xavier NX	37
3.8.1	Plataforma, entradas y compatibilidad	37
3.8.2	Perfiles de potencia y construcción de engines	38
3.8.3	Validación de paridad	39
3.8.4	Medición de latencia	39
3.8.5	Carga sostenida, telemetría y energía	39
3.9	Criterios de consolidación y comparación de resultados	40
4	Resultados y discusión	42
4.1	Resultados del benchmark inicial en CPU	42
4.1.1	Desempeño predictivo	42
4.1.2	Latencia y complejidad de los modelos	44
4.1.3	Resultado de la selección inicial	46
4.2	Efecto de la optimización sobre el desempeño predictivo	46
4.2.1	Comportamiento de las variantes FP16	47
4.2.2	Comportamiento de la cuantización INT8	47
4.3	Caracterización experimental en RTX 3060	48
4.3.1	Latencia de inferencia y flujo de procesamiento completo	48
4.3.2	Tasa de procesamiento y utilización de recursos	50
4.3.3	Energía por inferencia	51
4.4	Resultados en Jetson Xavier NX	53
4.4.1	Validación del despliegue	53
4.4.2	Latencia según el perfil de potencia	54
4.4.3	Tasa de procesamiento sostenida	55
4.4.4	Energía incremental por inferencia	57
4.4.5	Potencia efectiva de entrada del módulo	59
4.5	Análisis comparativo y discusión	60
4.5.1	Compromisos entre desempeño predictivo y computacional	60
4.5.2	Efecto del perfil de potencia	62
4.5.3	Relación con el escenario energético de referencia	62

4.5.4	Síntesis de la comparación	63
5	Conclusiones y trabajo futuro	65
5.1	Conclusiones	65
5.2	Trabajo futuro	67

Índice de figuras

2.1	Esquema general del proceso de inferencia de una red neuronal convolucional para clasificación de imágenes.	9
2.2	Representación simplificada de un bloque utilizado en MobileNetV2.	11
2.3	Representación simplificada de una conexión residual.	12
2.4	Representación conceptual de la matriz de confusión para la clasificación binaria utilizada en este trabajo.	13
2.5	Flujo conceptual de conversión y despliegue mediante ONNX y TensorRT.	16
2.6	Representación conceptual del uso de nodos Q/DQ en un modelo cuantizado.	17
3.1	Flujo general de la metodología experimental desarrollada en este trabajo.	22
3.2	Ejemplos de imágenes pertenecientes a las clases “nowildfire” y “wildfire” del dataset utilizado en la evaluación.	22
4.1	Comparación de exactitud y sensibilidad para la clase “wildfire” en el benchmark inicial en CPU.	44
4.2	Relación entre latencia de inferencia y F1 para los modelos evaluados en CPU. El tamaño del marcador representa el tamaño del archivo original y el eje de latencia se presenta en escala logarítmica.	45
4.3	Comparación de las distintas fronteras de latencia para las variantes evaluadas en la RTX 3060.	50
4.4	Relación entre tasa de procesamiento, energía incremental por inferencia y utilización de memoria gráfica en la RTX 3060. El tamaño de los marcadores representa la VRAM utilizada y el eje de energía se presenta en escala logarítmica.	52
4.5	Latencia del engine de TensorRT para las variantes evaluadas bajo los tres perfiles de la Jetson Xavier NX.	55

4.6	Tasa de procesamiento sostenida de las variantes evaluadas bajo los tres perfiles de la Jetson Xavier NX.	57
4.7	Energía incremental por inferencia de las variantes evaluadas bajo los tres perfiles de la Jetson Xavier NX.	58

Índice de tablas

3.1	Distribución de imágenes del test split utilizado en la evaluación.	23
3.2	Modelos considerados en el benchmark inicial.	25
3.3	Condiciones utilizadas en el benchmark inicial en CPU.	28
3.4	Modelos seleccionados para las etapas de optimización y despliegue.	31
3.5	Variantes TensorRT utilizadas en las campañas experimentales posteriores.	34
3.6	Plataforma utilizada en la campaña experimental RTX 3060.	35
3.7	Protocolo utilizado para las mediciones de carga sostenida en RTX 3060.	36
3.8	Plataforma utilizada en la campaña experimental Jetson Xavier NX.	37
3.9	Perfiles de potencia utilizados en la Jetson Xavier NX.	38
4.1	Resultados de desempeño predictivo y matriz de confusión del benchmark inicial en CPU.	43
4.2	Latencia y características de los modelos evaluados en CPU.	44
4.3	Desempeño predictivo y matriz de confusión de las variantes TensorRT evaluadas sobre el test split.	47
4.4	Latencia y tamaño de los engines de TensorRT en la RTX 3060.	49
4.5	Resultados de carga sostenida en la RTX 3060.	50
4.6	Energía por inferencia obtenida durante la carga sostenida en la RTX 3060.	51
4.7	Validación de paridad de los engines de TensorRT en Jetson Xavier NX.	53
4.8	Latencia del engine de TensorRT según el perfil operacional de la Jetson Xavier NX.	54
4.9	Tasa de procesamiento sostenida según el perfil operacional de la Jetson Xavier NX.	55
4.10	Energía incremental por inferencia según el perfil operacional de la Jetson Xavier NX.	57
4.11	Potencia efectiva de entrada de la Jetson Xavier NX durante carga sostenida.	59
4.12	Comparación de alternativas optimizadas.	61

Glosario

Exactitud	Proporción de predicciones correctas respecto del total de muestras evaluadas.
Precisión	Proporción de predicciones positivas que corresponden efectivamente a la clase positiva.
Sensibilidad	Proporción de muestras realmente positivas que son identificadas correctamente.
Puntuación F1	Media armónica entre precisión y sensibilidad.
Latencia	Tiempo requerido para completar una inferencia dentro de una frontera de medición definida.
Tasa de procesamiento	Cantidad de inferencias o imágenes procesadas por unidad de tiempo. En esta memoria se expresa en imágenes por segundo.
Referencia inicial	Condición o implementación utilizada como referencia inicial para una comparación posterior.
Benchmark	Procedimiento de evaluación realizado bajo condiciones controladas para comparar modelos o configuraciones.
Conjunto de datos	Conjunto de datos utilizado para entrenar, validar o evaluar un modelo.
Partición de entrenamiento	Partición de entrenamiento del conjunto de datos original. En esta memoria se utilizó para obtener muestras de calibración INT8.
Partición de validación	Partición utilizada para evaluar configuraciones durante etapas previas de selección.
Partición de prueba	Partición utilizada para la evaluación comparativa de los modelos y variantes funcionales.

Preprocesamiento	Conjunto de transformaciones aplicadas a una imagen antes de obtener el tensor de entrada requerido por el modelo.
Flujo de procesamiento	Secuencia completa de etapas necesarias para transformar los datos de entrada y producir una salida de inferencia.
Entorno de ejecución	Entorno encargado de ejecutar un modelo o un motor de inferencia sobre una plataforma determinada.
TensorRT	Herramienta de NVIDIA utilizada para optimizar y ejecutar redes neuronales mediante motores de inferencia adaptados al hardware de destino.
ONNX Runtime	Entorno de ejecución utilizado para ejecutar y validar modelos representados en ONNX.
Motor de inferencia	Artefacto de inferencia construido por TensorRT para una plataforma y configuración determinadas.
Host	Procesador o sistema que controla la ejecución y prepara los datos para el acelerador.
Device	Dispositivo acelerador sobre el que se ejecutan las operaciones de inferencia.
Tamaño del lote	Cantidad de entradas procesadas conjuntamente en una ejecución.
Calentamiento	Ejecuciones iniciales realizadas antes de una medición para estabilizar el entorno de ejecución y el hardware.
Speedup	Relación entre el tiempo de una referencia y el tiempo de una configuración acelerada.
Logit	Valor de salida de una red antes de su transformación a una probabilidad mediante una función sigmoide o una función <i>softmax</i> .

Siglas

CNN	Red neuronal convolucional.
CPU	Unidad central de procesamiento.
GPU	Unidad de procesamiento gráfico.
RAM	Memoria de acceso aleatorio utilizada como memoria principal del sistema.
VRAM	Memoria de video dedicada a una GPU discreta.
RGB	Modelo de color basado en los canales rojo, verde y azul.
CIELAB	Espacio de color CIE $L^*a^*b^*$, utilizado para representar luminosidad y componentes cromáticas.
CLAHE	Ecualización adaptativa de histograma con limitación de contraste.
NHWC	Disposición de tensor en el orden batch, altura, ancho y canales.
NCHW	Disposición de tensor en el orden batch, canales, altura y ancho.
ONNX	Open Neural Network Exchange, formato utilizado como representación intermedia de modelos.
JPEG	Formato de compresión y almacenamiento de imágenes digitales utilizado por el dataset principal.
CUDA	Plataforma de cómputo paralelo de NVIDIA utilizada para ejecutar operaciones sobre GPU.
FP32	Formato de punto flotante de 32 bits.
FP16	Formato de punto flotante de 16 bits.
TF32	Formato TensorFloat-32 disponible en determinadas GPU NVIDIA.
INT8	Representación entera de 8 bits utilizada en inferencia cuantizada.
QAT	<i>Quantization-Aware Training</i> , entrenamiento con simulación de los efectos de la cuantización.

PTQ	Post-Training Quantization, cuantización aplicada después del entrenamiento del modelo.
Q/DQ	Nodos Quantize/Dequantize utilizados para representar explícitamente operaciones de cuantización y de-cuantización.
H2D	Transferencia de datos desde el host hacia el device.
D2H	Transferencia de datos desde el device hacia el host.
DVFS	Dynamic Voltage and Frequency Scaling, mecanismo de ajuste dinámico de voltaje y frecuencia.
DLA	Deep Learning Accelerator, acelerador dedicado disponible en algunas plataformas NVIDIA Jetson.
L4T	Linux for Tegra, distribución base utilizada por NVIDIA Jetson.
VDD_IN	Rail de telemetría utilizado para representar la potencia total de entrada del módulo Jetson Xavier NX.
TN	True Negative. Muestra de la clase “nowildfire” clasificada correctamente como negativa.
TP	True Positive. Muestra de la clase “wildfire” clasificada correctamente como positiva.
FP	False Positive. Muestra de la clase “nowildfire” clasificada incorrectamente como positiva.
FN	False Negative. Muestra de la clase “wildfire” clasificada incorrectamente como negativa.

Capítulo 1

Introducción

1.1 Contexto y motivación

Los sistemas de observación de la Tierra generan grandes volúmenes de información que habitualmente deben almacenarse a bordo y transmitirse a estaciones terrestres antes de ser procesados. Esta dependencia de los enlaces de comunicación puede limitar la oportunidad con que la información se encuentra disponible y obliga a utilizar parte del enlace descendente para transferir datos que no necesariamente contienen información relevante. Una alternativa consiste en incorporar capacidades de procesamiento a bordo, permitiendo analizar los datos cerca de su fuente y seleccionar o priorizar la información que será transmitida a Tierra [1, 2].

Este enfoque resulta especialmente atractivo en plataformas CubeSat, donde las restricciones de tamaño, potencia eléctrica, memoria y capacidad computacional condicionan los sistemas que pueden integrarse a bordo. Estas limitaciones adquieren particular importancia al incorporar modelos de aprendizaje profundo, cuyo desempeño no depende solamente de su capacidad predictiva, sino también de los recursos necesarios para ejecutar cada inferencia [3, 4].

Dentro de este contexto, los incendios forestales constituyen una aplicación de interés para el procesamiento de información mediante técnicas de visión por computador. La posibilidad de clasificar imágenes a bordo permitiría realizar una primera discriminación de su contenido antes de la transmisión, reduciendo la necesidad de tratar todas las observaciones de la misma manera. Sin embargo, diferentes arquitecturas de redes neuronales pueden presentar compromisos considerablemente distintos entre desempeño predictivo, tamaño, latency y utilización de recursos [5, 6].

En este marco, la presente memoria se concentra específicamente en la evaluación experimental

de modelos de inferencia para clasificación binaria de imágenes entre las clases “wildfire” y “nowildfire”. Se estudia su comportamiento en distintas plataformas de ejecución y bajo diferentes precisiones numéricas, considerando no solamente el desempeño predictivo, sino también latency, throughput, utilización de memoria y consumo energético. De esta forma, el análisis busca determinar los compromisos que presentan distintas alternativas de inferencia cuando deben operar bajo las restricciones de recursos propias de una plataforma embebida.

1.2 Planteamiento del problema

La selección de un modelo de aprendizaje profundo para una plataforma con recursos limitados no puede realizarse a partir de una única métrica de desempeño. Dos arquitecturas pueden presentar resultados similares en términos de clasificación y, al mismo tiempo, diferir considerablemente en cantidad de parámetros, tamaño, tiempo de inferencia y demanda de recursos. En consecuencia, una arquitectura con una exactitud ligeramente superior no constituye necesariamente la alternativa más viable cuando existen restricciones computacionales y energéticas.

La aplicación de técnicas de cuantización y reducción de precisión numérica introduce además un compromiso adicional. El uso de FP16 permite representar los valores mediante un formato de punto flotante de menor precisión, mientras que INT8 requiere un proceso de cuantización hacia una representación entera de 8 bits. Estas técnicas pueden aprovechar capacidades especializadas del hardware de aceleración, pero su efecto depende tanto de la arquitectura como del entorno de ejecución. Como resultado, pueden producir desde una conservación prácticamente completa de las predicciones hasta degradaciones del desempeño predictivo o incompatibilidades durante el despliegue [6, 2]. Por esta razón, sus beneficios deben verificarse experimentalmente en lugar de asumirse de manera general.

En una aplicación asociada a incendios forestales, la comparación de modelos tampoco puede limitarse a la exactitud global. El recall correspondiente a la clase “wildfire” y la cantidad de falsos negativos resultan especialmente relevantes, puesto que un falso negativo representa una imagen perteneciente a dicha clase que es clasificada como “nowildfire”. Por lo tanto, la evaluación debe considerar conjuntamente las distintas métricas de desempeño predictivo y las prestaciones computacionales de cada alternativa.

A estas restricciones se suma la disponibilidad energética del escenario de aplicación. Como referencia para contextualizar esta dimensión, Silva estima para una misión CubeSat 12U específica un

margen de potencia del orden de 19 W disponible para una carga adicional de computación de borde durante su ventana de procesamiento [7]. Este valor no corresponde a una especificación general del estándar CubeSat ni al consumo total del satélite, sino a una estimación obtenida para las condiciones operacionales particulares estudiadas en dicho trabajo. Por esta razón, sirve como referencia para contextualizar el consumo de una plataforma embebida, sin asumir que esta reproduzca integralmente el sistema satelital considerado. Asimismo, un perfil nominal de potencia configurado en la plataforma no garantiza por sí mismo el cumplimiento de dicho margen, por lo que resulta necesario contrastarlo con el consumo efectivo observado durante la ejecución.

El problema abordado en esta memoria consiste, por tanto, en determinar experimentalmente cómo diferentes arquitecturas de clasificación y precisiones numéricas se comportan al ser ejecutadas en plataformas de cómputo con distintas capacidades, estableciendo los compromisos existentes entre desempeño predictivo, latency, throughput, utilización de memoria y consumo energético bajo condiciones de recursos restringidos.

1.3 Objetivo general

Evaluar y comparar modelos de inferencia ligeros para el procesamiento de imágenes a bordo de plataformas CubeSat, considerando métricas de desempeño predictivo, latency, utilización de recursos y consumo energético.

1.4 Objetivos específicos

1. Implementar una metodología de benchmarking que permita obtener métricas relevantes de desempeño predictivo y desempeño computacional para modelos de inferencia ejecutados bajo condiciones de recursos restringidos.
2. Evaluar el impacto de técnicas de optimización de inferencia sobre el desempeño predictivo y el desempeño computacional de los modelos, considerando diferentes precisiones numéricas y plataformas de ejecución.

1.5 Metodología general

El desarrollo del trabajo se estructuró en cuatro etapas principales:

1. **Benchmark inicial de modelos:** se evaluaron seis modelos existentes de clasificación binaria de imágenes bajo un entorno controlado y condiciones comunes de ejecución en CPU. Para cada modelo se mantuvo su preprocesamiento original y se obtuvieron métricas de desempeño predictivo, latency, cantidad de parámetros y tamaño, estableciendo un baseline para su comparación.
2. **Selección y optimización de modelos:** a partir de los resultados del benchmark se seleccionaron tres arquitecturas representativas de distintos compromisos entre desempeño predictivo y costo computacional. Los modelos seleccionados fueron adaptados a un pipeline de inferencia mediante ONNX y TensorRT, verificando la correspondencia de sus salidas antes de evaluar variantes en FP32, FP16 y, cuando fue técnicamente viable, INT8.
3. **Caracterización en GPU de desarrollo:** las variantes funcionales fueron evaluadas en una GPU NVIDIA RTX 3060 mediante una metodología común de benchmarking. Esta etapa permitió caracterizar latency, throughput, utilización de memoria y consumo energético, además de estudiar el efecto de las distintas precisiones numéricas sobre el desempeño predictivo y el desempeño computacional.
4. **Despliegue y evaluación en plataforma embebida:** las variantes seleccionadas fueron posteriormente desplegadas en una NVIDIA Jetson Xavier NX. Se verificó la correspondencia de las inferencias con las referencias obtenidas previamente y se caracterizó su desempeño bajo distintos perfiles de potencia, considerando latency, throughput, potencia de entrada y energía por inferencia.

Finalmente, los resultados de las distintas etapas fueron analizados de manera conjunta para estudiar los compromisos entre desempeño predictivo y requerimientos computacionales, y contextualizar el comportamiento observado en la plataforma embebida respecto de las restricciones consideradas para el escenario de procesamiento a bordo.

1.6 Alcances y limitaciones

El presente trabajo comprende la evaluación experimental de modelos existentes de clasificación binaria de imágenes, su comparación mediante una metodología común de benchmarking y la aplicación de técnicas de optimización orientadas a inferencia acelerada. El análisis considera desempeño predictivo y desempeño computacional en CPU, en una GPU NVIDIA RTX 3060

utilizada como plataforma de desarrollo y en una plataforma embebida NVIDIA Jetson Xavier NX. No se contempla el entrenamiento de nuevas arquitecturas ni la creación de un nuevo dataset.

La aplicación se restringe a la clasificación entre las clases “wildfire” y “nowildfire”. El dataset utilizado para la evaluación principal está compuesto por imágenes JPEG de un dataset público, por lo que los resultados no constituyen una validación operacional completa sobre imágenes adquiridas directamente por una misión satelital. Asimismo, no se abordan tareas de detección de objetos, segmentación, propagación del fuego ni susceptibilidad de ignición.

Las plataformas empleadas corresponden a entornos experimentales de referencia. La Jetson Xavier NX es una plataforma embebida comercial, por lo que no se evalúa su integración física en un CubeSat, efectos de radiación ni condiciones orbitales. Las mediciones energéticas caracterizan el consumo del módulo durante la ejecución de las cargas evaluadas y se contextualizan mediante el escenario CubeSat 12U descrito por Silva [7], sin asumir que este represente una especificación general para todas las plataformas CubeSat.

1.7 Estructura del documento

El presente documento se organiza en cinco capítulos. El Capítulo 1 introduce el contexto y la motivación del trabajo, plantea el problema de investigación y presenta los objetivos, la metodología general y los alcances del estudio. El Capítulo 2 desarrolla el marco teórico necesario para comprender el procesamiento de imágenes mediante modelos de aprendizaje profundo en plataformas con recursos restringidos y presenta una revisión del estado del arte relacionado con inferencia eficiente y procesamiento a bordo. El Capítulo 3 describe la metodología y el desarrollo experimental, incluyendo el dataset y los modelos utilizados, el benchmark inicial, el proceso de selección y optimización de los modelos, y los procedimientos de evaluación implementados en las distintas plataformas de cómputo. El Capítulo 4 presenta y discute los resultados obtenidos, comparando el desempeño predictivo y el desempeño computacional de las variantes evaluadas, así como su comportamiento en la plataforma embebida y su relación con las restricciones energéticas del escenario considerado. Finalmente, el Capítulo 5 reúne las principales conclusiones del trabajo, analiza el cumplimiento de los objetivos planteados y propone posibles líneas de trabajo futuro.

Capítulo 2

Marco teórico y estado del arte

Este capítulo presenta los conceptos necesarios para interpretar la evaluación experimental. Se abordan el procesamiento a bordo y las restricciones de plataformas CubeSat, los fundamentos de clasificación mediante CNN, las arquitecturas consideradas y las métricas de evaluación. Finalmente, se introducen las herramientas de optimización y se revisan trabajos relacionados con inteligencia artificial en sistemas embebidos y aplicaciones satelitales.

2.1 Procesamiento a bordo y restricciones en plataformas CubeSat

En los sistemas de observación de la Tierra, los datos adquiridos por los sensores pueden ser almacenados a bordo y transmitidos posteriormente a estaciones terrestres para su procesamiento. Este esquema concentra una parte importante del análisis en el segmento terrestre y condiciona la disponibilidad de la información a las oportunidades de comunicación. El procesamiento a bordo modifica esta cadena al ejecutar parte del análisis directamente en la plataforma satelital antes de la transmisión, permitiendo reducir, seleccionar o priorizar los datos que deben enviarse a Tierra [1, 2].

Cuando el análisis incluye modelos de aprendizaje automático, la plataforma satelital puede entenderse como un nodo de computación de borde, ya que la inferencia se realiza próxima a la fuente donde se generan los datos. En este esquema no es necesario trasladar el entrenamiento de los modelos al satélite. Un modelo previamente entrenado puede ser desplegado para ejecutar únicamente la etapa de inferencia y generar localmente una clasificación, detección u otra salida de interés. Esta separación resulta relevante en sistemas con recursos restringidos, donde el costo de ejecutar una inferencia debe evaluarse junto con la utilidad de la información producida [1, 6].

La implementación de estas capacidades se encuentra condicionada por los recursos disponibles

a bordo. Las plataformas CubeSat presentan restricciones asociadas al volumen físico, la potencia eléctrica, la capacidad de procesamiento, la memoria y la disipación térmica. En consecuencia, incorporar un modelo de aprendizaje profundo requiere considerar no solamente su desempeño predictivo, sino también el tiempo y los recursos necesarios para ejecutarlo [3, 4]. Estas restricciones favorecen el uso de arquitecturas y técnicas de optimización capaces de disminuir el costo computacional de la inferencia sin producir una degradación significativa de la salida del modelo.

En este trabajo, estas limitaciones se estudian a través de métricas observables de desempeño predictivo y desempeño computacional. La plataforma embebida utilizada experimentalmente no se considera equivalente a un sistema de vuelo ni se evalúa su calificación para el entorno espacial. Su utilización permite caracterizar el comportamiento de las variantes de inferencia bajo condiciones de recursos y potencia más restringidas que las de una GPU discreta, mientras que la relación con el escenario CubeSat se establece posteriormente mediante el presupuesto energético utilizado como referencia.

2.2 Clasificación de imágenes con redes neuronales convolucionales

Las CNN constituyen una familia de modelos ampliamente utilizada para el procesamiento de imágenes. Su principal característica es el uso de operaciones de convolución que permiten extraer patrones espaciales a partir de regiones locales de la entrada, evitando la necesidad de conectar directamente cada píxel con todas las unidades de la red [8, 9].

En una tarea de clasificación, una imagen es transformada inicialmente en un tensor numérico que representa sus dimensiones espaciales y sus canales. Para una imagen RGB, cada posición contiene información asociada a los canales rojo, verde y azul. Antes de ingresar al modelo, este tensor puede ser redimensionado, normalizado o sometido a otras transformaciones según el procedimiento definido durante el desarrollo de cada red. Mantener este preprocesamiento es necesario durante la inferencia, ya que el modelo fue ajustado considerando una representación específica de sus entradas.

2.2.1 Operaciones convolucionales

Una capa convolucional aplica un conjunto de filtros sobre regiones locales del tensor de entrada. Cada filtro contiene pesos aprendidos previamente y genera un mapa de características que responde a determinados patrones presentes en la imagen. De manera simplificada, una componente de salida puede expresarse como

$$y_{i,j,k} = b_k + \sum_c \sum_u \sum_v w_{u,v,c,k} x_{i+u,j+v,c}, \quad (2.1)$$

donde x representa el tensor de entrada, w los pesos del filtro, b_k el término de sesgo asociado al canal de salida y $y_{i,j,k}$ el valor resultante en una posición del mapa de características.

La utilización de los mismos pesos del filtro en distintas posiciones permite detectar un mismo patrón independientemente de su ubicación dentro de la imagen. Al apilar múltiples capas convolucionales, las representaciones obtenidas pueden evolucionar desde características locales relativamente simples hacia patrones de mayor nivel utilizados posteriormente para realizar la clasificación [9].

Después de una convolución suele aplicarse una función de activación no lineal. Una de las más utilizadas es ReLU, definida como

$$\text{ReLU}(x) = \max(0, x). \quad (2.2)$$

La incorporación de funciones no lineales permite que una red represente relaciones más complejas que las obtenibles mediante una sucesión exclusivamente lineal de operaciones.

2.2.2 Reducción espacial y clasificación

Las arquitecturas convolucionales pueden incorporar operaciones de reducción espacial o *pooling*. Estas operaciones disminuyen las dimensiones de los mapas de características conservando información representativa de una región. Por ejemplo, *max pooling* conserva el mayor valor dentro de cada región considerada. La reducción espacial disminuye la cantidad de datos que debe procesarse en las capas posteriores.

Una vez extraídas las características relevantes, la etapa final de la red produce los valores utilizados para determinar la clase de la imagen. Dependiendo de la arquitectura, esta etapa puede utilizar capas completamente conectadas u otras operaciones destinadas a transformar las características obtenidas en una salida de clasificación.

Los valores generados antes de convertir la salida a probabilidades se denominan *logits*. En un problema binario, una red puede generar un único valor y transformarlo mediante una función sigmoide,

$$\sigma(z) = \frac{1}{1 + e^{-z}}, \quad (2.3)$$

obteniendo un valor comprendido entre 0 y 1. Si este valor se interpreta como la probabilidad de la clase positiva, la probabilidad de la clase negativa puede expresarse como $1 - \sigma(z)$.

Otra posibilidad consiste en generar un logit por clase y utilizar una función softmax. Para una salida con K clases, la probabilidad asociada a la clase k se obtiene mediante

$$p_k = \frac{e^{z_k}}{\sum_{j=1}^K e^{z_j}}, \quad (2.4)$$

donde z_k corresponde al logit de la clase k . La suma de las probabilidades resultantes es igual a uno y la clase predicha se determina normalmente a partir de la componente de mayor valor.

En este trabajo se consideran ambas formas de salida. Algunos de los modelos evaluados producen una salida sigmoide escalar, mientras que otros generan dos valores asociados a las clases “nowildfire” y “wildfire”. Durante la evaluación, estas representaciones se expresan de manera común para permitir una comparación consistente entre modelos.

2.2.3 Representación de los tensores

Además de sus valores numéricos, un tensor de entrada se caracteriza por el orden en que se organizan sus dimensiones. Para un conjunto de imágenes, dos disposiciones habituales son NHWC y NCHW. En NHWC las dimensiones corresponden al tamaño del lote, altura, ancho y canales, mientras que en NCHW corresponden al tamaño del lote, canales, altura y ancho.

Estas disposiciones contienen la misma clase de información, pero no son intercambiables sin reorganizar explícitamente las dimensiones del tensor. Por esta razón, durante el despliegue debe mantenerse el formato esperado por cada modelo. En la metodología experimental se conserva la disposición original de cada arquitectura durante las etapas de preprocesamiento, exportación y ejecución acelerada.

La Figura 2.1 sintetiza el flujo general de inferencia descrito en esta sección, desde la preparación de una imagen de entrada hasta la obtención de la clase predicha. El esquema no representa una arquitectura específica, sino las etapas generales de preprocesamiento, extracción de características y generación de la salida de clasificación presentes en los modelos considerados.

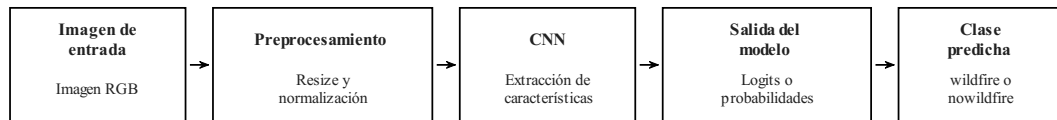


Figura 2.1: Esquema general del proceso de inferencia de una red neuronal convolucional para clasificación de imágenes.

2.3 Arquitecturas consideradas

Aunque las arquitecturas evaluadas en este trabajo pertenecen a la familia de las redes convolucionales, no todas organizan sus capas de la misma forma. Estas diferencias afectan la manera en que se extraen y reutilizan las características de una imagen y, como consecuencia, pueden producir requerimientos computacionales distintos durante la inferencia.

En el benchmark inicial se consideraron dos redes convolucionales personalizadas, MobileNetV2, DenseNet121 y dos implementaciones de ResNet50. A continuación se describen las principales ideas que caracterizan a estas arquitecturas.

2.3.1 Redes convolucionales personalizadas

Una CNN personalizada corresponde a una red cuya estructura fue definida específicamente para una tarea determinada, en lugar de utilizar directamente una arquitectura estandarizada como MobileNet o ResNet. Su diseño puede combinar distintas cantidades de capas convolucionales, operaciones de reducción espacial y capas de clasificación.

Esta libertad de diseño permite construir redes muy diferentes entre sí. Una CNN puede ser pequeña y utilizar pocos parámetros, mientras que otra puede incorporar un mayor número de filtros o capas completamente conectadas y alcanzar un tamaño considerablemente superior.

En este trabajo se evaluaron dos CNN provenientes de desarrollos previos diferentes. Ambas realizan la misma tarea de clasificación binaria, pero presentan estructuras y cantidades de parámetros distintas. Por esta razón, se consideran modelos independientes durante el benchmark.

2.3.2 MobileNetV2

MobileNetV2 fue diseñada específicamente para reducir el costo computacional de las redes convolucionales y facilitar su utilización en dispositivos con recursos limitados [10]. Para lograrlo, reemplaza parte de las convoluciones convencionales por operaciones más eficientes.

En una convolución convencional, los filtros procesan simultáneamente la información espacial de la imagen y la información proveniente de sus distintos canales. MobileNetV2 separa parte de este trabajo en dos pasos. Primero procesa cada canal de manera independiente mediante una convolución *depthwise*. Posteriormente utiliza convoluciones de 1×1 para combinar la información obtenida entre los diferentes canales.

Esta separación permite reducir considerablemente la cantidad de operaciones necesarias respecto de una convolución convencional equivalente.

MobileNetV2 organiza además estas operaciones en bloques que modifican temporalmente la cantidad de canales. De manera simplificada, el bloque primero expande la representación de entrada, luego aplica la convolución *depthwise* y finalmente vuelve a reducir la cantidad de canales. Cuando la forma de entrada y salida lo permite, también se incorpora una conexión directa entre ambas.

La Figura 2.2 representa de forma simplificada esta secuencia de operaciones. La expansión aumenta temporalmente la cantidad de canales, la convolución *depthwise* realiza el procesamiento espacial de cada canal y la proyección reduce nuevamente la representación antes de producir la salida del bloque.

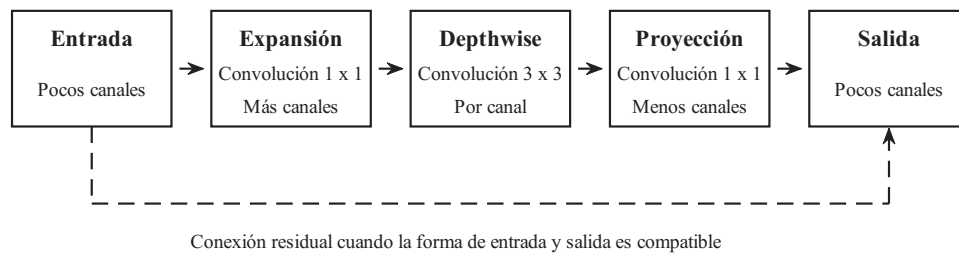


Figura 2.2: Representación simplificada de un bloque utilizado en MobileNetV2.

El objetivo de esta estructura es mantener una capacidad adecuada para extraer características de las imágenes reduciendo el número de operaciones necesarias. Esta propiedad explica el interés de MobileNetV2 en aplicaciones donde la capacidad de procesamiento y la energía disponible son limitadas.

2.3.3 DenseNet121

DenseNet adopta una estrategia diferente para reutilizar la información generada dentro de la red. En una CNN convencional, cada capa recibe principalmente la salida de la capa inmediatamente anterior. En DenseNet, en cambio, las capas dentro de un mismo bloque reciben también las características generadas por varias de las capas anteriores [11].

De manera intuitiva, esto significa que una característica detectada en una etapa temprana de la red puede continuar disponible para capas posteriores sin necesidad de volver a generarla. Las nuevas características se van agregando a las ya existentes a medida que la información avanza por el bloque.

DenseNet121 organiza este mecanismo mediante varios bloques densos separados por etapas que reducen el tamaño de las representaciones. La reutilización de características puede ser beneficiosa

para el procesamiento de la información, pero durante la inferencia también implica manipular numerosos mapas de características intermedios.

Por esta razón, una arquitectura como DenseNet puede presentar un costo de ejecución elevado aun cuando su cantidad de parámetros no sea la mayor entre los modelos comparados.

2.3.4 ResNet50

ResNet introduce conexiones residuales que permiten que la información atraviese directamente determinados grupos de capas [12]. En lugar de obligar a que toda la información sea transformada sucesivamente por cada operación, una parte de la entrada puede avanzar por un camino directo y combinarse posteriormente con el resultado obtenido por las capas intermedias.

De forma simplificada, si un conjunto de capas transforma una entrada x mediante una función $F(x)$, la salida del bloque puede expresarse como

$$y = F(x) + x. \quad (2.5)$$

El término x corresponde al camino directo o conexión residual.

La Figura 2.3 ilustra este mecanismo de forma simplificada. La entrada x se procesa mediante el bloque convolucional $F(x)$, mientras que la conexión residual permite transportar directamente la información de entrada hasta el punto donde ambas representaciones se combinan.

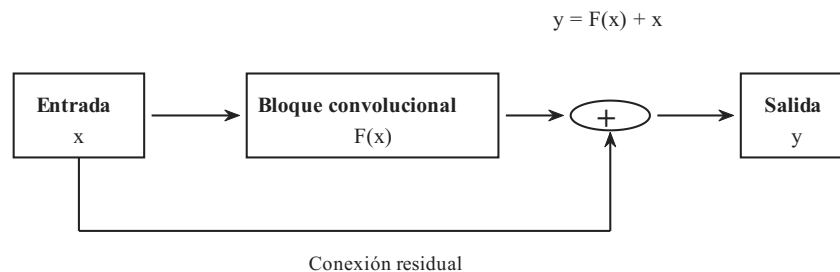


Figura 2.3: Representación simplificada de una conexión residual.

Esta estructura facilita la construcción de redes con una gran cantidad de capas. ResNet50 utiliza repetidamente estos bloques a lo largo de la arquitectura, dando lugar a un modelo considerablemente más profundo y de mayor tamaño que arquitecturas específicamente orientadas a reducir el costo de inferencia, como MobileNetV2.

En el benchmark de este trabajo se evaluaron dos implementaciones distintas de ResNet50. Aunque ambas pertenecen a la misma familia arquitectónica, poseen pesos, procedimientos de

preprocesamiento y formatos de ejecución diferentes. Por este motivo, sus resultados se consideran de manera independiente y no como una comparación directa entre frameworks.

2.4 Métricas de evaluación

La comparación de modelos considera tanto su desempeño predictivo como los recursos necesarios para producir las predicciones. En este trabajo se utilizan métricas derivadas de la matriz de confusión y métricas de latencia, throughput, potencia y energía. Estas métricas se describen a continuación.

2.4.1 Matriz de confusión y desempeño predictivo

En la clasificación binaria utilizada, “wildfire” corresponde a la clase positiva y “nowildfire” a la negativa. TP y TN representan clasificaciones correctas, mientras que FP representa una muestra negativa clasificada como positiva y FN una muestra positiva clasificada como negativa. La Figura 2.4 resume estas posibilidades y permite identificar visualmente los dos tipos de error.

		Clase predicha	
		nowildfire	wildfire
Clase real	nowildfire	TN Clasificación correcta	FP Error
	wildfire	FN Error	TP Clasificación correcta

Figura 2.4: Representación conceptual de la matriz de confusión para la clasificación binaria utilizada en este trabajo.

A partir de estos valores se calculan

$$\text{Exactitud} = \frac{TP + TN}{TP + TN + FP + FN}, \quad (2.6)$$

$$\text{Precisión} = \frac{TP}{TP + FP}, \quad (2.7)$$

$$\text{Sensibilidad} = \frac{TP}{TP + FN}, \quad (2.8)$$

y

$$F_1 = \frac{2TP}{2TP + FP + FN}. \quad (2.9)$$

La exactitud resume la proporción total de aciertos, la precisión caracteriza la confiabilidad de las predicciones positivas y la sensibilidad (*recall*) indica la fracción de muestras positivas identificadas correctamente. En esta aplicación, la sensibilidad resulta especialmente relevante porque disminuye al aumentar los falsos negativos. La puntuación F1 combina precisión y sensibilidad mediante su media armónica. Por ello, la comparación conserva también la matriz de confusión y no depende de una única métrica.

2.4.2 Latencia y tasa de procesamiento

Latencia corresponde al tiempo requerido para completar una inferencia dentro de una frontera de medición determinada. Esta frontera puede incluir solo la ejecución del modelo o también lectura, preprocesamiento, transferencias de memoria y postprocesamiento. En consecuencia, dos valores son directamente comparables solo cuando representan etapas equivalentes.

El throughput expresa la cantidad de inferencias completadas durante un intervalo,

$$R = \frac{N_{\text{inf}}}{\Delta t}, \quad (2.10)$$

donde N_{inf} es el número de inferencias completadas. Aunque ambas métricas se relacionan, un throughput medido bajo carga sostenida no equivale necesariamente al inverso de la latencia obtenida mediante inferencias aisladas.

2.4.3 Potencia y energía

La potencia eléctrica puede variar durante la ejecución debido a cambios de carga, frecuencia y mecanismos de gestión del hardware. La energía consumida durante un intervalo se obtiene integrando la potencia en el tiempo,

$$E = \int_{t_0}^{t_1} P(t) dt. \quad (2.11)$$

Para comparar configuraciones puede expresarse además la energía por inferencia,

$$E_{\text{inf}} = \frac{E_{\text{carga}}}{N_{\text{inf}}}. \quad (2.12)$$

Las campañas distinguen energía bruta, asociada al consumo total durante la ejecución, e incremental, que utiliza el reposo como referencia para estimar el aumento debido a la carga. Potencia y energía deben interpretarse junto con throughput, pues una configuración de mayor potencia puede completar más inferencias durante el mismo intervalo.

2.5 Optimización y despliegue de inferencia

Una vez entrenado un modelo, su ejecución puede optimizarse sin modificar la tarea aprendida. En este trabajo intervienen una representación intermedia común, un runtime de inferencia acelerada y precisiones numéricas reducidas.

2.5.1 ONNX como representación intermedia

TensorFlow y PyTorch utilizan estructuras internas diferentes para representar una red. ONNX proporciona una descripción común mediante un grafo computacional, donde los nodos representan operaciones y las conexiones el flujo de tensores. Esto permite separar parcialmente el framework de desarrollo del runtime de inferencia. Sin embargo, una exportación debe validarse porque la compatibilidad depende de las operaciones y del opset implementado por cada runtime.

2.5.2 TensorRT y engines

TensorRT optimiza y ejecuta redes neuronales sobre GPU NVIDIA. A partir de una representación del modelo puede combinar operaciones compatibles, seleccionar implementaciones y emplear precisiones reducidas cuando el hardware lo permite. El resultado es un engine que incorpora decisiones específicas de plataforma y versión de TensorRT, por lo que no debe considerarse un archivo universalmente transferible.

La Figura 2.5 resume la diferencia entre ONNX como representación intermedia y el engine como implementación optimizada para el hardware de destino.

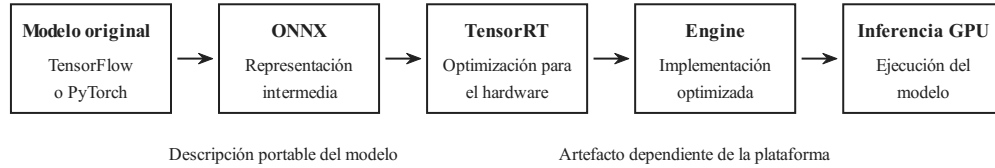


Figura 2.5: Flujo conceptual de conversión y despliegue mediante ONNX y TensorRT.

2.5.3 Precisión numérica

FP32 y FP16 representan valores en punto flotante con 32 y 16 bits, mientras que INT8 utiliza enteros de 8 bits. Reducir la precisión puede disminuir requerimientos de memoria y aprovechar unidades de cálculo especializadas, pero también introduce diferencias numéricas que deben verificarse frente a la referencia. Habilitar FP16 en TensorRT no implica necesariamente que todas las operaciones se ejecuten exclusivamente en esa precisión, pues el runtime puede conservar otras representaciones cuando resulte necesario.

2.5.4 Cuantización INT8

La cuantización aproxima valores representados en punto flotante mediante un conjunto reducido de niveles, permitiendo utilizar formatos como INT8 para disminuir los requerimientos de memoria y aprovechar unidades de cómputo especializadas [13].

Existen dos estrategias principales para incorporar cuantización en una red neuronal. En *Quantization-Aware Training* (QAT), los efectos de la cuantización se simulan durante el entrenamiento, de modo que los pesos del modelo puedan adaptarse a las diferencias numéricas introducidas por la menor precisión. En cambio, *Post-Training Quantization* (PTQ) se aplica sobre un modelo previamente entrenado, sin realizar un ajuste adicional de los pesos mediante entrenamiento. En este último caso, un conjunto representativo de datos puede utilizarse para estimar los rangos y parámetros de cuantización requeridos por las activaciones.

En esta memoria se utiliza PTQ, ya que los modelos evaluados fueron obtenidos previamente entrenados y no se realizó una etapa adicional de entrenamiento. Para las variantes INT8 se empleó un conjunto de calibración destinado a estimar los parámetros necesarios para representar las activaciones en precisión reducida.

En ONNX, la cuantización puede representarse explícitamente mediante nodos Q/DQ. Q transforma valores hacia la representación cuantizada y DQ los devuelve a punto flotante cuando una operación posterior lo requiere. La Figura 2.6 ilustra este mecanismo.

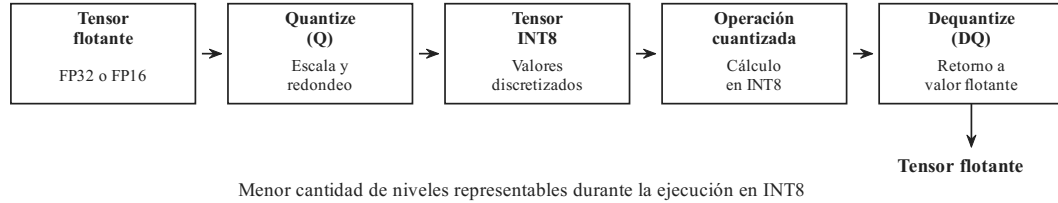


Figura 2.6: Representación conceptual del uso de nodos Q/DQ en un modelo cuantizado.

Una variante INT8 no garantiza automáticamente una mejora útil. Su viabilidad depende de la arquitectura, el runtime y el hardware, por lo que debe verificarse tanto numéricamente como mediante las métricas predictivas correspondientes.

2.5.5 Ejecución sobre GPU

Las GPU permiten ejecutar numerosas operaciones en paralelo y resultan adecuadas para las operaciones repetitivas de las redes neuronales. En plataformas NVIDIA, CUDA proporciona el entorno utilizado por TensorRT. Cuando host y device poseen espacios de memoria diferenciados, una inferencia puede incluir transferencias H2D y D2H además de la ejecución del engine. Por ello, el tiempo exclusivo del engine puede ser menor que la latencia observada desde la aplicación.

2.5.6 Condiciones de ejecución

El desempeño medido también depende de las condiciones de ejecución. El batch size determina cuántas entradas se procesan conjuntamente y el warm-up permite excluir inferencias iniciales afectadas por inicializaciones del entorno de ejecución. Además, los mecanismos de escalado dinámico de voltaje y frecuencia (DVFS) ajustan durante la ejecución las frecuencias y voltajes de operación del hardware en función de factores como la carga computacional, la temperatura y las restricciones de potencia. En plataformas Jetson, los perfiles de potencia condicionan este comportamiento y su valor nominal no equivale necesariamente a la potencia instantánea consumida.

2.6 Estado del arte

La incorporación de inteligencia artificial en plataformas satelitales ha avanzado desde propuestas conceptuales hasta demostraciones experimentales con distintos aceleradores. Paralelamente, el monitoreo de incendios forestales ha motivado el desarrollo de modelos de visión por computador orientados a reducir el tiempo entre la adquisición de datos y la generación de información útil. En

esta sección se revisan los trabajos más relacionados con ambas líneas y se establece su relación con la evaluación desarrollada en esta memoria.

2.6.1 Inferencia de redes neuronales a bordo

Furano et al. analizan las oportunidades y dificultades asociadas a trasladar algoritmos de inteligencia artificial hacia sistemas espaciales, destacando que el procesamiento cercano al sensor puede disminuir la dependencia de la transmisión de datos hacia Tierra. Al mismo tiempo, identifican restricciones asociadas al consumo energético, la capacidad computacional y las condiciones particulares del entorno espacial [1].

Más recientemente, Diana y Dini revisan las alternativas de hardware y software disponibles para ejecutar redes neuronales a bordo de satélites. Su análisis considera aceleradores comerciales y otras arquitecturas especializadas, junto con técnicas destinadas a disminuir los requerimientos computacionales de los modelos. Los autores destacan que estas decisiones adquieren especial importancia en misiones de bajo presupuesto y plataformas CubeSat, donde la potencia y la capacidad de cómputo pueden limitar la complejidad de las redes ejecutables [2].

Estos trabajos muestran que el problema del procesamiento a bordo no puede reducirse únicamente a seleccionar una arquitectura con buen desempeño predictivo. El modelo, las técnicas de optimización y el hardware de ejecución forman parte de un mismo problema de despliegue y deben considerarse conjuntamente.

2.6.2 Inteligencia artificial aplicada a incendios en sistemas restringidos

Dentro de las aplicaciones relacionadas con incendios forestales, James et al. desarrollan un sistema de clasificación basado en imágenes satelitales que distingue entre imágenes con y sin presencia de fuego. El trabajo evalúa variantes de MobileNet y selecciona una configuración basada en MobileNetV2, considerando explícitamente la reducción de los recursos requeridos para facilitar su utilización en aplicaciones de inteligencia artificial embebida [5]. Este antecedente resulta especialmente cercano a la tarea de clasificación binaria estudiada en la presente memoria, aunque el objetivo de este trabajo es comparar distintas arquitecturas y configuraciones de ejecución en lugar de desarrollar un nuevo clasificador.

Tsalera et al. estudian la identificación de incendios en entornos con capacidad computacional restringida utilizando arquitecturas como SqueezeNet, ShuffleNet, MobileNetV2 y ResNet50. Su análisis muestra la necesidad de considerar conjuntamente desempeño predictivo y complejidad

computacional al seleccionar una red para aplicaciones con recursos limitados [14]. La inclusión de MobileNetV2 y ResNet50 en dicho estudio también evidencia que arquitecturas con objetivos de diseño diferentes pueden ser utilizadas para una misma tarea, pero presentar compromisos distintos durante su implementación.

Una aproximación más directamente relacionada con el procesamiento satelital es presentada por Thangavel et al., quienes estudian la detección autónoma de incendios a partir de imágenes hiperespectrales y redes neuronales. Los autores evalúan la implementación de su modelo sobre distintos aceleradores, entre ellos Intel Movidius NCS-2, NVIDIA Jetson Nano y NVIDIA Jetson TX2, con el propósito de analizar su factibilidad para procesamiento a bordo [15]. Este trabajo constituye un antecedente relevante porque vincula explícitamente imágenes obtenidas mediante observación terrestre, inferencia neuronal y hardware embebido orientado a una futura operación satelital.

Zheng et al. abordan la detección de llamas y humo mediante una arquitectura basada en MobileNetV3 y YOLOv4. El modelo fue posteriormente desplegado sobre una NVIDIA Jetson Xavier NX y acelerado mediante TensorRT, alcanzando un throughput cercano a 26 imágenes por segundo [16]. Aunque se trata de una tarea de detección de objetos y no de la clasificación binaria utilizada en esta memoria, el trabajo demuestra la utilización conjunta de una arquitectura ligera, TensorRT y la misma familia de plataforma embebida considerada experimentalmente en este estudio.

También se han desarrollado propuestas específicamente orientadas al uso de inteligencia artificial en plataformas CubeSat. Cratere et al. estudian el procesamiento de incendios directamente a bordo mediante redes convolucionales ligeras y aceleración por hardware, planteando una solución de segmentación de incendios bajo las restricciones de tamaño, masa, potencia y comunicaciones propias de este tipo de plataforma [17]. Este antecedente muestra que el procesamiento local de información relacionada con incendios constituye una línea de investigación activa y que la eficiencia computacional forma parte central del diseño de estas soluciones.

2.6.3 Posicionamiento del presente trabajo

Los antecedentes revisados abordan distintos componentes del problema. Algunos trabajos estudian las tecnologías necesarias para ejecutar redes neuronales a bordo de satélites [1, 2], mientras que otros desarrollan modelos específicos para clasificación, detección o segmentación de incendios sobre imágenes satelitales o sistemas embebidos [5, 14, 15, 16, 17]. Sin embargo, las diferencias de tarea, dataset, arquitectura y plataforma dificultan utilizar sus resultados para determinar directamente qué

configuración ofrece el mejor compromiso para el escenario estudiado en esta memoria.

El presente trabajo adopta una perspectiva complementaria. En lugar de proponer una nueva arquitectura, se evalúan modelos existentes bajo una metodología común y se estudia cómo cambia su comportamiento al modificar la plataforma y la precisión numérica. La comparación incorpora desempeño predictivo, latencia, throughput y consumo energético, permitiendo observar conjuntamente los beneficios y costos asociados a las distintas alternativas de inferencia.

El escenario de aplicación se vincula además con el trabajo de Silva, quien desarrolla un concepto de misión CubeSat 12U con procesamiento a bordo orientado a incendios forestales y establece restricciones operacionales y energéticas para dicha capacidad [7]. La presente memoria utiliza este antecedente como contexto de aplicación, pero se concentra específicamente en la evaluación experimental, optimización y despliegue de modelos de inferencia sobre hardware real.

De esta forma, el aporte del estudio se sitúa en la caracterización experimental de los compromisos que surgen al trasladar modelos de clasificación hacia condiciones de ejecución más restringidas. Esta perspectiva permite complementar los antecedentes existentes con mediciones obtenidas bajo un protocolo común y establecer criterios cuantitativos para la selección de una configuración según las prioridades de la aplicación.

Capítulo 3

Metodología experimental

3.1 Diseño experimental y flujo general

La metodología experimental se diseñó para comparar modelos de inferencia bajo condiciones controladas y estudiar posteriormente el efecto de su optimización y despliegue en distintas plataformas. El procedimiento avanzó desde una referencia común en CPU hacia la conversión a ONNX, la construcción de engines TensorRT y la caracterización en una GPU discreta y una plataforma embebida. En todas las etapas se conservaron el preprocesamiento y la interpretación de salida de cada modelo.

En una primera etapa se evaluaron seis modelos de clasificación binaria en CPU y se seleccionaron tres arquitecturas representativas de distintos compromisos entre desempeño predictivo y requerimientos computacionales. Los modelos seleccionados se exportaron a ONNX y se verificó su correspondencia con las implementaciones originales antes de construir variantes TensorRT en FP32, FP16 y, cuando fue técnicamente viable, INT8. Las siete variantes funcionales se caracterizaron en una RTX 3060 utilizada como plataforma de desarrollo y posteriormente se desplegaron en una Jetson Xavier NX, donde los engines se reconstruyeron para el entorno embebido y se evaluaron bajo diferentes perfiles de potencia.

Los resultados de cada etapa se consolidaron manteniendo separadas las métricas predictivas de las fronteras temporales y energéticas propias de cada plataforma. La Figura 3.1 resume el flujo general seguido.

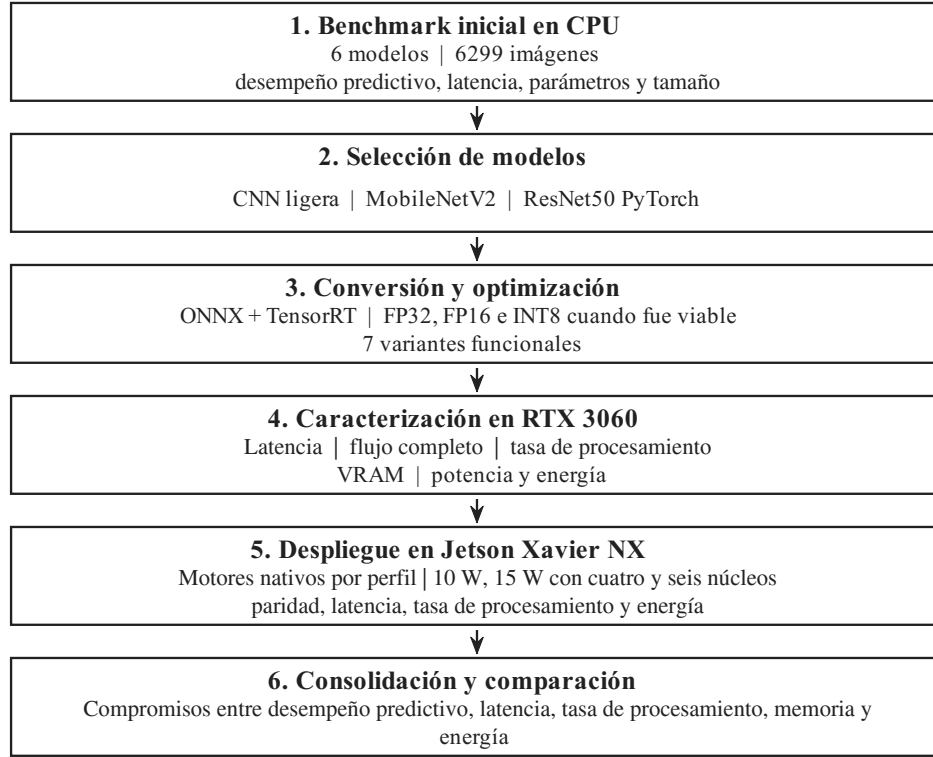


Figura 3.1: Flujo general de la metodología experimental desarrollada en este trabajo.

3.2 Dataset y preparación de entradas

3.2.1 Dataset

Para la evaluación experimental se utilizó el dataset público *Wildfire Prediction Dataset*, disponible en Kaggle [18]. Este dataset contiene imágenes RGB en formato JPEG, originalmente de 350×350 píxeles, organizadas en las clases “wildfire” y “nowildfire”. El dataset se encuentra dividido en train, validation y test splits. En este trabajo, la evaluación comparativa definitiva del desempeño predictivo se realizó utilizando el test split oficial.

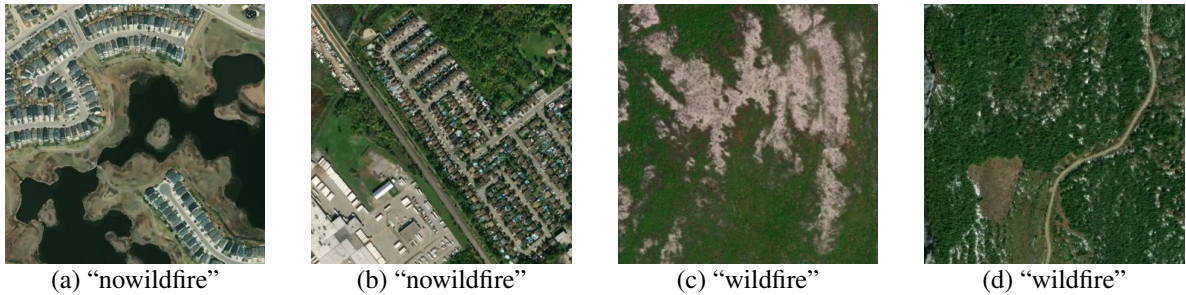


Figura 3.2: Ejemplos de imágenes pertenecientes a las clases “nowildfire” y “wildfire” del dataset utilizado en la evaluación.

La clase “wildfire” se consideró como clase positiva durante el cálculo de las métricas de clasificación. Inicialmente se identificaron 6300 archivos en el test split. Durante la validación de las imágenes se detectó un archivo JPEG truncado perteneciente a la clase “wildfire”, el cual fue excluido de manera uniforme en todas las evaluaciones. Como resultado, el test split definitivo quedó compuesto por 6299 imágenes válidas, cuya distribución se presenta en la Tabla 3.1.

Tabla 3.1: Distribución de imágenes del test split utilizado en la evaluación.

Clase	Archivos originales	Excluidos	Válidos
nowildfire	2820	0	2820
wildfire	3480	1	3479
Total	6300	1	6299

Los train y validation splits no se utilizaron para calcular las métricas reportadas sobre el test split. El train split se empleó posteriormente para obtener los datos de calibración requeridos por la cuantización INT8, mientras que el validation split se utilizó para la selección previa de configuraciones cuantizadas. El test split se empleó tanto en el benchmark inicial de los seis modelos, que orientó la selección de las tres arquitecturas que continuaron a optimización, como en la comparación definitiva de las variantes funcionales. En ningún caso las imágenes de prueba fueron utilizadas para calibrar o seleccionar configuraciones INT8. En consecuencia, la comparación final describe el desempeño de las arquitecturas seleccionadas sobre este conjunto, pero no constituye una estimación completamente independiente del procedimiento de selección inicial.

3.2.2 Preparación de entradas para las campañas de inferencia

Las diferentes arquitecturas evaluadas no comparten necesariamente el mismo preprocesamiento ni la misma disposición de dimensiones de entrada. Por este motivo, cada imagen fue transformada de acuerdo con el procedimiento correspondiente al modelo original, el cual se describe en la Sección 3.3. Esta decisión permitió comparar las implementaciones sin modificar las condiciones bajo las cuales cada modelo había sido diseñado.

Para la campaña experimental en Jetson Xavier NX se preparó previamente un subconjunto estático y balanceado de 200 imágenes del test split. Se seleccionaron las primeras 100 imágenes válidas de cada clase según el orden fijado durante la evaluación del conjunto de prueba y posteriormente se intercalaron, alternando entre “nowildfire” y “wildfire”. Sobre esta selección se aplicó el preprocesamiento específico de cada arquitectura y se almacenaron los tensores resultantes,

junto con salidas de referencia obtenidas mediante los modelos ONNX previamente validados. las mismas imágenes de origen y, dentro de cada arquitectura, los mismos tensores de entrada entre variantes y perfiles de ejecución, reduciendo fuentes de variabilidad ajenas al runtime de inferencia.

A partir de esta selección se utilizaron 32 casos para las verificaciones iniciales de paridad, 64 tensores para las pruebas de carga sostenida y el conjunto completo de 200 entradas para las mediciones de latencia. En los casos donde una verificación de precisión reducida requirió una comparación más extensa, la paridad fue ampliada hasta las 200 entradas disponibles. Los tensores fueron almacenados en formato NumPy y conservaron la organización requerida por cada modelo: NHWC para la CNN ligera y MobileNetV2, y NCHW para ResNet50.

El uso de tensores precalculados en estas mediciones permitió aislar el comportamiento del runtime de inferencia respecto de la lectura y transformación de las imágenes. Por esta razón, las campañas basadas en tensores no deben interpretarse como mediciones del pipeline completo desde un archivo JPEG hasta la obtención de una predicción. Las fronteras temporales específicas empleadas en cada plataforma se definen posteriormente en las secciones correspondientes.

3.3 Modelos evaluados y preprocesamiento

3.3.1 Modelos evaluados

El benchmark inicial consideró seis modelos previamente entrenados para clasificación binaria entre las clases “wildfire” y “nowildfire”. Los modelos fueron obtenidos a partir de implementaciones públicas desarrolladas por distintos autores y se utilizaron con sus pesos originales, sin realizar etapas adicionales de entrenamiento durante este trabajo.

La CNN ligera, almacenada en el archivo `final_wildfire_model.keras`, fue obtenida a partir del notebook publicado por el usuario `humankey` en Kaggle [19]. La implementación de ResNet50 en PyTorch, correspondiente a `resnet50_wildfire_20.pth`, proviene del notebook publicado por `pratyushrao` [20]. Por otra parte, los modelos MobileNetV2, DenseNet121, ResNet50 Keras y la CNN de mayor tamaño, almacenados respectivamente en los archivos `mobilenet.h5`, `densenet.h5`, `resnet.h5` y `cnn_model.h5`, fueron obtenidos de una implementación publicada por `alesengj` [21].

Los modelos considerados fueron, por tanto, una CNN ligera personalizada, MobileNetV2, DenseNet121, una implementación de ResNet50 en Keras, una CNN personalizada de mayor tamaño y una segunda implementación de ResNet50 en PyTorch. Sus principales características y las métricas

de desempeño predictivo obtenidas durante el benchmark inicial se resumen en la Tabla 3.2.

Tabla 3.2: Modelos considerados en el benchmark inicial.

Modelo	Parámetros	Tamaño (MiB)	Exactitud (%)	Precision (%)	Recall (%)	F1-score (%)
CNN ligera	470 274	5,44	97,02	97,03	97,59	97,31
MobileNetV2	2 586 177	12,72	97,27	98,22	96,81	97,51
DenseNet121	7 300 161	30,68	97,29	97,89	97,18	97,53
ResNet50 Keras	24 112 513	96,37	91,95	93,35	91,98	92,66
CNN de mayor tamaño	25 784 769	295,16	97,13	99,28	95,49	97,35
ResNet50 PyTorch	23 512 130	90,00	97,62	98,40	97,27	97,83

La presencia de dos implementaciones de ResNet50 responde a que los modelos provenían de desarrollos previos distintos y conservaban tanto sus pesos como sus procedimientos originales de inferencia. De manera similar, las dos redes convolucionales personalizadas presentan diferencias importantes en cantidad de parámetros y estructura. Por este motivo, cada archivo se trató como un modelo independiente durante el benchmark.

Para mantener la fidelidad respecto de las implementaciones originales, no se impuso un preprocesamiento único para los seis modelos. En su lugar, cada arquitectura conservó la transformación de entrada y la interpretación de salida asociadas a su implementación de origen. Esta decisión resulta relevante, ya que modificar el preprocesamiento puede alterar directamente las predicciones de una red aun cuando sus pesos permanezcan sin cambios.

3.3.2 Preprocesamiento y representación de las salidas

A pesar de que todos los modelos reciben imágenes RGB redimensionadas a 224×224 píxeles, se identificaron tres procedimientos de preprocesamiento distintos. Estas diferencias se mantuvieron durante todas las etapas posteriores para asegurar que las comparaciones entre las implementaciones originales, ONNX y TensorRT utilizaran entradas equivalentes.

CNN ligera

La CNN ligera recibe imágenes RGB redimensionadas a 224×224 píxeles. Los valores de los píxeles se convierten a punto flotante de 32 bits y se mantienen en el intervalo entre 0 y 255. Los tensores utilizan disposición NHWC, correspondiente al orden lote, altura, ancho y canales.

No se aplica una división externa por 255, ya que el propio modelo incorpora una capa de reescalamiento que realiza esta normalización como parte de la red. Aplicar nuevamente esta

operación antes de la inferencia modificaría el rango esperado por el modelo.

La salida corresponde a un vector de dos probabilidades obtenido mediante una función softmax. El orden de las clases es:

$$[P(\text{nowildfire}), P(\text{wildfire})]. \quad (3.1)$$

La clase predicha corresponde a la componente de mayor probabilidad.

Modelos Keras con preprocesamiento CLAHE

MobileNetV2, DenseNet121, ResNet50 Keras y la CNN de mayor tamaño utilizan el mismo procedimiento de preprocesamiento. Primero, la imagen se decodifica en RGB y se redimensiona a 224×224 píxeles. A continuación, los valores se convierten a enteros de 8 bits y la imagen se transforma desde el espacio de color RGB al espacio CIELAB.

Sobre el canal de luminancia se aplica ecualización adaptativa de histograma con contraste limitado (CLAHE), utilizando un límite de contraste de 2,5 y una cuadrícula de 8×8 regiones. Una vez realizada esta operación, la imagen se transforma nuevamente a RGB, se convierte a punto flotante de 32 bits y sus valores se normalizan al intervalo entre 0 y 1 mediante una división por 255. La disposición final de los tensores es NHWC.

Los cuatro modelos generan una salida sigmoide escalar. Si se denota esta salida como s , su valor se interpreta como la probabilidad asociada a la clase “wildfire”. Para mantener una representación común de dos clases durante los procedimientos de evaluación, la salida se expresa como

$$[P(\text{nowildfire}), P(\text{wildfire})] = [1 - s, s]. \quad (3.2)$$

De manera equivalente, valores de s superiores a 0,5 corresponden a una predicción de la clase “wildfire”.

ResNet50 PyTorch

La implementación de ResNet50 en PyTorch utiliza un procedimiento diferente. La imagen se carga en formato RGB y se redimensiona a 224×224 píxeles mediante interpolación bilineal. Posteriormente, los valores se convierten al intervalo entre 0 y 1 y se normaliza cada canal utilizando los parámetros asociados a ImageNet:

$$\mu = [0.485, 0.456, 0.406], \quad (3.3)$$

$$\sigma = [0.229, 0.224, 0.225]. \quad (3.4)$$

Para cada canal c , la transformación aplicada puede expresarse como

$$x'_c = \frac{x_c - \mu_c}{\sigma_c}. \quad (3.5)$$

En este caso, los tensores utilizan disposición NCHW, correspondiente al orden lote, canales, altura y ancho. Esta organización difiere de la utilizada por los modelos Keras y se conservó durante las etapas posteriores de exportación y despliegue.

La red produce dos valores de salida en forma de logits, ordenados como “nowildfire” y “wildfire”. Para obtener probabilidades comparables con las demás arquitecturas, se aplica una función softmax sobre ambos valores. Durante la implementación se utilizó una forma numéricamente estable de esta operación, restando previamente el valor máximo de los logits antes de calcular las exponenciales.

En consecuencia, los tres procedimientos de preprocesamiento se mantuvieron separados durante todo el desarrollo experimental. Las diferencias de formato, normalización y representación de salida fueron tratadas explícitamente en los scripts de evaluación para evitar que una transformación propia de una arquitectura fuese aplicada accidentalmente a otra.

3.4 Benchmark inicial en CPU

El benchmark inicial tuvo como objetivo establecer una referencia común para comparar los seis modelos antes de aplicar optimizaciones orientadas a GPU. Para reducir diferencias atribuibles al runtime, todos los modelos fueron evaluados bajo restricciones homogéneas de recursos y utilizando el mismo conjunto de imágenes válidas. Cada modelo conservó el preprocesamiento descrito en la Sección 3.3.

3.4.1 Entorno y condiciones de ejecución

Las evaluaciones se realizaron sobre un computador equipado con un procesador Intel Core i5-12450H. Para establecer condiciones homogéneas entre los modelos se utilizó un entorno Docker basado en Python 3.9, restringido a 4 vCPU y 4 GiB de memoria RAM. El número de hilos

internos utilizados por los frameworks se limitó a cuatro y cada modelo fue ejecutado en un proceso independiente. De esta forma, se evitó que el estado de una evaluación anterior afectara la memoria o los recursos disponibles para el modelo siguiente.

El entorno de software se construyó a partir de la imagen base `python:3.9-slim` y utilizó versiones fijadas de las principales dependencias. En particular, se emplearon TensorFlow CPU 2.20.0, Keras 3.10.0, PyTorch 2.8.0, torchvision 0.23.0, NumPy 2.0.2 y OpenCV 5.0.0.93. La fijación de estas versiones permitió mantener un entorno reproducible para las seis evaluaciones.

La configuración utilizada se resume en la Tabla 3.3.

Tabla 3.3: Condiciones utilizadas en el benchmark inicial en CPU.

Parámetro	Configuración
Procesador host	Intel Core i5-12450H
Imagen base	<code>python:3.9-slim</code>
Python	3.9
TensorFlow CPU / Keras	2.20.0 / 3.10.0
PyTorch / torchvision	2.8.0 / 0.23.0
Runtime	Docker
CPU disponible	4 vCPU
Memoria disponible	4 GiB
Hilos internos	4
Evaluación de desempeño predictivo	Batch size 16
Medición de latencia	Batch size 1
Iteraciones de warm-up	10
Muestras de latencia	200
Clase positiva	wildfire
Imágenes válidas	6299

Inicialmente se evaluó la posibilidad de utilizar un batch size de 32 imágenes para la evaluación completa. Sin embargo, DenseNet121 agotó la memoria disponible bajo esta configuración. Para mantener una condición común entre los seis modelos, se adoptó un batch size de 16 imágenes para todas las evaluaciones posteriores. Esta decisión evitó utilizar valores de batch size diferentes según la arquitectura y mantuvo constante la restricción de memoria del entorno experimental.

3.4.2 Evaluación de desempeño predictivo

El desempeño predictivo de cada modelo se evaluó sobre las 6299 imágenes válidas del test split. Todos los modelos utilizaron el mismo conjunto de archivos y mantuvieron un orden de procesamiento

común. Las inferencias se realizaron utilizando un batch size de 16 imágenes hasta completar el conjunto.

La clase “wildfire” fue definida como clase positiva. A partir de las predicciones obtenidas se construyó la matriz de confusión y se calcularon accuracy, precision, recall y F1-score. También se registraron explícitamente los falsos negativos, debido a su relevancia dentro de la aplicación estudiada. Las definiciones de estas métricas se presentan en el Capítulo 2, mientras que los criterios utilizados para su comparación se detallan posteriormente en la Sección 3.9.

Además de las métricas de clasificación, para cada modelo se registraron la cantidad de parámetros y el tamaño del archivo original. Estas características se utilizaron junto con el desempeño predictivo y latencia para realizar la selección de candidatos que continuarían a la etapa de optimización.

3.4.3 Medición de latencia

La medición de latencia se realizó de forma independiente de la evaluación completa del test split. Se utilizó batch size 1, ya que el objetivo era caracterizar el tiempo requerido para procesar una imagen individual y disponer de una referencia común para comparar los seis modelos. Para mantener un procedimiento determinista, se utilizaron las primeras 200 imágenes válidas según el orden fijado para el conjunto de prueba.

Antes de comenzar las mediciones se preparó la primera entrada y se realizaron 10 inferencias de warm-up sobre ella. Estas ejecuciones se utilizaron únicamente para estabilizar la ejecución inicial del modelo y no fueron consideradas en los resultados. Posteriormente, cada una de las 200 imágenes se procesó de manera individual y se registraron por separado el tiempo de preprocesamiento y el tiempo de inferencia. Para cada etapa se tomó una referencia de tiempo inmediatamente antes de comenzar su ejecución y otra al finalizar. La diferencia entre ambas correspondió al tiempo registrado para esa muestra. Los intervalos se midieron mediante `time.perf_counter_ns()`, y las diferencias expresadas en nanosegundos se dividieron por 10^6 para obtener milisegundos.

En el caso del preprocesamiento, la primera referencia de tiempo se tomó antes de leer la imagen y la segunda una vez obtenido el tensor de entrada requerido por el modelo. Para la inferencia, la primera referencia se tomó con el tensor ya preparado, justo antes de ejecutar el modelo, y la segunda después de obtener e interpretar su salida para determinar la clase predicha. De esta forma, la latencia de inferencia reportada no incluye la lectura, decodificación ni las transformaciones aplicadas a la imagen durante el preprocesamiento.

En Keras se ejecutó `model(batch, training=False)` y la salida se convirtió a un

arreglo NumPy antes de determinar la clase predicha. En PyTorch se ejecutó el modelo dentro de `torch.inference_mode()`, se aplicó la función softmax y se obtuvieron las clases y probabilidades como arreglos NumPy. La preparación del lote se realizó antes de iniciar la medición de inferencia.

A partir de las 200 mediciones individuales se calcularon la mediana y el percentil 95. La mediana de latencia de inferencia se utilizó como valor principal para comparar los modelos, mientras que el tiempo de preprocesamiento se mantuvo como una medición independiente. Esta separación evita atribuir al costo de inferencia diferencias provenientes del preprocesamiento particular de cada arquitectura.

Cuando se presenta un throughput derivado de este benchmark, este se calcula a partir de la mediana de latencia de inferencia y corresponde únicamente a una estimación basada en mediciones individuales. No representa una medición de throughput sostenido bajo ejecución continua, la cual se evaluó posteriormente mediante campañas de carga específicas en las plataformas aceleradas.

3.5 Selección de modelos para optimización

A partir del benchmark inicial se seleccionaron tres modelos para continuar con las etapas de conversión, optimización y despliegue acelerado. La selección no se realizó únicamente a partir del valor de accuracy obtenido sobre el test split. Se consideraron de manera conjunta el desempeño predictivo, el recall frente a la clase “wildfire”, latencia de inferencia, la cantidad de parámetros y el tamaño del modelo.

La CNN ligera correspondiente a `final_wildfire_model.keras` fue seleccionada como representante de una arquitectura compacta. Entre los seis modelos evaluados presentó la menor cantidad de parámetros y el menor tamaño de archivo, además de un alto desempeño predictivo y el mayor recall para la clase “wildfire”. Estas características la convirtieron en un candidato de interés para estudiar hasta qué punto una arquitectura pequeña podía beneficiarse de técnicas adicionales de optimización.

MobileNetV2 fue incorporada como una alternativa intermedia y como representante de una familia de arquitecturas diseñada específicamente para reducir el costo computacional de redes convolucionales. Aunque otros modelos del benchmark alcanzaron resultados predictivos similares, MobileNetV2 presentaba una cantidad de parámetros y un tamaño considerablemente menores que arquitecturas como DenseNet121 y ResNet50 Keras. Su inclusión permitió evaluar el comportamiento

de una arquitectura ligera ampliamente orientada a escenarios de inferencia con recursos limitados.

Finalmente, la implementación de ResNet50 en PyTorch fue seleccionada como referencia de mayor complejidad. Este modelo presentó el mayor desempeño predictivo global dentro del benchmark inicial, aunque con una cantidad de parámetros y un tamaño considerablemente superiores a los de las dos alternativas ligeras. Su incorporación permitió estudiar si el mayor costo computacional asociado a una arquitectura de esta escala podía compensarse mediante optimización y aceleración en hardware especializado.

Los otros tres modelos no continuaron a la etapa de optimización. DenseNet121 presentó un costo de inferencia y una demanda de memoria elevados en relación con las alternativas seleccionadas. ResNet50 Keras mostró un desempeño predictivo inferior al resto de los principales candidatos. Por otra parte, la CNN de mayor tamaño alcanzó un valor bajo de latencia de inferencia en CPU, pero presentó el mayor tamaño de archivo del conjunto y un menor recall frente a la clase “wildfire” que la CNN ligera.

De esta forma, la selección final quedó compuesta por tres modelos que representan diferentes compromisos dentro del espacio evaluado, como se resume en la Tabla 3.4. La selección no establece todavía una alternativa óptima para el escenario de aplicación. Su propósito fue conservar modelos con características suficientemente diferentes para estudiar posteriormente el efecto de las técnicas de optimización y del hardware de ejecución.

Tabla 3.4: Modelos seleccionados para las etapas de optimización y despliegue.

Modelo	Archivo original	Rol en la comparación
CNN ligera	<code>final_wildfire_model.keras</code>	Arquitectura compacta
MobileNetV2	<code>mobilenet.h5</code>	Arquitectura orientada a eficiencia
ResNet50 PyTorch	<code>resnet50_wildfire_20.pth</code>	Referencia de mayor complejidad

3.6 Conversión, validación y optimización con ONNX y TensorRT

Los tres modelos seleccionados provenían de frameworks diferentes y no podían ser utilizados directamente bajo una única interfaz de inferencia acelerada. Por este motivo, se adoptó ONNX como representación intermedia y TensorRT como runtime de optimización y ejecución sobre GPU NVIDIA. El procedimiento se diseñó de forma progresiva, verificando primero la correspondencia entre el modelo original y su representación ONNX antes de construir variantes con distintas precisiones numéricas.

3.6.1 Exportación a ONNX y validación

Cada modelo fue exportado a una representación ONNX en precisión FP32 conservando su forma de entrada, disposición de dimensiones y salida original. Las tres arquitecturas seleccionadas se exportaron utilizando opset 18. Los archivos obtenidos fueron sometidos a las verificaciones de consistencia del formato ONNX antes de continuar con la construcción de engines de TensorRT.

La correspondencia numérica se verificó utilizando ONNX Runtime. Para ello, se procesó un subconjunto común de entradas con el framework original y con la representación ONNX, comparando las salidas numéricas y las clases obtenidas. Esta etapa permitió comprobar que el proceso de exportación no había alterado de forma significativa el comportamiento de los modelos.

La comparación se realizó respetando la forma de salida descrita en la Sección 3.3. En la CNN ligera, el modelo entrega directamente dos probabilidades mediante una salida softmax, correspondientes a las clases “nowildfire” y “wildfire”. En MobileNetV2, en cambio, la salida corresponde a un único valor obtenido mediante una función sigmoide. Denotando este valor como s , donde s representa la probabilidad asignada a la clase “wildfire”, la salida se transformó a la representación binaria $[1 - s, s]$, correspondiente a las probabilidades de las clases “nowildfire” y “wildfire”, respectivamente. Finalmente, ResNet50 entrega dos logits, es decir, dos valores de salida que todavía no representan probabilidades. Por este motivo, se aplicó una función softmax para transformarlos en dos probabilidades asociadas a las clases “nowildfire” y “wildfire”. De esta forma, las salidas de los tres modelos pudieron compararse utilizando una representación común de dos probabilidades.

Solo después de verificar la correspondencia entre el modelo original y ONNX se continuó con la generación de engines de TensorRT. De esta forma, las diferencias observadas en etapas posteriores podían analizarse respecto de una referencia previamente validada y no atribuirse directamente al proceso de exportación.

3.6.2 Construcción de variantes TensorRT

A partir de los modelos ONNX validados se construyeron engines de TensorRT. La primera variante generada para cada arquitectura utilizó precisión FP32 y se empleó como referencia para comprobar el correcto funcionamiento del despliegue acelerado. Las predicciones obtenidas fueron comparadas con las referencias correspondientes antes de continuar con precisiones reducidas.

Durante la construcción de los engines FP32 y FP16 destinados a la RTX 3060 se deshabilitó

explícitamente TensorFlow-32 mediante la opción `--noTF32`. En particular, esta decisión evitó que las operaciones elegibles de la variante de referencia FP32 utilizaran implícitamente TF32, manteniendo diferenciada esta configuración respecto de las optimizaciones posteriores en FP16 e INT8.

Posteriormente se habilitó el uso de FP16 durante la construcción de los engines. Esta configuración permite a TensorRT seleccionar operaciones en menor precisión cuando la plataforma y las implementaciones disponibles lo permiten. La interfaz de entrada y salida se mantuvo de acuerdo con el contrato definido para cada modelo. Por este motivo, el uso de FP16 se interpreta como una optimización habilitada dentro del engine y no como una garantía de que todas las operaciones de la red se ejecuten exclusivamente en dicha precisión.

Las variantes FP32 y FP16 se construyeron satisfactoriamente para las tres arquitecturas seleccionadas. Cada engine fue sometido a verificaciones de inferencia y correspondencia de clases antes de ser incorporado a las campañas posteriores.

3.6.3 Cuantización INT8

La cuantización INT8 se abordó mediante cuantización posterior al entrenamiento. Para ello, se utilizaron representaciones ONNX con nodos explícitos de cuantización y de-cuantización, comúnmente denominados Q/DQ. Los datos empleados para la calibración fueron tomados del train split y se utilizó un subconjunto balanceado de 256 imágenes, compuesto por 128 casos de cada clase. La selección se realizó de forma pseudoaleatoria sin reemplazo utilizando una semilla fija igual a 2026, de manera que el conjunto de calibración pudiera ser reproducido. El test split no fue utilizado durante esta etapa.

Se evaluaron métodos de calibración MinMax, Entropy y Percentile, según la viabilidad de cada arquitectura. Las variantes candidatas fueron inspeccionadas y evaluadas previamente sobre datos de validación antes de considerar su utilización sobre el test split. Este procedimiento permitió evitar que una configuración de cuantización no funcional fuese incorporada directamente a la evaluación definitiva.

En la CNN ligera, los tres métodos de calibración fueron comparados sobre el validation split antes de seleccionar la configuración que continuaría hacia la evaluación definitiva. El método Percentile con percentil 99,999 fue seleccionado debido a que presentó un mayor recall para la clase “wildfire” y un menor número de falsos negativos que MinMax y Entropy, además de producir una menor cantidad de cambios de clasificación respecto de la referencia FP32. Aunque Entropy obtuvo valores ligeramente

superiores de accuracy y F1-score, se priorizó la reducción de falsos negativos debido a la relevancia de esta métrica para la aplicación estudiada.

La cuantización no produjo una variante funcional para todas las arquitecturas. En MobileNetV2, los modelos ONNX cuantizados conservaron capacidad discriminativa al ser ejecutados con ONNX Runtime. Sin embargo, los engines de TensorRT contruidos a partir de estas representaciones produjeron salidas prácticamente constantes y perdieron la capacidad de discriminar la clase “wildfire”. Por este motivo, MobileNetV2 INT8 fue descartada antes de la evaluación definitiva sobre el test split y no se realizaron mediciones de latencia para esta variante.

En ResNet50 tampoco se obtuvo un engine INT8 ejecutable. La variante basada en MinMax pudo ser cuantizada a ONNX, pero TensorRT no encontró una implementación compatible para uno de los primeros bloques de operaciones cuantizadas durante la construcción del engine. Los intentos con Entropy y Percentile finalizaron bajo una presión de memoria elevada, incluso al reducir el batch size utilizado durante el proceso. En consecuencia, ResNet50 INT8 fue excluida de las campañas posteriores.

Estos casos se conservaron como parte de la trazabilidad experimental. La ausencia de una variante INT8 no se interpretó como un resultado faltante, sino como una limitación observada durante el proceso de optimización bajo la combinación de modelos, herramientas y hardware utilizada.

3.6.4 Variantes resultantes

Como resultado del procedimiento de conversión y optimización se obtuvieron siete variantes funcionales que continuaron hacia las campañas de caracterización. Estas configuraciones se presentan en la Tabla 3.5.

Tabla 3.5: Variantes TensorRT utilizadas en las campañas experimentales posteriores.

Modelo	FP32	FP16	INT8
CNN ligera	Sí	Sí	Sí
MobileNetV2	Sí	Sí	No
ResNet50	Sí	Sí	No

De esta manera, las campañas posteriores se realizaron sobre CNN FP32, CNN FP16, CNN INT8, MobileNetV2 FP32, MobileNetV2 FP16, ResNet50 FP32 y ResNet50 FP16. Las variantes descartadas no fueron utilizadas para completar artificialmente las comparaciones ni se les asignaron métricas de desempeño que no hubiesen sido medidas.

3.7 Metodología experimental en RTX 3060

Las siete variantes se caracterizaron sobre una NVIDIA RTX 3060 Laptop utilizada como plataforma de desarrollo. Esta campaña midió latencia, pipeline completo, throughput, VRAM, potencia y energía bajo condiciones homogéneas. Los resultados describen una GPU discreta y no una simulación de la plataforma embebida.

3.7.1 Plataforma y condiciones de ejecución

La campaña se ejecutó en un ASUS TUF Dash F15 FX517ZM con RTX 3060 Laptop de 6 GiB, sistema CachyOS/Arch Linux y NVIDIA Container Toolkit. La imagen Docker `memoria-gpu-tensorrt:26.01-opencv` incorporó TensorRT 10.14.1.48 y las dependencias requeridas por los tres procedimientos de preprocesamiento. La configuración se resume en la Tabla 3.6.

Tabla 3.6: Plataforma utilizada en la campaña experimental RTX 3060.

Componente	Configuración
Equipo	ASUS TUF Dash F15 FX517ZM
GPU	NVIDIA GeForce RTX 3060 Laptop GPU
Memoria gráfica	6 GiB
Driver NVIDIA	610.43.03
Sistema anfitrión	CachyOS / Arch Linux
TensorRT	10.14.1.48
Perfil TensorRT min/opt/max	1/1/16
Modo de GPU	Hybrid
Perfil de plataforma	balanced
Preferencia energética CPU	balance_performance

Los engines utilizaron perfil min/opt/max 1/1/16, permitiendo batch size 16 para evaluación predictiva y batch size 1 para mediciones individuales. Antes de cada repetición el equipo permaneció conectado a alimentación y sin otros procesos CUDA relevantes. Se verificó estado P8, temperatura no superior a 50 °C, utilización de GPU igual a cero y ausencia de una pantalla conectada directamente a la GPU discreta.

3.7.2 Medición de latencia y flujo de procesamiento completo

La latencia se midió con batch size 1. Cada repetición comenzó con 10 inferencias de warm-up y continuó con 200 muestras. Se registraron tres fronteras principales: engine CUDA, correspondiente

solo a la ejecución del engine; CUDA total, que incorpora H2D, engine y D2H; y host NumPy a predicción, desde el tensor de entrada en RAM hasta el índice de clase final, incluyendo preparación de buffers, transferencias, sincronización y postprocesamiento. Esta última fue la medida exterior principal de inferencia para la comparación transversal.

También se midió de forma independiente el pipeline completo desde archivo JPEG hasta clase predicha, nuevamente con 10 warm-ups y 200 muestras. El tiempo se descompuso en lectura/decodificación, preprocesamiento e inferencia NumPy a predicción. Esta medición permitió distinguir una reducción del tiempo de engine de la aceleración obtenida sobre el procesamiento extremo a extremo.

3.7.3 Carga sostenida, telemetría y energía

La campaña de carga sostenida utilizó 64 tensores reales precargados, de modo que las ventanas monitorizadas no incluyeran lectura ni preprocesamiento. Cada repetición se dividió en las cuatro fases de la Tabla 3.7.

Tabla 3.7: Protocolo utilizado para las mediciones de carga sostenida en RTX 3060.

Etapa	Duración
Estabilización en reposo	60 s
Reposo medido	30 s
Estabilización bajo carga	60 s
Carga medida	180 s

Durante las pruebas se registraron inferencias completadas, throughput, VRAM, potencia y temperatura. La telemetría se obtuvo mediante `nvidia-smi` cada 200 ms. El throughput se calculó a partir del número efectivo de inferencias completadas durante la ventana medida, según la definición del Capítulo 2.

La potencia de la GPU se obtuvo de `power.draw`. A partir de sus muestras discretas, la energía se aproximó mediante regla trapezoidal,

$$E \approx \sum_{i=1}^{n-1} \frac{P_i + P_{i+1}}{2} (t_{i+1} - t_i). \quad (3.6)$$

Se calculó energía bruta por inferencia y energía incremental utilizando como referencia la potencia de reposo de la misma repetición. Estas mediciones corresponden exclusivamente a la GPU y no al consumo total del portátil.

Cada una de las siete variantes se sometió a tres repeticiones independientes de latencia, pipeline y carga sostenida. Los archivos se verificaron antes de consolidar cada métrica mediante la mediana de las tres repeticiones.

3.8 Despliegue y metodología en Jetson Xavier NX

La segunda plataforma acelerada fue una NVIDIA Jetson Xavier NX Developer Kit. A diferencia de la RTX 3060, integra CPU, GPU y memoria en un sistema embebido y dispone de perfiles configurables de potencia. Los engines de la RTX no se reutilizaron, porque dependen del hardware y de TensorRT, por lo que se reconstruyeron localmente a partir de los ONNX validados.

3.8.1 Plataforma, entradas y compatibilidad

La plataforma utilizó Ubuntu 20.04, JetPack 5.1.2, L4T R35.4.1, Python 3.8.10, CUDA 11.4, TensorRT 8.5.2.2 y aproximadamente 6,7 GiB de RAM. La configuración se presenta en la Tabla 3.8. Durante las campañas, `jetson_clocks` permaneció desactivado y DVFS se mantuvo activo, por lo que la plataforma pudo ajustar dinámicamente sus frecuencias de operación. La Jetson Xavier NX dispone además de un Deep Learning Accelerator (DLA), correspondiente a hardware dedicado a la ejecución de redes neuronales. Este acelerador no fue configurado durante los experimentos, por lo que todas las variantes evaluadas con TensorRT se ejecutaron sobre la GPU.

Tabla 3.8: Plataforma utilizada en la campaña experimental Jetson Xavier NX.

Componente	Configuración
Plataforma	NVIDIA Jetson Xavier NX Developer Kit
Sistema operativo	Ubuntu 20.04
JetPack	5.1.2
L4T	R35.4.1
Arquitectura	aarch64
Python	3.8.10
CUDA	11.4
TensorRT	8.5.2.2
NumPy	1.17.4
Memoria RAM disponible	~6,7 GiB
Batch size	1
Perfil TensorRT min/opt/max	1/1/1
DLA	No utilizado

La campaña utilizó los 200 tensores precalculados descritos en la Subsección 3.2.2, junto con las referencias ONNX y manifiestos de integridad. Esto permitió usar exactamente las mismas imágenes de origen y, dentro de cada arquitectura, los mismos tensores de entrada entre variantes y perfiles, pero implica que las métricas de Xavier NX caracterizan el runtime y no incluyen lectura, decodificación ni preprocesamiento.

Durante la construcción inicial de CNN FP32 se detectó una incompatibilidad entre el ONNX canónico opset 18 y el parser de TensorRT 8.5.2 en una operación `ReduceMean` usada para representar Global Average Pooling. Los ejes [1,2] correspondían a las dimensiones espaciales NHWC, pero TensorRT infería incorrectamente la salida antes de la capa densa. Se generó una copia compatible en opset 16 que mantuvo la misma operación matemática y trasladó los ejes a un atributo de `ReduceMean`, sin modificar pesos, arquitectura, preprocesamiento ni salidas. ONNX Runtime confirmó equivalencia exacta, con diferencias absoluta máxima y media iguales a cero. La misma adaptación se aplicó a CNN INT8 conservando Q/DQ.

3.8.2 Perfiles de potencia y construcción de engines

Se evaluaron tres configuraciones de `nvpmodel`: 10 W, 15 W con cuatro núcleos y 15 W con seis núcleos. Las dos últimas se incorporaron para contextualizar las mediciones respecto del margen energético del escenario CubeSat 12U descrito por Silva [7]. La Tabla 3.9 presenta las configuraciones. La potencia nominal del perfil no se utilizó como sustituto de la potencia realmente consumida.

Tabla 3.9: Perfiles de potencia utilizados en la Jetson Xavier NX.

Condición	Perfil <code>nvpmodel</code>	ID	CPU activas
10 W	MODE_10W_DESKTOP	5	4 núcleos
15 W, 4 núcleos	MODE_15W_4CORE	1	4 núcleos
15 W, 6 núcleos	MODE_15W_6CORE	2	6 núcleos

Los siete engines se reconstruyeron bajo cada una de las tres condiciones porque reutilizar un plan generado con otro perfil producía advertencias de compatibilidad. Antes de cada campaña se verificaron carga, inferencia e identidad de los artefactos. Por ello, las diferencias entre los perfiles de cuatro y seis núcleos representan el despliegue TensorRT optimizado bajo cada condición y no pueden atribuirse exclusivamente a los dos núcleos adicionales, pues la reconstrucción también puede modificar las tácticas seleccionadas. Una campaña exploratoria que reutilizó engines de cuatro núcleos bajo el perfil de seis núcleos se excluyó de los resultados oficiales.

3.8.3 Validación de paridad

Antes de las campañas computacionales se verificó la correspondencia de cada engine con sus referencias, comprobando forma de salida, ausencia de valores no finitos, diferencias absoluta máxima y media, concordancia numérica y clase predicha. Se utilizaron tolerancias $atol = 10^{-5}$ y $rtol = 10^{-4}$ para FP32, $atol = 10^{-3}$ y $rtol = 10^{-2}$ para FP16, y $atol = 10^{-2}$ y $rtol = 5 \times 10^{-2}$ para INT8.

La validación inicial se realizó sobre las primeras 32 entradas del subconjunto estático de 200 imágenes preparado para la campaña en Jetson Xavier NX. Este conjunto estaba balanceado e intercalado por clase, por lo que las 32 entradas utilizadas correspondieron a 16 imágenes “nowildfire” y 16 imágenes “wildfire”. Cuando una variante no cumplió el criterio de concordancia numérica pero mantuvo las decisiones de clasificación, la comparación se extendió a las 200 entradas disponibles para comprobar si las diferencias numéricas se traducían en cambios de clase.

3.8.4 Medición de latencia

El benchmark utilizó batch size 1, 50 inferencias de warm-up y 200 muestras por variante y perfil. El warm-up más prolongado que en RTX buscó reducir la influencia de transitorios iniciales y del ajuste de frecuencias con DVFS activo. La métrica principal fue el tiempo exclusivo del engine medido con eventos CUDA. También se registraron H2D, D2H, CUDA total y NumPy a predicción como fronteras complementarias.

Cada procedimiento se repitió tres veces. Para cada repetición se obtuvo la mediana de 200 muestras y el resultado consolidado correspondió a la mediana de las tres medianas, sin eliminar repeticiones por presentar valores altos o bajos.

3.8.5 Carga sostenida, telemetría y energía

La caracterización sostenida reutilizó el protocolo temporal de la RTX 3060: 60 s de estabilización en reposo, 30 s de reposo medido, 60 s de estabilización bajo carga y 180 s de carga medida. La telemetría se obtuvo mediante `tegrastats` cada 200 ms, registrando memoria, temperaturas y rails de potencia. Para caracterizar la potencia total de entrada del módulo se utilizó exclusivamente `VDD_IN`; los rails internos no se sumaron entre sí.

Durante cada ventana de carga se contabilizaron las inferencias completadas para obtener throughput. La energía se estimó integrando `VDD_IN` mediante la Ecuación 3.6. Se conservaron

energía bruta e incremental por inferencia, utilizando para esta última el reposo de la misma repetición como referencia. La comparación principal entre perfiles utiliza energía incremental complementada con la potencia efectiva de entrada.

Cada combinación de variante y perfil tuvo tres repeticiones de latencia y tres de carga sostenida. El orden de las variantes se modificó entre repeticiones para reducir sesgos asociados a la posición dentro de la campaña. Antes de consolidar mediante la mediana se verificaron las series esperadas, cantidad de muestras, identidad de artefactos y disponibilidad de VDD_IN.

3.9 Criterios de consolidación y comparación de resultados

La evaluación final mantiene separadas cuatro dimensiones: desempeño predictivo, paridad del despliegue, latencia/throughput y utilización de recursos/energía. Las métricas predictivas consideran “wildfire” como clase positiva y provienen de la evaluación de las 6299 imágenes válidas. Además de accuracy, precision, recall y F1-score se registra el número de falsos negativos, y las variantes FP16 e INT8 se comparan respecto de FP32 mediante cambios de clase.

La paridad tiene un propósito diferente: compara dos implementaciones del mismo modelo sobre entradas comunes. En la Xavier NX se verifican las salidas de los engines frente a referencias ONNX sobre subconjuntos deterministas, pero accuracy, precision, recall y F1-score continúan asociados a la evaluación completa realizada previamente. Una variante puede conservar las decisiones de clasificación aunque sus salidas numéricas no sean idénticas.

Las comparaciones temporales respetan la frontera de cada campaña. En CPU se distinguen preprocesamiento e inferencia del framework. En RTX se separan engine, CUDA total, NumPy a predicción y pipeline desde archivo. En Xavier NX, la comparación principal entre perfiles utiliza el tiempo exclusivo del engine. Del mismo modo, el throughput sostenido de RTX y Xavier se mantiene diferenciado del valor derivado de latencia CPU.

Las métricas de recursos también dependen de la plataforma. En RTX se utiliza VRAM y la potencia corresponde exclusivamente a la GPU discreta obtenida mediante `nvidia-smi`. En Xavier NX, CPU y GPU comparten memoria y la potencia se obtiene desde VDD_IN, por lo que no se realizan comparaciones directas de memoria o potencia absoluta entre ambas plataformas. En las campañas sostenidas se prioriza la energía incremental por inferencia respecto del reposo de la misma repetición.

Finalmente, la comparación no utiliza una única métrica para seleccionar una variante. Los

resultados se analizan conjuntamente considerando desempeño predictivo, falsos negativos, latencia, throughput, memoria y energía. FP32 actúa como referencia dentro de cada arquitectura y los perfiles Xavier se comparan mediante sus resultados consolidados, teniendo presente que los engines fueron reconstruidos bajo cada condición.

Capítulo 4

Resultados y discusión

En este capítulo se presentan y analizan los resultados obtenidos durante las distintas etapas experimentales. En primer lugar, se examina el benchmark inicial de los seis modelos ejecutados en CPU, a partir del cual se identificaron los candidatos que continuaron hacia la etapa de optimización. Posteriormente se estudia el efecto de las distintas precisiones numéricas sobre el desempeño predictivo y se presentan las campañas realizadas en la RTX 3060 y en la Jetson Xavier NX. Finalmente, los resultados se analizan de forma conjunta para identificar los principales compromisos entre desempeño predictivo y desempeño computacional dentro del escenario de aplicación considerado.

4.1 Resultados del benchmark inicial en CPU

El benchmark inicial permitió observar diferencias relevantes entre los seis modelos tanto en desempeño predictivo como en costo computacional. Si bien cinco de las seis alternativas alcanzaron valores de accuracy superiores al 97 %, los resultados muestran que un desempeño similar en términos de clasificación no implica necesariamente requerimientos computacionales comparables. Por este motivo, la selección de modelos para las etapas posteriores consideró simultáneamente la capacidad de clasificación, el recall frente a la clase “wildfire”, latencia y las características de cada modelo.

4.1.1 Desempeño predictivo

La Tabla 4.1 presenta las principales métricas de clasificación obtenidas sobre las 6299 imágenes válidas del test split. La clase “wildfire” se mantiene como clase positiva.

ResNet50 PyTorch obtuvo el mayor accuracy y el mayor F1-score del conjunto, alcanzando

Tabla 4.1: Resultados de desempeño predictivo y matriz de confusión del benchmark inicial en CPU.

Modelo	Accuracy (%)	Recall (%)	F1-score (%)	TN	FP	FN	TP
CNN ligera	97,02	97,59	97,31	2716	104	84	3395
MobileNetV2	97,27	96,81	97,51	2759	61	111	3368
DenseNet121	97,29	97,18	97,53	2747	73	98	3381
ResNet50 Keras	91,95	91,98	92,66	2592	228	279	3200
CNN mayor	97,13	95,49	97,35	2796	24	157	3322
ResNet50 PyTorch	97,62	97,27	97,83	2765	55	95	3384

97,62 % y 97,83 %, respectivamente. Sin embargo, la CNN ligera presentó el mayor recall para la clase “wildfire”, con 97,59 %, y el menor número de falsos negativos, con 84 casos. Esta diferencia resulta relevante porque muestra que el modelo con mayor accuracy global no coincide necesariamente con aquel que identifica una mayor proporción de las muestras pertenecientes a la clase positiva.

MobileNetV2 y DenseNet121 presentaron resultados de clasificación muy similares entre sí. DenseNet121 alcanzó un valor de accuracy ligeramente superior, mientras que MobileNetV2 obtuvo un recall algo menor. En ambos casos, las diferencias de desempeño predictivo respecto de ResNet50 PyTorch y de la CNN ligera fueron reducidas en términos absolutos.

La CNN de mayor tamaño también superó el 97 % de accuracy, pero presentó un recall de 95,49 %, correspondiente a 157 falsos negativos. Por otra parte, ResNet50 Keras mostró el desempeño predictivo más bajo del conjunto, con un accuracy de 91,95 % y 279 falsos negativos. Estos resultados evidencian diferencias importantes entre implementaciones que, aun perteneciendo a familias arquitectónicas similares, conservan pesos, preprocesamiento y procedimientos de inferencia distintos.

La Figura 4.1 permite observar visualmente que las diferencias entre los cinco modelos con mejor desempeño global son relativamente pequeñas en términos de accuracy, pero no necesariamente en recall. Este comportamiento es particularmente visible en la CNN de mayor tamaño, cuya accuracy permanece por sobre el 97 %, mientras que su recall frente a “wildfire” disminuye respecto de las alternativas seleccionadas. ResNet50 Keras se separa con mayor claridad del resto de los modelos en ambas métricas.

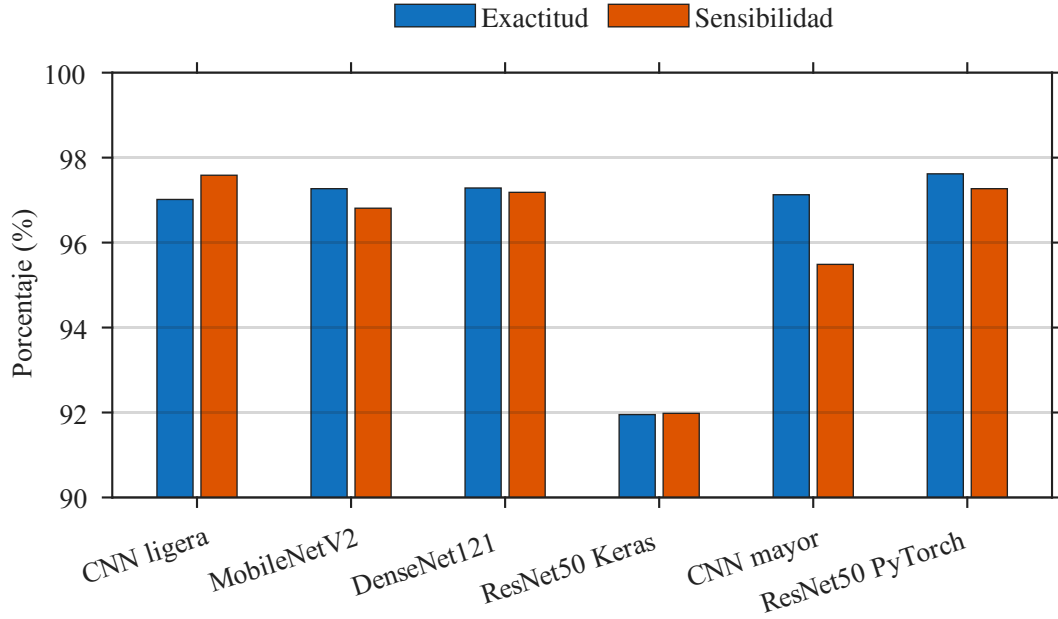


Figura 4.1: Comparación de exactitud y sensibilidad para la clase “wildfire” en el benchmark inicial en CPU.

4.1.2 Latencia y complejidad de los modelos

Las diferencias entre los modelos se vuelven más pronunciadas al considerar latencia de inferencia y sus características estructurales. La Tabla 4.2 presenta la mediana de la latencia obtenida con batch size 1, junto con la cantidad de parámetros y el tamaño del archivo original de cada modelo.

Tabla 4.2: Latencia y características de los modelos evaluados en CPU.

Modelo	latencia (ms)	Parámetros	Archivo (MiB)
CNN ligera	37,73	470 274	5,44
MobileNetV2	106,68	2 586 177	12,72
DenseNet121	350,71	7 300 161	30,68
ResNet50 Keras	218,52	24 112 513	96,37
CNN mayor	27,13	25 784 769	295,16
ResNet50 PyTorch	46,66	23 512 130	90,00

El menor valor de latencia fue obtenido por la CNN de mayor tamaño, con una mediana de 27,13 ms. Este resultado muestra que la cantidad de parámetros o el tamaño del archivo no determinan por sí solos el tiempo de inferencia. El valor de latencia efectivo también depende de la secuencia y del tipo de operaciones que componen la red, de las dimensiones de los tensores intermedios y de las implementaciones utilizadas por el framework sobre el hardware disponible.

La CNN ligera alcanzó el segundo menor valor de latencia, con 37,73 ms, utilizando solo 470 274 parámetros y un archivo de 5,44 MiB. En comparación, ResNet50 PyTorch presentó un valor de latencia de 46,66 ms y aproximadamente 23,5 millones de parámetros. De esta forma, ambos modelos alcanzaron tiempos de inferencia del mismo orden de magnitud, aunque con diferencias sustanciales en cantidad de parámetros y tamaño del archivo.

MobileNetV2 presentó una latencia de 106,68 ms, superior a las latencias de las dos CNN y de ResNet50 PyTorch a pesar de su menor cantidad de parámetros. Este comportamiento refuerza la idea de que el costo efectivo de inferencia depende no solamente del número de parámetros, sino también de las operaciones que conforman la red y de su ejecución por parte del framework.

DenseNet121 presentó el mayor valor de latencia del benchmark, con 350,71 ms, mientras que ResNet50 Keras alcanzó 218,52 ms. En el caso de DenseNet121, este costo computacional se suma al agotamiento de memoria observado durante la prueba preliminar con batch size 32, lo que motivó la utilización de batch size 16 de forma homogénea para los seis modelos.

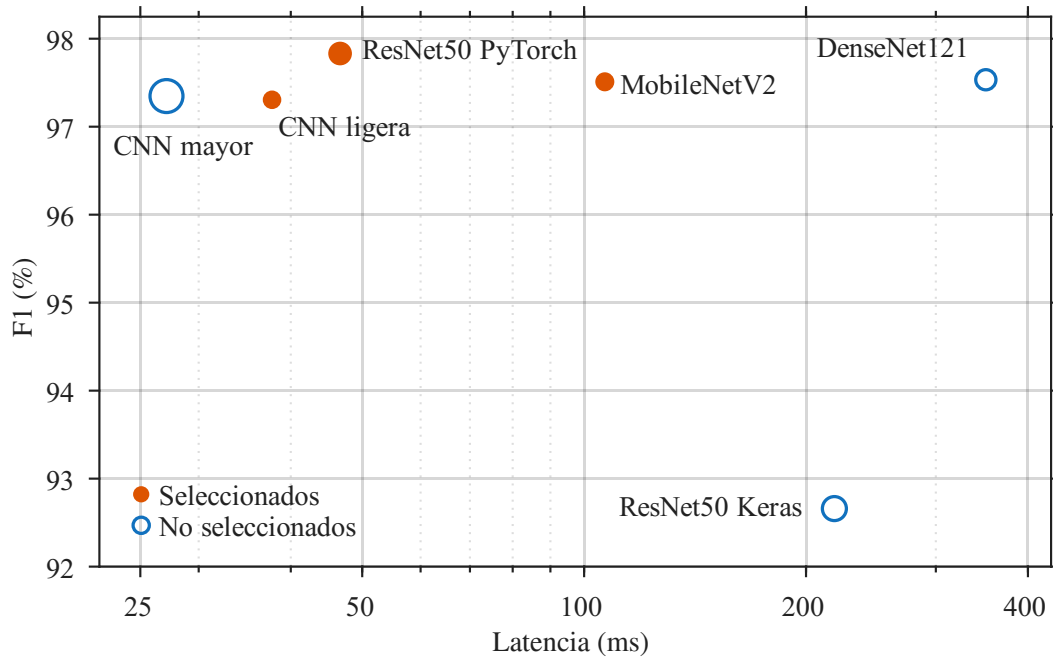


Figura 4.2: Relación entre latencia de inferencia y F1 para los modelos evaluados en CPU. El tamaño del marcador representa el tamaño del archivo original y el eje de latencia se presenta en escala logarítmica.

La Figura 4.2 permite visualizar conjuntamente latencia de inferencia, el desempeño predictivo y el tamaño de los artefactos originales. Se observa que modelos con valores de F1-score similares pueden ocupar regiones muy diferentes del espacio de diseño. La CNN ligera combina un valor de

latencia reducido con el menor tamaño de archivo del conjunto, mientras que ResNet50 PyTorch alcanza el mayor F1-score con un costo superior en tamaño. La CNN de mayor tamaño presenta el menor valor de latencia, pero esta ventaja se acompaña de un artefacto considerablemente más grande y de un recall menor frente a la clase “wildfire”. Estas diferencias refuerzan la necesidad de considerar múltiples criterios durante la selección de los modelos.

4.1.3 Resultado de la selección inicial

Los resultados anteriores muestran que ninguna de las seis alternativas presenta simultáneamente el mejor comportamiento en todas las dimensiones evaluadas. ResNet50 PyTorch presentó el mayor accuracy y F1-score, la CNN ligera obtuvo el mayor recall frente a “wildfire” y la CNN de mayor tamaño alcanzó el menor valor de latencia en CPU. Sin embargo, esta última también presentó el archivo de mayor tamaño y un número de falsos negativos considerablemente superior al de la CNN ligera.

En consecuencia, la selección de las tres arquitecturas que continuaron hacia las etapas de optimización respondió a compromisos diferentes. La CNN ligera fue seleccionada por combinar un recall elevado con una cantidad reducida de parámetros y un archivo compacto. MobileNetV2 se conservó como representante de una arquitectura específicamente orientada a eficiencia, mientras que ResNet50 PyTorch se utilizó como referencia de mayor complejidad y desempeño predictivo.

Esta selección permitió que las etapas posteriores no se limitaran a optimizar el modelo con mayor accuracy, sino que evaluaran cómo arquitecturas con características diferentes responden a la aceleración mediante TensorRT y a la reducción de precisión numérica.

4.2 Efecto de la optimización sobre el desempeño predictivo

Una vez seleccionadas las tres arquitecturas, se evaluó el efecto de las precisiones numéricas utilizadas en los engines de TensorRT sobre el desempeño predictivo. Las variantes FP32 se utilizaron como referencia para cada modelo y se compararon con las configuraciones FP16 y, cuando fue técnicamente viable, INT8.

Las variantes FP32 reprodujeron las métricas obtenidas previamente por los modelos originales en el benchmark en CPU. Este resultado muestra que la conversión a ONNX y la construcción de los motores de TensorRT conservaron el desempeño predictivo agregado sobre el conjunto evaluado.

La Tabla 4.3 resume los resultados obtenidos sobre las 6299 imágenes válidas del test split.

Tabla 4.3: Desempeño predictivo y matriz de confusión de las variantes TensorRT evaluadas sobre el test split.

Modelo	Formato	Accuracy (%)	Recall (%)	F1-score (%)	TN	FP	FN	TP	Cambios
CNN ligera	FP32	97,02	97,59	97,31	2716	104	84	3395	0
	FP16	97,02	97,59	97,31	2716	104	84	3395	0
	INT8	96,75	97,30	97,06	2709	111	94	3385	43
MobileNetV2	FP32	97,27	96,81	97,51	2759	61	111	3368	0
	FP16	97,24	96,75	97,48	2759	61	113	3366	2
ResNet50	FP32	97,62	97,27	97,83	2765	55	95	3384	0
	FP16	97,65	97,30	97,86	2766	54	94	3385	4

4.2.1 Comportamiento de las variantes FP16

FP16 conservó en gran medida el desempeño predictivo en las tres arquitecturas. En la CNN ligera, la variante FP16 produjo exactamente las mismas decisiones de clasificación que FP32, por lo que las métricas permanecieron sin cambios.

En MobileNetV2 se observaron únicamente dos cambios entre las 6299 predicciones. Ambos correspondieron a verdaderos positivos en FP32 que pasaron a ser falsos negativos en FP16. Por esta razón, los falsos negativos aumentaron de 111 a 113 y los verdaderos positivos disminuyeron de 3368 a 3366, mientras que las predicciones sobre la clase “nowildfire” permanecieron sin cambios.

ResNet50 presentó cuatro cambios de clasificación al utilizar FP16. De ellos, tres corrigieron predicciones respecto de FP32 y uno introdujo un nuevo error, produciendo una mejora neta de dos aciertos. Como resultado, los falsos positivos disminuyeron de 55 a 54 y los falsos negativos de 95 a 94. Esta variación reducida no se interpreta como una mejora intrínseca causada por FP16, sino como una conservación práctica del desempeño predictivo ante pequeñas diferencias numéricas.

En conjunto, los resultados muestran que FP16 fue una optimización estable desde el punto de vista de la clasificación. Ninguna de las tres arquitecturas presentó una degradación relevante del desempeño predictivo al pasar desde FP32 a esta precisión.

4.2.2 Comportamiento de la cuantización INT8

La cuantización INT8 presentó un comportamiento diferente según la arquitectura. De los tres modelos seleccionados, solamente la CNN ligera produjo una variante INT8 funcional que pudo continuar hasta la evaluación completa sobre el test split.

En la CNN ligera, INT8 produjo 43 cambios de clasificación respecto de FP32. El efecto neto sobre la matriz de confusión fue un aumento de los falsos positivos de 104 a 111 y de los falsos negativos de 84 a 94. En consecuencia, la cuantización introdujo siete falsos positivos y diez falsos

negativos adicionales en términos netos, lo que explica la disminución observada en accuracy y recall.

En MobileNetV2, los modelos ONNX cuantizados mantuvieron capacidad discriminativa al ser evaluados mediante ONNX Runtime. Sin embargo, los engines de TensorRT INT8 evaluados produjeron salidas prácticamente constantes y clasificaron las muestras de validación como “nowildfire”, perdiendo la capacidad de identificar la clase positiva. Debido a este comportamiento, la variante no fue utilizada sobre el test split ni incorporada a las campañas posteriores.

En ResNet50 tampoco se obtuvo un engine de TensorRT INT8 ejecutable. En consecuencia, la comparación posterior de esta arquitectura quedó limitada a las variantes FP32 y FP16.

Estos resultados muestran que la reducción a INT8 no presentó la misma viabilidad en las tres arquitecturas. Mientras que la CNN ligera permitió alcanzar una cuantización funcional con una degradación acotada del desempeño predictivo, MobileNetV2 y ResNet50 no produjeron variantes TensorRT INT8 aptas para continuar con la evaluación experimental. De esta forma, las campañas posteriores quedaron conformadas por las siete variantes funcionales presentadas en la Tabla 4.3.

4.3 Caracterización experimental en RTX 3060

Una vez verificado el desempeño predictivo de las siete variantes funcionales, se realizó una campaña común en la RTX 3060 para caracterizar el efecto de las optimizaciones sobre el desempeño computacional. Esta etapa permitió comparar las precisiones numéricas bajo condiciones homogéneas de ejecución y observar en qué medida las reducciones obtenidas dentro del engine de TensorRT se trasladaron al tiempo total de procesamiento y al comportamiento bajo carga sostenida.

4.3.1 Latencia de inferencia y flujo de procesamiento completo

La Tabla 4.4 presenta los resultados consolidados de latency. Se muestran el tiempo exclusivo de ejecución del engine medido mediante eventos CUDA, la frontera desde un tensor NumPy hasta la obtención de la predicción y el pipeline completo desde la lectura de la imagen. También se incluye el tamaño de cada engine de TensorRT.

La reducción de precisión produjo una disminución de latency en las tres arquitecturas. En la CNN ligera, FP16 redujo el tiempo desde el tensor NumPy hasta la predicción en aproximadamente 46 % respecto de FP32, mientras que INT8 alcanzó una reducción cercana al 58 %. La diferencia es todavía mayor al considerar únicamente la ejecución del engine, donde las reducciones fueron aproximadamente 59 % y 75 %, respectivamente.

Tabla 4.4: Latencia y tamaño de los engines de TensorRT en la RTX 3060.

Variante	Engine CUDA (ms)	NumPy→pred. (ms)	Pipeline (ms)	Engine (MiB)
CNN FP32	0,623	0,794	1,810	3,01
CNN FP16	0,257	0,427	1,513	1,03
CNN INT8	0,159	0,332	1,392	0,65
MobileNet FP32	0,637	0,840	2,851	10,19
MobileNet FP16	0,345	0,534	2,578	5,21
ResNet50 FP32	2,246	2,522	4,248	113,17
ResNet50 FP16	0,786	1,026	2,920	45,28

MobileNetV2 también presentó una disminución de latency con FP16. El tiempo de ejecución del engine se redujo cerca de 46 %, mientras que la frontera NumPy a predicción disminuyó aproximadamente 36 %. Sin embargo, el pipeline completo pasó de 2,85 ms a 2,58 ms, correspondiente a una reducción menor, cercana al 10 %. Esto muestra que una parte importante del tiempo total permanece asociada a las etapas anteriores a la inferencia, entre ellas el preprocesamiento mediante CLAHE.

El efecto de FP16 fue particularmente importante en ResNet50. Latency desde el tensor NumPy hasta la predicción disminuyó de 2,52 ms a 1,03 ms y el tiempo exclusivo del engine pasó de 2,25 ms a 0,79 ms. El pipeline completo también se redujo de 4,25 ms a 2,92 ms.

La reducción de precisión también disminuyó el tamaño de los engines. En la CNN ligera, el engine pasó de 3,01 MiB en FP32 a 1,03 MiB en FP16 y 0,65 MiB en INT8. MobileNetV2 redujo el tamaño de su engine de 10,19 MiB a 5,21 MiB, mientras que ResNet50 pasó de 113,17 MiB a 45,28 MiB. Esta diferencia resulta especialmente relevante en ResNet50, cuyo engine FP32 es considerablemente mayor que los correspondientes a las arquitecturas ligeras.

La Figura 4.3 evidencia la diferencia entre acelerar exclusivamente el engine y reducir el tiempo del pipeline completo. Este efecto es especialmente visible en MobileNetV2, donde FP16 disminuye de forma importante el tiempo del engine, pero la reducción sobre el pipeline es considerablemente menor. En la CNN ligera ocurre un comportamiento similar, aunque INT8 permite alcanzar el menor valor de latency en las tres fronteras temporales consideradas.

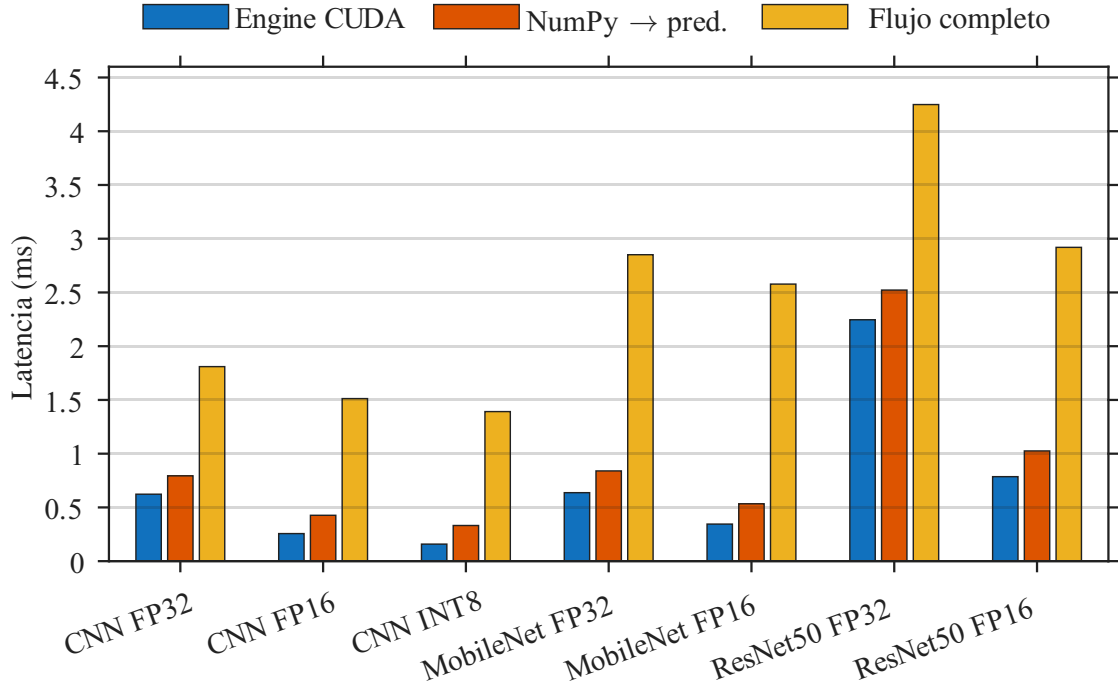


Figura 4.3: Comparación de las distintas fronteras de latencia para las variantes evaluadas en la RTX 3060.

4.3.2 Tasa de procesamiento y utilización de recursos

La campaña de carga sostenida permitió complementar las mediciones de latency mediante una ejecución continua de los engines. La Tabla 4.5 presenta el throughput obtenido junto con la utilización de memoria gráfica, la potencia máxima y la temperatura máxima registradas.

Tabla 4.5: Resultados de carga sostenida en la RTX 3060.

Variante	Throughput			
	(img/s)	VRAM (MiB)	Potencia máx. (W)	Temp. máx. (°C)
CNN FP32	1413,76	426	85,25	76
CNN FP16	2778,55	306	73,32	72
CNN INT8	3648,37	227	61,25	70
MobileNet FP32	1452,20	273	72,18	71
MobileNet FP16	2281,43	219	59,22	70
ResNet50 FP32	388,46	395	87,13	79
ResNet50 FP16	1138,14	261	82,71	77

Las variantes en precisión reducida presentaron un mayor throughput que sus referencias FP32. En la CNN ligera, FP16 prácticamente duplicó la cantidad de imágenes procesadas por segundo, mientras que INT8 alcanzó 3648,37 img/s, aproximadamente 2,6 veces el valor obtenido en FP32. Esta variante presentó una utilización de VRAM de 227 MiB, mientras que el menor valor del conjunto

correspondió a MobileNetV2 FP16, con 219 MiB.

MobileNetV2 aumentó su throughput desde 1452,20 img/s hasta 2281,43 img/s al utilizar FP16 y redujo la utilización de VRAM desde 273 MiB hasta 219 MiB. ResNet50 presentó el mayor incremento relativo de throughput entre FP32 y FP16, pasando de 388,46 img/s a 1138,14 img/s. A pesar de esta mejora, su variante FP16 mantuvo un throughput inferior a las variantes optimizadas de las arquitecturas ligeras.

Las precisiones reducidas también mostraron menores valores máximos de potencia y temperatura dentro de cada arquitectura. MobileNetV2 FP16 alcanzó la menor potencia máxima registrada, con 59,22 W, mientras que ResNet50 FP32 presentó la mayor, con 87,13 W. Las temperaturas máximas se mantuvieron entre 70 °C y 79 °C para las variantes consideradas.

4.3.3 Energía por inferencia

La Tabla 4.6 presenta las estimaciones de energía bruta e incremental por inferencia obtenidas durante las ventanas de carga sostenida.

Tabla 4.6: Energía por inferencia obtenida durante la carga sostenida en la RTX 3060.

Variante	Energía bruta (J/img)	Energía incremental (J/img)
CNN FP32	0,05944	0,04857
CNN FP16	0,02605	0,02060
CNN INT8	0,01646	0,01229
MobileNet FP32	0,04944	0,03888
MobileNet FP16	0,02545	0,01879
ResNet50 FP32	0,21606	0,17657
ResNet50 FP16	0,06971	0,05621

El efecto de la reducción de precisión también se observa de forma clara en el costo energético. En la CNN ligera, la energía incremental disminuyó aproximadamente 58 % al utilizar FP16 y cerca de 75 % con INT8 respecto de FP32. La variante CNN INT8 presentó el menor valor del conjunto, con 0,01229 J por imagen.

En MobileNetV2, FP16 redujo la energía incremental desde 0,03888 J/img hasta 0,01879 J/img, equivalente a una disminución cercana al 52 %. ResNet50 mostró una reducción desde 0,17657 J/img hasta 0,05621 J/img, aproximadamente 68 %. Aun después de esta reducción, ResNet50 FP16 presentó un costo energético superior al de las variantes FP16 de la CNN ligera y MobileNetV2.

La energía bruta presentó la misma tendencia general. Esto indica que las mejoras observadas no

se limitan a una disminución del tiempo de inferencia, sino que se reflejan también en una reducción del costo energético por imagen durante una ejecución sostenida.

La Figura 4.4 resume el compromiso entre throughput y costo energético. Las variantes de precisión reducida se desplazan hacia una región de mayor throughput y menor energía respecto de sus referencias FP32. La CNN INT8 ocupa la posición más favorable dentro de esta plataforma en ambas variables, aunque este resultado debe considerarse conjuntamente con la pequeña degradación de desempeño predictivo observada en la Sección 4.2.2.

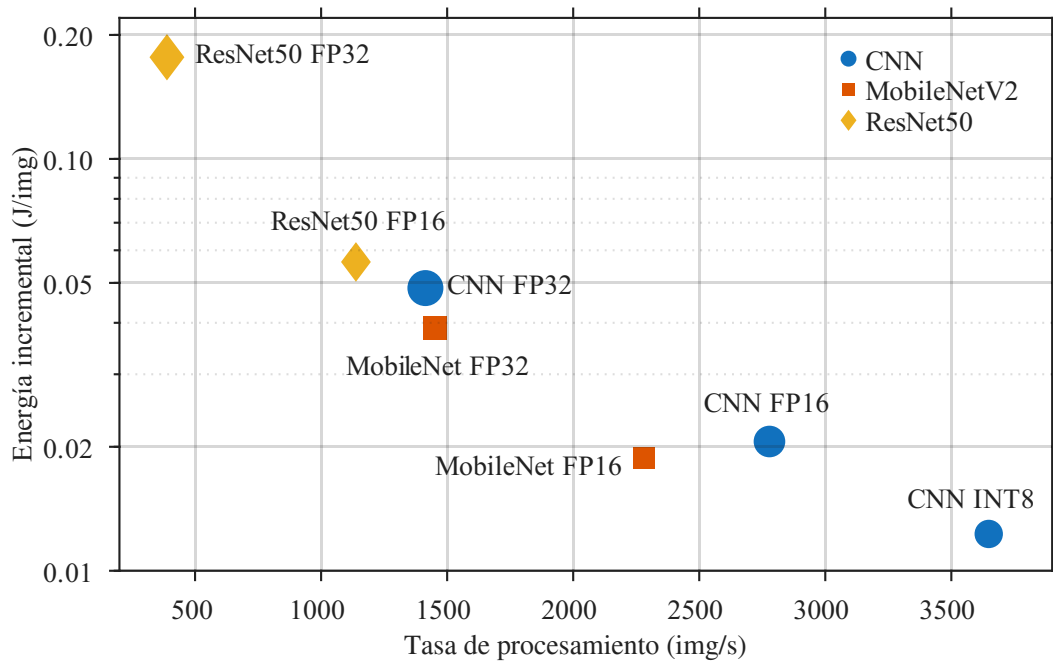


Figura 4.4: Relación entre tasa de procesamiento, energía incremental por inferencia y utilización de memoria gráfica en la RTX 3060. El tamaño de los marcadores representa la VRAM utilizada y el eje de energía se presenta en escala logarítmica.

En términos generales, FP16 produjo mejoras consistentes en las tres arquitecturas, reduciendo latency, tamaño del engine, utilización de memoria gráfica y energía por inferencia sin introducir una degradación relevante del desempeño predictivo. INT8 permitió extender estas mejoras en la CNN ligera, aunque su viabilidad no se reprodujo en MobileNetV2 ni ResNet50. Estos resultados caracterizan el efecto de las optimizaciones sobre la RTX 3060 y sirven como antecedente para analizar posteriormente su comportamiento en la plataforma embebida.

4.4 Resultados en Jetson Xavier NX

Las siete variantes funcionales fueron desplegadas posteriormente en la Jetson Xavier NX y evaluadas bajo los perfiles nominales de 10 W, 15 W con cuatro núcleos y 15 W con seis núcleos. Los resultados de esta sección corresponden a los engines de TensorRT contruidos de forma nativa bajo cada condición operacional.

En las tres campañas se mantuvo el mismo procedimiento de adquisición de potencia, utilizando el rail `VDD_IN` como medida de la potencia de entrada del módulo y un intervalo de muestreo de 200 ms. Por lo tanto, las mediciones energéticas obtenidas bajo los distintos perfiles son metodológicamente comparables. Los valores nominales asociados a `nvpmode1` se utilizan únicamente para identificar las condiciones de operación y no como sustituto de la potencia efectiva observada.

A diferencia de la campaña RTX, en esta etapa no se recalcularon las métricas de clasificación sobre las 6299 imágenes del test split. El desempeño predictivo permanece asociado a las variantes evaluadas en la Sección 4.2, mientras que la Xavier NX aporta la caracterización del despliegue y del desempeño computacional.

4.4.1 Validación del despliegue

Antes de analizar el desempeño computacional se verificó la correspondencia entre las salidas de los engines de TensorRT y las referencias generadas previamente. La validación final de los engines nativos utilizados en la campaña de 15 W con seis núcleos se resume en la Tabla 4.7.

Tabla 4.7: Validación de paridad de los engines de TensorRT en Jetson Xavier NX.

Variante	Concordancia num.	Error abs. máx.	Error abs. medio	Clases coincidentes
CNN FP32	Sí	$1,19 \times 10^{-7}$	$3,23 \times 10^{-8}$	32/32
CNN FP16	Sí	$3,33 \times 10^{-4}$	$9,63 \times 10^{-5}$	32/32
CNN INT8	Sí	$2,37 \times 10^{-3}$	$7,63 \times 10^{-5}$	32/32
MobileNet FP32	Sí	$1,16 \times 10^{-6}$	$1,46 \times 10^{-7}$	32/32
MobileNet FP16	No	$1,37 \times 10^{-2}$	$8,32 \times 10^{-4}$	200/200
ResNet50 FP32	Sí	$7,15 \times 10^{-7}$	$1,16 \times 10^{-7}$	32/32
ResNet50 FP16	No	$9,66 \times 10^{-3}$	$3,76 \times 10^{-4}$	200/200

Las variantes MobileNetV2 FP16 y ResNet50 FP16 no cumplieron el criterio de concordancia numérica definido en la metodología sobre las 32 entradas iniciales, aunque mantuvieron las 32 decisiones de clasificación. Por este motivo, su validación fue extendida a 200 entradas. En ambos casos, las 200 decisiones de clasificación coincidieron con la referencia, por lo que las diferencias

numéricas observadas en las salidas no se tradujeron en cambios de clase dentro del subconjunto evaluado.

Esta validación confirma la correspondencia del despliegue sobre las entradas utilizadas, pero no constituye una nueva medición de accuracy sobre el conjunto completo de prueba.

4.4.2 Latencia según el perfil de potencia

La Tabla 4.8 presenta la medición consolidada de latency de ejecución del engine de TensorRT para las tres condiciones evaluadas. Los valores corresponden a la mediana de las tres medianas obtenidas en cada campaña.

Tabla 4.8: Latencia del engine de TensorRT según el perfil operacional de la Jetson Xavier NX.

Variante	10 W (ms)	15 W, 4 núcleos (ms)	15 W, 6 núcleos (ms)
CNN FP32	7,50	4,30	4,75
CNN FP16	3,63	3,00	1,94
CNN INT8	2,26	2,71	1,28
MobileNet FP32	5,27	4,69	3,07
MobileNet FP16	2,88	3,15	1,82
ResNet50 FP32	27,84	13,87	13,87
ResNet50 FP16	7,97	4,68	4,60

En términos generales, los perfiles de 15 W permitieron reducir latency respecto de la configuración de 10 W. El efecto es particularmente marcado en ResNet50 FP32, cuyo valor de latency disminuyó desde 27,84 ms hasta aproximadamente 13,87 ms. ResNet50 FP16 también presentó una reducción importante, desde 7,97 ms hasta valores cercanos a 4,6 ms.

Las variantes de precisión reducida mantuvieron una ventaja clara dentro de cada arquitectura. En el perfil de 15 W con seis núcleos, CNN INT8 presentó el menor valor de latency del conjunto con 1,28 ms, seguida por MobileNetV2 FP16 con 1,82 ms y CNN FP16 con 1,94 ms.

No todas las mediciones presentaron un comportamiento estrictamente monotónico al aumentar el perfil de potencia. Por ejemplo, CNN INT8 y MobileNetV2 FP16 registraron un valor de latency ligeramente superior en 15 W con cuatro núcleos respecto de 10 W. Asimismo, CNN FP32 presentó un valor de latency menor en el perfil de cuatro núcleos que en el de seis. Estas variaciones deben interpretarse considerando el funcionamiento dinámico de DVFS y que los engines de TensorRT fueron reconstruidos específicamente para cada condición.

La Figura 4.5 resume visualmente la comparación entre los tres perfiles. Se observa que las

variantes de precisión reducida mantienen menores valores de latency dentro de cada arquitectura, mientras que el efecto de pasar entre los perfiles de 15 W depende de la variante evaluada y no presenta una tendencia uniforme.

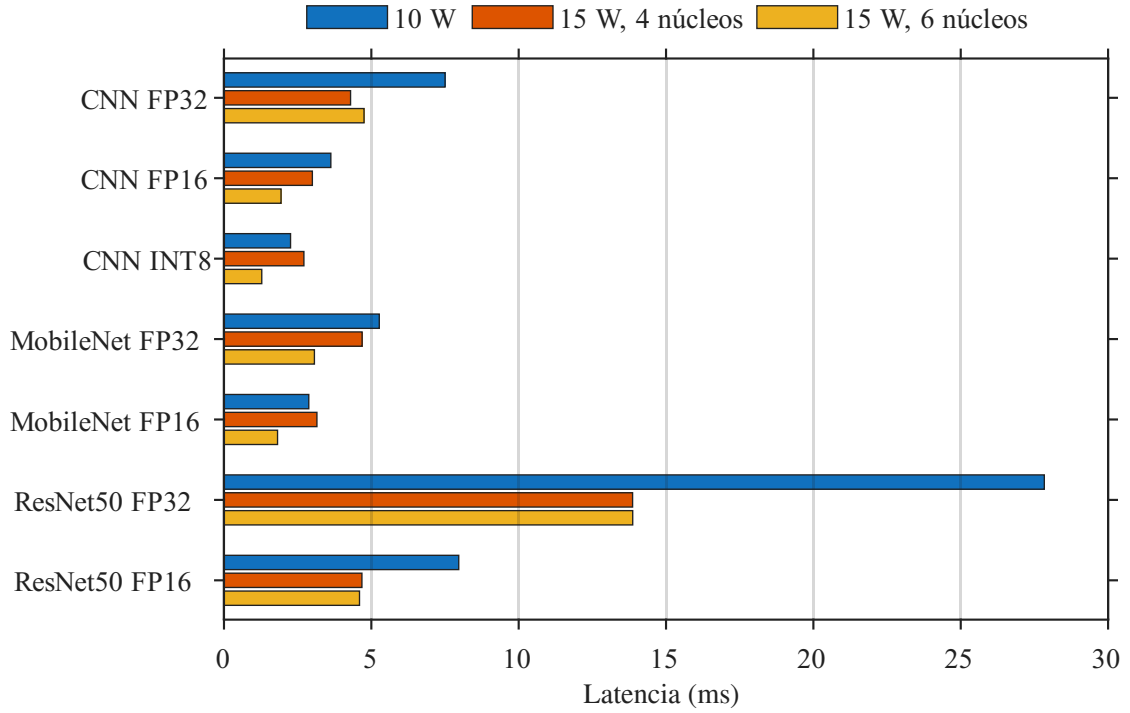


Figura 4.5: Latencia del engine de TensorRT para las variantes evaluadas bajo los tres perfiles de la Jetson Xavier NX.

4.4.3 Tasa de procesamiento sostenida

El comportamiento bajo carga sostenida se presenta en la Tabla 4.9. A diferencia del benchmark corto de latency, estos valores corresponden al número efectivo de inferencias completadas durante una ventana prolongada de ejecución.

Tabla 4.9: Tasa de procesamiento sostenida según el perfil operacional de la Jetson Xavier NX.

Variante	10 W (img/s)	15 W, 4 núcleos (img/s)	15 W, 6 núcleos (img/s)
CNN FP32	124,76	211,62	211,44
CNN FP16	290,33	459,26	457,27
CNN INT8	422,17	595,98	668,52
MobileNet FP32	174,48	279,34	278,16
MobileNet FP16	332,56	485,90	491,78
ResNet50 FP32	35,26	69,50	69,41
ResNet50 FP16	118,89	196,46	195,65

El throughput sostenido no corresponde necesariamente al inverso de latency obtenida en el benchmark corto. Ambas métricas se obtuvieron mediante procedimientos independientes: latency caracteriza ejecuciones individuales del engine, mientras que throughput se mide durante una carga continua previamente estabilizada. Debido a que DVFS permaneció activo, las condiciones dinámicas de operación pueden diferir entre ambas pruebas.

El aumento desde 10 W hasta 15 W con cuatro núcleos produjo incrementos de throughput entre aproximadamente 41 % y 97 %, dependiendo de la variante. ResNet50 FP32 presentó el mayor incremento relativo, mientras que CNN INT8 mantuvo el mayor throughput absoluto en los tres perfiles.

La comparación entre los dos perfiles de 15 W muestra un comportamiento distinto. En seis de las siete variantes, las diferencias de throughput se mantuvieron alrededor de un 1 % o menos. La principal excepción fue CNN INT8, que aumentó desde 595,98 img/s hasta 668,52 img/s, equivalente a aproximadamente un 12 %.

Por lo tanto, la utilización del perfil de seis núcleos no produjo una mejora generalizada de throughput respecto del perfil de cuatro núcleos para estas cargas. Esta observación describe el comportamiento de los despliegues evaluados y no se atribuye exclusivamente al número de núcleos activos, ya que los engines de TensorRT también fueron reconstruidos bajo cada configuración.

La Figura 4.6 resume visualmente el efecto de los perfiles operacionales sobre el throughput sostenido. El incremento desde 10 W hacia las configuraciones de 15 W es visible en todas las variantes, mientras que las diferencias entre los perfiles de cuatro y seis núcleos son reducidas en la mayoría de los casos, con CNN INT8 como principal excepción.

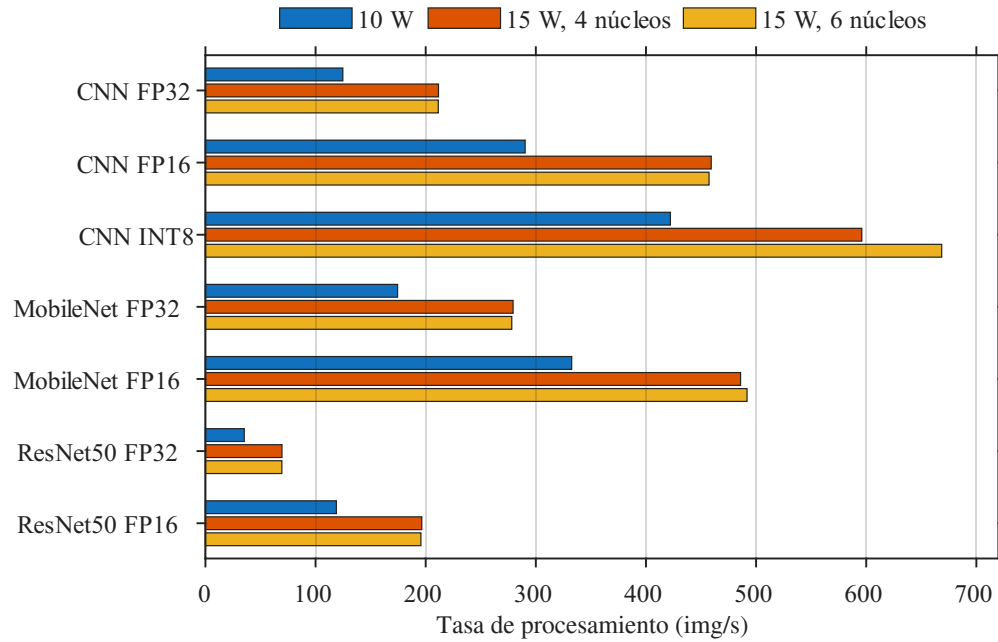


Figura 4.6: Tasa de procesamiento sostenida de las variantes evaluadas bajo los tres perfiles de la Jetson Xavier NX.

4.4.4 Energía incremental por inferencia

La Tabla 4.10 presenta la energía incremental por inferencia obtenida a partir de las mediciones de VDD_IN. Debido a que el mismo rail y protocolo de adquisición fueron utilizados en los tres perfiles, estos resultados permiten comparar de forma directa el costo energético adicional de las variantes bajo las distintas condiciones de operación. La métrica utiliza como referencia el consumo en reposo correspondiente a cada repetición.

Tabla 4.10: Energía incremental por inferencia según el perfil operacional de la Jetson Xavier NX.

Variante	10 W (J/img)	15 W, 4 núcleos (J/img)	15 W, 6 núcleos (J/img)
CNN FP32	0,05270	0,05878	0,05927
CNN FP16	0,01782	0,02001	0,02015
CNN INT8	0,01113	0,01263	0,01201
MobileNet FP32	0,02872	0,03154	0,03164
MobileNet FP16	0,01459	0,01583	0,01591
ResNet50 FP32	0,16268	0,18136	0,18207
ResNet50 FP16	0,04683	0,05229	0,05265

Las variantes en precisión reducida presentaron un menor costo energético dentro de cada arquitectura. En la CNN ligera, INT8 alcanzó los valores más bajos del conjunto, entre 0,01113 y

0,01263 J/img según el perfil. MobileNetV2 FP16 presentó valores entre 0,01459 y 0,01591 J/img, mientras que ResNet50 FP16 redujo de forma importante la energía respecto de su referencia FP32.

El paso desde 10 W hacia 15 W aumentó ligeramente la energía incremental por inferencia para todas las variantes. Entre 10 W y 15 W con cuatro núcleos, el incremento se situó aproximadamente entre 8 % y 14 %. Por lo tanto, el mayor throughput obtenido bajo el perfil de 15 W se acompañó de un costo energético por imagen algo superior.

Las diferencias energéticas entre los dos perfiles de 15 W fueron pequeñas. Para seis de las siete variantes se mantuvieron por debajo de 1 %, mientras que CNN INT8 presentó una disminución cercana al 5 % en el perfil de seis núcleos. En esta variante, el mismo perfil produjo además el mayor throughput de toda la campaña.

La Figura 4.7 permite comparar visualmente el costo energético de las variantes bajo los tres perfiles. Las configuraciones de precisión reducida mantienen una ventaja clara respecto de sus referencias FP32, mientras que el aumento hacia 15 W produce variaciones energéticas mucho menores que las observadas en throughput.

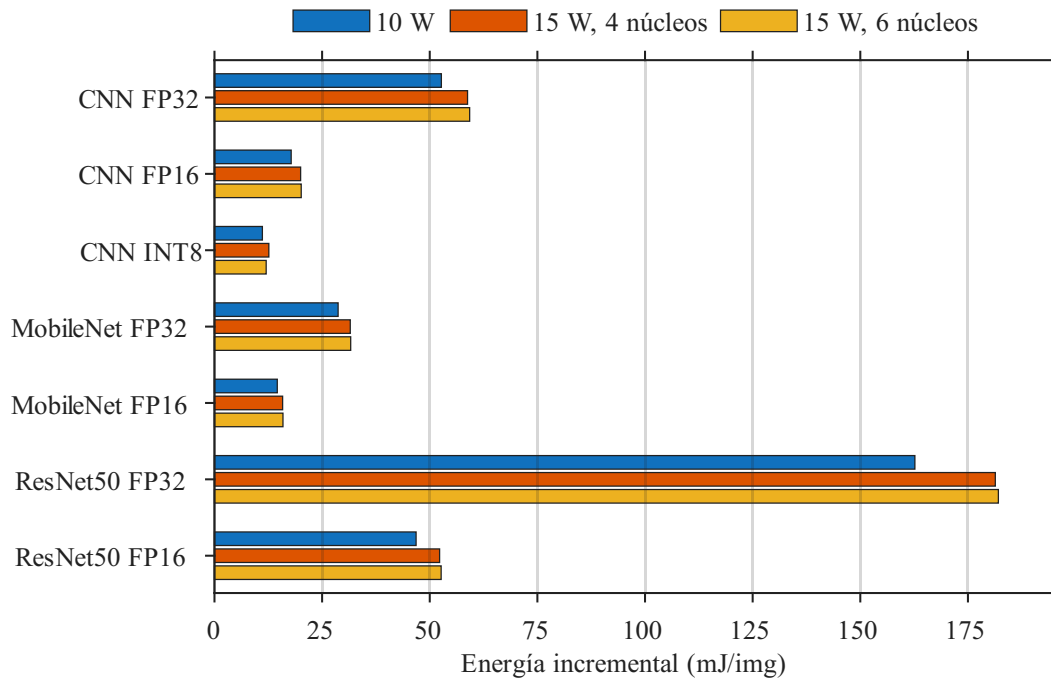


Figura 4.7: Energía incremental por inferencia de las variantes evaluadas bajo los tres perfiles de la Jetson Xavier NX.

4.4.5 Potencia efectiva de entrada del módulo

Además de la energía por inferencia, se analizó la potencia efectiva de entrada de la Jetson Xavier NX mediante el rail `VDD_IN`. En las tres campañas se utilizó el mismo procedimiento de adquisición, con un intervalo de muestreo de 200 ms durante las ventanas de carga sostenida. Para cada variante y perfil, la potencia promedio corresponde a la mediana de los promedios obtenidos en las tres repeticiones, mientras que la potencia máxima corresponde al mayor valor observado entre ellas.

La Tabla 4.11 presenta los valores consolidados. Los nombres de los perfiles de `nvmodel` corresponden a configuraciones nominales y no representan límites instantáneos rígidos de la potencia medida mediante `VDD_IN`.

Tabla 4.11: Potencia efectiva de entrada de la Jetson Xavier NX durante carga sostenida.

Variante	10 W		15 W - 4 núcleos		15 W - 6 núcleos	
	Prom. (W)	Máx. (W)	Prom. (W)	Máx. (W)	Prom. (W)	Máx. (W)
CNN FP32	8,829	9,420	14,732	14,950	14,873	15,155
CNN FP16	7,481	8,028	11,454	11,673	11,546	11,796
CNN INT8	6,999	7,659	9,835	10,199	10,364	10,567
MobileNet FP32	7,296	7,782	11,057	11,304	11,149	11,304
MobileNet FP16	7,141	7,618	9,990	10,240	10,129	10,403
ResNet50 FP32	8,014	8,560	14,857	15,032	14,987	15,196
ResNet50 FP16	7,860	8,355	12,505	12,656	12,615	12,820

Como era esperable, los perfiles de 15 W permitieron que la plataforma operara con una potencia de entrada superior a la observada en 10 W. Sin embargo, la potencia efectiva dependió también de la variante ejecutada. En el perfil de 15 W con seis núcleos, por ejemplo, los promedios bajo carga variaron desde 10,129 W para MobileNetV2 FP16 hasta 14,987 W para ResNet50 FP32.

Las variantes de precisión reducida presentaron, dentro de cada arquitectura, una menor potencia promedio que sus respectivas referencias FP32. Este comportamiento es consistente con las reducciones observadas previamente en energía por inferencia. CNN INT8 presentó la menor potencia promedio en los perfiles de 10 W y 15 W con cuatro núcleos, mientras que en la configuración de 15 W con seis núcleos el menor promedio correspondió a MobileNetV2 FP16, con 10,129 W.

Los máximos registrados tampoco coincidieron necesariamente con el valor nominal del perfil. En las configuraciones de 15 W se observaron valores ligeramente superiores a 15 W, alcanzando un máximo de 15,196 W para ResNet50 FP32 en el perfil de seis núcleos. Esto confirma que la denominación de `nvmodel` debe interpretarse como una configuración operacional y no como un límite instantáneo estricto de `VDD_IN`. Dado que la telemetría se adquirió cada 200 ms, los máximos

informados corresponden a los mayores valores observados en las muestras registradas y no descartan la existencia de transitorios de menor duración entre mediciones.

Todos los máximos de potencia observados durante las campañas permanecieron por debajo de la referencia de aproximadamente 19 W considerada para la carga adicional de procesamiento en el escenario CubeSat 12U descrito en este trabajo. La interpretación de esta relación y sus implicancias para la selección de las variantes se desarrolla posteriormente en la Sección 4.5.

4.5 Análisis comparativo y discusión

Los resultados obtenidos muestran que la reducción de precisión numérica puede producir mejoras importantes en el desempeño de inferencia, aunque su efectividad depende tanto de la arquitectura como del hardware utilizado. FP16 presentó el comportamiento más consistente, ya que pudo aplicarse a las tres arquitecturas seleccionadas manteniendo prácticamente inalterado su desempeño predictivo y reduciendo simultáneamente latency, el tamaño de los engines y el costo energético. INT8 permitió extender estas mejoras en la CNN ligera, pero no produjo variantes TensorRT funcionales para MobileNetV2 y ResNet50.

Esta tendencia se observó tanto en la RTX 3060 como en la Jetson Xavier NX. Los valores absolutos obtenidos en ambas plataformas no son directamente equivalentes debido a sus diferencias de hardware y a las distintas fronteras de medición energética. Sin embargo, en ambas campañas las variantes de precisión reducida presentaron menores valores de latency y energía por inferencia que sus respectivas referencias FP32. Este resultado indica que las ventajas de la optimización no estuvieron restringidas a la GPU utilizada durante el desarrollo, sino que se conservaron al realizar el despliegue sobre la plataforma embebida.

4.5.1 Compromisos entre desempeño predictivo y computacional

La Tabla 4.12 reúne las principales métricas de cuatro variantes representativas bajo el perfil de 15 W con seis núcleos. Esta configuración se utiliza como una condición común para contrastar las alternativas optimizadas, mientras que las diferencias entre los perfiles operacionales se analizan de forma separada en la Subsección 4.5.2. De esta forma, se pueden relacionar las métricas de desempeño predictivo previamente obtenidas sobre el test split con el desempeño medido durante el despliegue en la Xavier NX.

ResNet50 FP16 conserva el mayor accuracy y F1-score de las alternativas optimizadas, pero esta

Tabla 4.12: Comparación de alternativas optimizadas. Las métricas predictivas corresponden a la evaluación previa sobre las 6299 imágenes válidas; las métricas computacionales y energéticas corresponden a la Jetson Xavier NX bajo el perfil de 15 W con seis núcleos.

Variante	Accuracy (%)	Recall (%)	FN	Latency (ms)	Throughput (img/s)	E. incr. (J/img)	VDD_IN máx. (W)
CNN FP16	97,02	97,59	84	1,94	457,27	0,02015	11,796
CNN INT8	96,75	97,30	94	1,28	668,52	0,01201	10,567
MobileNet FP16	97,24	96,75	113	1,82	491,78	0,01591	10,403
ResNet50 FP16	97,65	97,30	94	4,60	195,65	0,05265	12,820

ventaja predictiva se obtiene a costa del mayor tiempo de inferencia y del mayor consumo energético entre las variantes mostradas. Frente a MobileNetV2 FP16, la diferencia de accuracy es inferior a medio punto porcentual, mientras que su throughput es considerablemente menor y su energía incremental por imagen es más de tres veces superior. En un escenario con recursos limitados, esta diferencia de desempeño predictivo debe evaluarse frente al costo computacional adicional requerido para obtenerla.

La CNN FP16 presenta una situación distinta. Su principal ventaja se encuentra en el recall frente a la clase “wildfire”, que alcanza 97,59 % y corresponde a 84 falsos negativos, el menor valor entre las alternativas evaluadas. Esto la convierte en una opción de interés cuando se otorga mayor importancia a reducir la cantidad de imágenes de la clase *wildfire* clasificadas incorrectamente. Su costo computacional continúa siendo reducido, aunque MobileNetV2 FP16 y CNN INT8 presentan un mayor throughput y menor energía por imagen.

MobileNetV2 FP16 ofrece un compromiso diferente. Sus valores de accuracy y F1-score son superiores a los obtenidos por las variantes optimizadas de la CNN, mientras que mantiene un valor de latency inferior a 2 ms, un throughput cercano a 492 img/s y una energía incremental de 0,01591 J/img. Su principal desventaja frente a las CNN se encuentra en el recall para la clase positiva, ya que registra 113 falsos negativos.

La CNN INT8 presenta el comportamiento más favorable desde el punto de vista de eficiencia computacional. Bajo el perfil de seis núcleos alcanzó el menor valor de latency, con 1,28 ms, el mayor throughput, con 668,52 img/s, y la menor energía incremental, con 0,01201 J/img. Respecto de MobileNetV2 FP16, esto representa aproximadamente un 36 % más de imágenes procesadas por segundo y un costo energético por imagen cerca de un 24 % menor. Además, su recall frente a “wildfire” es superior y presenta 19 falsos negativos menos.

La contraparte de esta ventaja se observa en el desempeño predictivo global. El accuracy de CNN INT8 es aproximadamente 0,49 puntos porcentuales inferior al de MobileNetV2 FP16 y su F1-score

es alrededor de 0,42 puntos porcentuales menor. Esta diferencia se explica en parte por una mayor cantidad de falsos positivos. Por lo tanto, la elección entre ambas alternativas depende del costo relativo que la aplicación asigne a falsos positivos y falsos negativos, además de los requerimientos de throughput y energía.

4.5.2 Efecto del perfil de potencia

El aumento desde el perfil nominal de 10 W hacia las configuraciones de 15 W produjo una mejora clara del throughput para las siete variantes. Esta ganancia estuvo acompañada por un incremento moderado de la energía por imagen, por lo que el perfil de mayor potencia no domina al de 10 W en todas las dimensiones.

La comparación entre los perfiles de 15 W con cuatro y seis núcleos mostró diferencias menores para la mayoría de las variantes en carga sostenida. La excepción más marcada fue CNN INT8, cuyo throughput aumentó desde 595,98 hasta 668,52 img/s, al mismo tiempo que su energía incremental disminuyó ligeramente desde 0,01263 hasta 0,01201 J/img. Para otras variantes, las diferencias de throughput y energía entre ambos perfiles fueron pequeñas.

Los resultados de latency muestran algunas diferencias adicionales a favor de la condición de seis núcleos, particularmente en las variantes de precisión reducida. Sin embargo, debido a que los engines de TensorRT fueron reconstruidos específicamente bajo cada perfil, estas diferencias corresponden al comportamiento completo del despliegue obtenido bajo cada configuración y no deben atribuirse únicamente al número de núcleos activos.

En consecuencia, el perfil de 10 W resulta atractivo cuando se prioriza minimizar el costo energético por imagen, mientras que las condiciones de 15 W permiten aumentar considerablemente el throughput. Entre las dos configuraciones de 15 W no se observa una ventaja general para todas las variantes, aunque el perfil de seis núcleos presenta un comportamiento particularmente favorable para CNN INT8.

4.5.3 Relación con el escenario energético de referencia

Silva estima para el escenario CubeSat 12U utilizado como referencia un margen de potencia de aproximadamente 18,9 W para una carga adicional de procesamiento y una ventana operacional de aproximadamente 5 minutos. Para dicha ventana, este margen equivale a una energía disponible del orden de 1,6 W h [7].

Las mediciones realizadas en la Xavier NX muestran que todas las variantes y perfiles evaluados

permanecieron bajo este margen en términos de potencia de entrada observada. El máximo de VDD_IN registrado en toda la campaña fue de 15,196 W, correspondiente a ResNet50 FP32 en el perfil de 15 W con seis núcleos. Este valor se encuentra aproximadamente 3,7 W por debajo del margen de referencia de 18,9 W.

Como estimación de referencia, si el máximo observado de 15,196 W se mantuviera durante la totalidad de una ventana de cinco minutos, la energía correspondiente sería

$$E_{5\min} = 15,196 \text{ W} \left(\frac{5}{60} \right) \text{ h} \approx 1,27 \text{ W h.} \quad (4.1)$$

La estimación supone que el mayor valor de potencia registrado se mantiene durante toda la ventana. No corresponde a una medición de cinco minutos ni constituye una cota superior garantizada, debido a la resolución temporal de la telemetría.

Estos resultados indican que, desde el punto de vista de la potencia de entrada medida en laboratorio, las cargas evaluadas son compatibles con el margen energético utilizado como referencia para el escenario considerado. Esto no implica que la Jetson Xavier NX constituya directamente una plataforma de vuelo ni que se haya validado su integración completa en un CubeSat. La comparación se limita al presupuesto energético disponible para la carga adicional de procesamiento.

4.5.4 Síntesis de la comparación

Los resultados no identifican una única variante que sea superior en todas las dimensiones. La alternativa más conveniente depende de qué aspecto se priorice dentro de la aplicación.

ResNet50 FP16 conserva el mayor desempeño predictivo global, pero presenta un costo computacional y energético considerablemente superior. CNN FP16 ofrece el mayor recall frente a “wildfire” y el menor número de falsos negativos. MobileNetV2 FP16 combina un desempeño predictivo global elevado con una reducción importante de los requerimientos computacionales. Finalmente, CNN INT8 proporciona la mayor eficiencia, alcanzando el menor valor de latency, el mayor throughput y la menor energía por inferencia de las alternativas evaluadas bajo el perfil de 15 W con seis núcleos.

En particular, MobileNetV2 FP16 representa una alternativa equilibrada cuando se busca conservar un desempeño predictivo global cercano al de ResNet50 con un costo considerablemente menor. CNN INT8 resulta especialmente favorable cuando se priorizan throughput y eficiencia energética, además de mantener un recall elevado para la clase “wildfire”. Si la reducción de falsos

negativos constituye el criterio principal, CNN FP16 mantiene el mayor recall del conjunto.

Por lo tanto, la evaluación realizada no conduce a una selección universal independiente de los requisitos de la misión. En cambio, permite identificar de forma cuantitativa los compromisos entre desempeño predictivo, latency, throughput y consumo energético, proporcionando criterios para seleccionar una configuración de inferencia de acuerdo con las prioridades operacionales del sistema.

Capítulo 5

Conclusiones y trabajo futuro

5.1 Conclusiones

Los resultados obtenidos muestran que la selección de una alternativa de inferencia para un sistema con recursos restringidos no puede basarse únicamente en el desempeño predictivo del modelo. Las arquitecturas evaluadas presentaron compromisos diferentes entre accuracy, recall frente a la clase “wildfire”, latency, throughput y consumo energético. En consecuencia, no se identificó una configuración universalmente superior, sino alternativas más convenientes según la prioridad asignada a cada una de estas métricas.

La reducción de precisión permitió mejorar de manera importante el desempeño computacional sin requerir el reentrenamiento de los modelos. FP16 presentó el comportamiento más consistente, ya que pudo aplicarse a las tres arquitecturas seleccionadas manteniendo prácticamente el desempeño predictivo de sus referencias FP32 y reduciendo simultáneamente latency y energía por inferencia. INT8 produjo beneficios aún mayores en la CNN ligera, pero mostró una dependencia más fuerte de la arquitectura y del runtime, dado que las variantes INT8 de MobileNetV2 y ResNet50 no dieron lugar a engines de TensorRT aptos para continuar con la campaña experimental.

Las alternativas optimizadas también mostraron ventajas diferentes entre sí. ResNet50 FP16 mantuvo el mayor desempeño predictivo global, CNN FP16 presentó el mayor recall frente a “wildfire” y el menor número de falsos negativos, MobileNetV2 FP16 ofreció un compromiso intermedio entre desempeño predictivo y costo computacional, y CNN INT8 alcanzó la mayor eficiencia de procesamiento entre las variantes evaluadas en la Xavier NX. Bajo el perfil de 15 W con seis núcleos, esta última alcanzó un valor de latency de 1,28 ms, un throughput de 668,52 img/s y una energía incremental de 0,01201 J/img.

La comparación entre plataformas confirmó además que el comportamiento de una variante depende de las condiciones concretas de ejecución. Las mejoras observadas en la GPU discreta no pueden trasladarse directamente a la plataforma embebida, y los engines optimizados deben construirse considerando el hardware y el runtime de TensorRT de destino. En la Xavier NX, el aumento desde el perfil nominal de 10 W hacia los perfiles de 15 W permitió incrementar de manera importante el throughput, mientras que las diferencias entre las configuraciones de cuatro y seis núcleos fueron reducidas para la mayoría de las variantes.

Desde el punto de vista del escenario CubeSat utilizado como referencia, las mediciones realizadas en la Xavier NX permanecieron dentro del margen energético considerado. El máximo de potencia de entrada observado fue de 15,196 W, inferior al margen aproximado de 18,9 W disponible para una carga adicional de procesamiento en el escenario de Silva [7]. Una proyección basada en este máximo observado durante una ventana completa de cinco minutos corresponde aproximadamente a 1,27 W h, también por debajo de la referencia cercana a 1,6 W h. Esta comparación no demuestra la integración de la Xavier NX en un sistema de vuelo, pero permite relacionar experimentalmente las cargas de inferencia evaluadas con una restricción energética concreta de la aplicación.

En relación con los objetivos planteados, la metodología implementada permitió obtener y comparar métricas de desempeño predictivo, latency, throughput y consumo energético bajo condiciones controladas y reproducibles. Asimismo, fue posible cuantificar el efecto de FP16 sobre las tres arquitecturas y de INT8 sobre la CNN ligera, además de documentar las limitaciones que impidieron obtener variantes INT8 funcionales para MobileNetV2 y ResNet50. En consecuencia, los resultados obtenidos permiten identificar de forma cuantitativa los compromisos asociados a distintas configuraciones de inferencia y proporcionan una base experimental para seleccionar alternativas de procesamiento de imágenes en escenarios con recursos y potencia restringidos.

En síntesis, la principal conclusión de esta memoria es que la optimización de inferencia debe abordarse como un problema conjunto de modelo, precisión numérica y plataforma de ejecución. Un mayor desempeño predictivo no implica necesariamente una mejor alternativa para un sistema restringido, del mismo modo que una menor precisión numérica no garantiza por sí sola una implementación viable. La evaluación conjunta de desempeño predictivo, latency y energía permite establecer estos compromisos de manera experimental y evita seleccionar una configuración utilizando una única métrica.

5.2 Trabajo futuro

Una primera línea de continuación consiste en ampliar la validación hacia datos más representativos de una operación satelital. La evaluación desarrollada en esta memoria se basó principalmente en el dataset público utilizado por los modelos originales, por lo que sería de interés estudiar su comportamiento sobre productos provenientes directamente de misiones de observación de la Tierra y bajo una mayor diversidad de condiciones geográficas, atmosféricas y de adquisición. Esto permitiría determinar en qué medida el desempeño observado se mantiene al trasladar los modelos hacia entradas más próximas a las disponibles en una aplicación operacional.

También resulta relevante extender la evaluación hacia un pipeline de procesamiento completo sobre la plataforma embebida. Las mediciones principales realizadas en la Xavier NX utilizaron tensores previamente preparados con el propósito de aislar el costo de la inferencia. Como continuación, podrían incorporarse la lectura del producto satelital, su decodificación, el preprocesamiento necesario y la generación del tensor de entrada, permitiendo caracterizar latency y consumo energético desde la recepción de los datos hasta la obtención de la clasificación.

Finalmente, una continuación natural consiste en estudiar cómo una etapa de clasificación como la evaluada en esta memoria podría integrarse dentro de un pipeline operacional más amplio de procesamiento a bordo. En particular, el pipeline propuesto por Silva genera estimaciones de susceptibilidad de ignición a partir de información espectral y variables complementarias, para posteriormente producir alertas georreferenciadas [7]. Una posible extensión sería evaluar el uso de un clasificador de imágenes como etapa complementaria para confirmar o priorizar regiones de interés identificadas previamente por el modelo de susceptibilidad. Esta integración requeriría adaptar y validar las entradas utilizadas por ambos componentes y evaluar el pipeline completo bajo las restricciones temporales y energéticas de la misión de referencia.

Bibliografía

- [1] G. Furano, G. Meoni, A. Dunne, D. Moloney, V. Ferlet-Cavrois, A. Tavoularis, J. Byrne, L. Buckley, M. Psarakis, K.-O. Voss, and L. Fanucci, “Towards the use of artificial intelligence on the edge in space systems: Challenges and opportunities,” *IEEE Aerospace and Electronic Systems Magazine*, vol. 35, no. 12, pp. 44–56, 2020.
- [2] L. Diana and P. Dini, “Review on hardware devices and software techniques enabling neural network inference onboard satellites,” *Remote Sensing*, vol. 16, no. 21, p. 3957, 2024.
- [3] D. J. Curnick, A. J. Davies, C. Duncan, R. Freeman, D. M. P. Jacoby, H. T. E. Shelley, C. Rossi, O. R. Wearn, M. J. Williamson, and N. Pettorelli, “SmallSats: A new technological frontier in ecology and conservation?” *Remote Sensing in Ecology and Conservation*, vol. 8, no. 2, pp. 139–150, 2022.
- [4] National Aeronautics and Space Administration, “State-of-the-art of small spacecraft technology,” <https://www.nasa.gov/smallsat-institute/sst-soa/>, 2026, informe institucional sobre tecnologías de satélites pequeños.
- [5] G. L. James, R. B. Ansaf, S. S. Al Samahi, R. D. Parker, J. M. Cutler, R. V. Gachette, and B. I. Ansaf, “An efficient wildfire detection system for AI-embedded applications using satellite imagery,” *Fire*, vol. 6, no. 4, p. 169, 2023.
- [6] M. M. H. Shuvo, S. K. Islam, J. Cheng, and B. I. Morshed, “Efficient acceleration of deep learning inference on resource-constrained edge devices: A review,” *Proceedings of the IEEE*, vol. 111, no. 1, pp. 42–91, 2023.
- [7] R. Silva Espinoza, “Modelo de edge computing a bordo de microsátélites para la susceptibilidad de incendios forestales en la zona centro y centro-sur de Chile,” 2026, tesina, Universidad Técnica Federico Santa María.

- [8] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [9] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.
- [10] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4510–4520.
- [11] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 4700–4708.
- [12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.
- [13] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2704–2713.
- [14] E. Tsalera, A. Papadakis, I. Voyiatzis, and M. Samarakou, "CNN-based, contextualized, real-time fire detection in computational resource-constrained environments," *Energy Reports*, vol. 9, pp. 247–257, 2023.
- [15] K. Thangavel, D. Spiller, R. Sabatini, S. Amici, S. T. Sasidharan, H. Fayek, and P. Marzocca, "Autonomous satellite wildfire detection using hyperspectral imagery and neural networks: A case study on australian wildfire," *Remote Sensing*, vol. 15, no. 3, p. 720, 2023.
- [16] H. Zheng, J. Duan, Y. Dong, and Y. Liu, "Real-time fire detection algorithms running on small embedded devices based on MobileNetV3 and YOLOv4," *Fire Ecology*, vol. 19, p. 31, 2023.
- [17] A. Cratere, A. Carbone, I. Cannizzaro, M. Ascioffa, D. Spiller, and F. Dell'Olio, "Enabling real-time wildfire detection through edge AI onboard CubeSat platforms," in *Artificial Intelligence and Image and Signal Processing for Remote Sensing XXXI*, ser. Proceedings of SPIE, vol. 13670. SPIE, 2025, p. 136700S.

- [18] A. Aaba, “Wildfire prediction dataset (satellite images),” <https://www.kaggle.com/datasets/abdelghaniaaba/wildfire-prediction-dataset>, 2023, conjunto de datos disponible en Kaggle.
- [19] humankey, “Wildfire classification using satellite images cnn,” Kaggle Notebook, consultado en agosto de 2026. [Online]. Available: <https://www.kaggle.com/code/humankey/wildfire-classification-using-satellite-images-cnn/notebook>
- [20] pratyushrao, “Wildfire detection,” Kaggle Notebook, consultado en agosto de 2026. [Online]. Available: <https://www.kaggle.com/code/pratyushrao/wildfire-detection/notebook>
- [21] alesengj, “notebook15bebdb990,” Kaggle Notebook, consultado en agosto de 2026. [Online]. Available: <https://www.kaggle.com/code/alesengj/notebook15bebdb990/notebook>

Repositorio reproducible y fuentes externas

Este anexo reúne la información necesaria para localizar el repositorio reproducible asociado a esta memoria y las fuentes externas utilizadas para obtener el dataset y los modelos originales. El repositorio acompaña el trabajo experimental y contiene los scripts, configuraciones, entornos y resultados consolidados seleccionados, pero no redistribuye el dataset, los pesos originales ni los engines de TensorRT.

Repositorio asociado

El código y la documentación utilizados para reproducir los procedimientos descritos en esta memoria se organizan en un repositorio público independiente.

`https://github.com/DiegoH119/memoria-benchmark-cubesat`

El repositorio se divide según los tres ejes experimentales del trabajo. La carpeta `cpu/` contiene el benchmark de los seis modelos originales y su procedimiento de consolidación. La carpeta `gpu/` reúne la exportación y validación ONNX, la cuantización INT8, la construcción de engines y los benchmarks realizados con TensorRT sobre GPU NVIDIA. La carpeta `jetson/` contiene el procedimiento específico utilizado en la Jetson Xavier NX, incluida la preparación de tensores, la compatibilidad con TensorRT 8.5, la validación de paridad y las campañas de latencia y carga sostenida. Los entornos de ejecución se documentan en `docker/` y los resultados públicos se encuentran en `results/`.

Cada eje contiene un archivo README con el propósito de los scripts, sus entradas, salidas y comandos de uso. El archivo `SHA256SUMS.txt` registra los hashes SHA-256 de los archivos publicados y permite comprobar la integridad de una copia del repositorio.

Dataset utilizado

El conjunto de datos utilizado corresponde a *Wildfire Prediction Dataset (Satellite Images)*, publicado en Kaggle por Abdelghani Aaba [18]. El dataset no se incorpora al repositorio y debe obtenerse desde su fuente original:

```
https://www.kaggle.com/datasets/abdelghaniaaba/wildfire-prediction-dataset
```

Las campañas descritas en esta memoria utilizaron la estructura de particiones proporcionada por dicho conjunto de datos. Las exclusiones y selecciones específicas empleadas durante la evaluación se documentan en la metodología y en los archivos de configuración del repositorio.

Fuentes de los modelos originales

Los seis modelos del benchmark inicial fueron obtenidos como artefactos asociados a notebooks públicos de Kaggle. Los pesos originales no se redistribuyen en el repositorio de esta memoria. Las fuentes registradas son las siguientes:

- `final_wildfire_model.keras`: notebook *Wildfire Classification Using Satellite Images CNN*, de humankey [19].

```
https://www.kaggle.com/code/humankey/wildfire-classification-using-satellite-images-cnn/notebook
```
- `resnet50_wildfire_20.pth`: notebook *Wildfire Detection*, de pratyushrao [20].

```
https://www.kaggle.com/code/pratyushrao/wildfire-detection/notebook
```
- `mobilenet.h5`, `densenet.h5`, `resnet.h5` y `cnn_model.h5`: notebook *notebook15bebdb990*, de alesengj [21].

```
https://www.kaggle.com/code/alesengj/notebook15bebdb990/notebook
```

Estos enlaces identifican la procedencia de los modelos utilizados y permiten obtener nuevamente las fuentes externas requeridas para reproducir las etapas que dependen de ellos.

Artefactos no incluidos en el repositorio

Con el propósito de evitar redistribuir artefactos externos o dependientes de hardware, no se incluyen el dataset, los modelos y pesos originales, los archivos ONNX derivados, los engines TensorRT, los tensores NPY/NPZ, las predicciones individuales, los reportes por repetición, los logs ni las series temporales completas.

Los ONNX pueden regenerarse mediante los scripts de exportación documentados y verificarse utilizando los hashes esperados. Del mismo modo, los tensores empleados en la Jetson Xavier NX pueden reconstruirse a partir del dataset y de la selección determinista registrada. Los engines TensorRT deben construirse nuevamente en la plataforma objetivo, ya que dependen del hardware y de la versión del stack de TensorRT utilizada.

Los resultados públicos del repositorio corresponden a consolidaciones de las campañas originales. La RTX 3060 utilizada durante el desarrollo constituye una plataforma experimental independiente y no se interpreta como una simulación de la Jetson Xavier NX ni de la plataforma CubeSat.

Verificación de integridad

En sistemas Linux, una copia del repositorio puede verificarse desde su directorio raíz mediante:

```
sha256sum -c SHA256SUMS.txt
```

El manifiesto permite detectar modificaciones accidentales en los archivos públicos. Las métricas de rendimiento, sin embargo, dependen del hardware, las versiones de CUDA y TensorRT, los drivers y las condiciones de ejecución, por lo que no se espera que una reproducción sobre una plataforma diferente entregue valores temporales, térmicos o energéticos idénticos a los reportados en esta memoria.