UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC DEPARTAMENT DE SISTEMES INFORMÀTICS I COMPUTACIÓ

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dpto. de Sistemas Informáticos y Computación

Desarrollo, optimización y despliegue de un servidor eficiente para la inferencia de redes neuronales de apoyo al radiodiagnóstico.

Trabajo Fin de Máster

Máster Universitario en Computación en la Nube y de Altas Prestaciones / Cloud and High-Performance Computing

AUTOR/A: Dorronsoro Larbide, Ibai

Tutor/a: Alonso Jordá, Pedro

Cotutor/a externo: DOLZ ZARAGOZA, MANUEL FRANCISCO

CURSO ACADÉMICO: 2022/2023

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Máster Universitario en Computación en la Nube y de Altas Prestaciones

Trabajo Final de Máster

Desarrollo, optimización y despliegue de un servidor eficiente para la inferencia de redes neuronales de apoyo al radiodiagnóstico

Autor:
Ibai Dorronsoro Larbide

Tutor:
Manuel F. Dolz Zaragozá
Cotutores:
Pedro Alonso Jordá
José I. Aliaga Estellés

Departamento de Sistemas Informáticos y Computación

Curso académico 2021/2022

Resumen

En este documento se presenta el Trabajo de Final de Máster del Máster Universitario en Computación en la Nube y de Altas Prestaciones de la Universidad Politécnica de Valencia, consistente en el desarrollo, optimización y puesta en marcha de un servidor de redes neuronales de altas prestaciones para el apoyo al radiodiagnóstico. La herramienta base utilizada para ello ha sido NVIDIA Triton Inference Server (Triton), un software de código abierto que permite estandarizar el despliegue y la ejecución de redes de inteligencia artificial (Artificial Intelligence o AI) independientemente del entorno de desarrollo en el que fueron construidos, posibilitando su ejecución en cualquier infraestructura hardware basada en CPU o GPU.

En el servidor construido, desplegado en el Hospital General Universitario de Castellón (HGUCS), se han implementado los modelos de AI desarrollados en dos proyectos de investigación liderados por la Universitat Jaume I (UJI) de Castellón, dirigidos ambos al apoyo al radiodiagnóstico para la detección de patologías en los servicios de urgencias hospitalarios: una red neuronal profunda de tipo convolucional y un segundo modelo de tipo transformador visual.

Una de las tareas fundamentales a la hora de implantar una solución de inferencia en AI a escala es completar una fase de optimización que permita cumplir con los requisitos de servicio de latencia y rendimiento de la solución. Para ello, se han realizado ensayos software sobre las redes anteriormente mencionadas, variando parámetros como la concurrencia de clientes, el tamaño de lote de la red neuronal y el tiempo de espera máximo hasta llenar el lote, con el objetivo de reducir la latencia y aumentar el rendimiento.

Estos test se han repetido en máquinas de características diferentes para valorar los cambios de productividad, latencia y rendimiento general en función de la plataforma hardware subyacente. En concreto, se ha trabajado con un nodo con dos GPUs NVIDIA RTX A6000, así como con tres plataformas de alto rendimiento y bajo consumo, Jetson Nano, Jetson Xavier y Jetson Orin, en las que se han probado distintos modos de configuración de la CPU y la GPU para ajustar el rendimiento y el consumo de energía.

Las imágenes a analizar se obtienen en formato DICOM (Digital Imaging and Communications in Medicine), estándar de transmisión y almacenamiento de datos médicos. Dado que Triton carece de soporte para dicho protocolo, ha sido necesario el desarrollo de un módulo extra para comunicarse con el sistema de información del HGUCS. Este modulo también almacena los resultados anonimizados de la inferencia de las imágenes médicas en una base de datos (*Database* o DB). En un futuro, los datos acumulados se utilizarán para realizar una validación doble ciego que permita comprobar el funcionamiento de las redes en un escenario real, donde una persona experta en radiología diagnosticará las mismas imágenes que el servidor en el HGUCS.

Palabras clave

Redes neuronales, inferencia, computación de altas prestaciones, eficiencia, optimización, imagen médica, aprendizaje profundo, DICOM, NVIDIA Triton.

Abstract

This document describes the Master's Dissertation of the Master's Degree in Cloud and High-Performance Computing of the Universidad Politécnica de Valencia, consisting of the development, optimization and implementation of a high performance neural network server to support radiodiagnosis. The core tool used to achieve this goal has been NVIDIA Triton Inference Server (Triton), an open-source software that allows standardizing the deployment and execution of artificial intelligence networks (Artificial Intelligence or AI) regardless of the development environment in which they were built, enabling their execution on any hardware infrastructure based on CPU or GPU.

In the built server, deployed at the General University Hospital of Castellón (HGUCS), the AI models developed in two research projects led by the Universitat Jaume I (UJI) of Castellón have been implemented, both aimed at supporting radiodiagnosis for the detection of pathologies in hospital emergency departments: a convolutional deep neural network and a second model of visual transformer type.

One of the essential tasks involved in implementing an AI inference solution at scale is to complete an optimization phase to meet the latency and throughput service requirements of the solution. To this end, software experiments have been performed on the aforementioned, varying parameters such as client concurrency, neural network batch size and maximum waiting time to fill the batch, with the aim of reducing latency and increasing throughput.

These tests have been repeated on machines with different characteristics to assess the throughput, latency and overall performance variations depending on the underlying hardware platform. Specifically, a node with two NVIDIA RTX A6000 GPUs, as well as three high-performance, low-power platforms, Jetson Nano, Jetson Xavier and Jetson Orin have been tested with different CPU and GPU configuration modes to tune performance and power consumption.

The images to be analyzed are obtained in DICOM (Digital Imaging and Communications in Medicine) format, a standard for transmission and storage of medical data. Since Triton lacks support for this protocol, it has been necessary to develop an extra module to communicate with the HGUCS information system. This module also stores the anonymized results of medical image inference in a database (DB). In the future, the accumulated data will be used to perform a double-blind validation to test the performance of the networks in a real scenario, where a radiology expert will diagnose the same images as the server at HGUCS.

Keywords

Neural networks, inference, high performance computing, efficiency, optimization, medical image, deep learning, DICOM, NVIDIA Triton.

Índice general

Capítulo 1

Introducción

En este capítulo se exponen brevemente las bases del trabajo, enumerando sus objetivos y explicando tanto el contexto como la motivación de los mismos. A continuación, se revisa el estado de la cuestión en el ámbito de este proyecto y se presenta la estructura de la memoria.

1.1. Contexto y motivación del trabajo

Las radiografías (RX) tienen un papel clave en la detección de distintas patologías y más concretamente en los servicios de urgencia de los hospitales. En toda España diariamente se genera un gran volumen de radiografías que requiere ser analizado por personal técnico especializado que, en algún caso, incluye imágenes de difícil interpretación. Teniendo en cuenta esto y la necesidad de una rápida respuesta en muchas de estas pruebas, los recursos sanitarios expertos en visualizar imágenes radiológicas urgentes pueden llegar a ser insuficientes.

En este entorno, investigadores del grupo de arquitecturas y computación de altas prestaciones (*High Performance Computing and Architectures* o HPC&A) de la UJI de Castellón (España) llevaron a cabo el proyecto RADiología Inteligente de precisión en procesos Asistenciales urgeNTes (RADIANT) que tiene actualmente su continuación con el proyecto Radiología intELIgente en procesos AsisteNCiales urgEntes (RELIANCE). El objetivo principal de ambos trabajos es el diseño de un sistema de apoyo al diagnóstico por imagen en procesos asistenciales urgentes, basado en técnicas de computación de AI y en el tratamiento de grandes volúmenes de datos (*Big Data*). Concretamente el grupo de trabajo ha producido un módulo de detección de patologías en RX de tórax (RXT) posterior anterior (PA) y anterior posterior (AP).

La herramienta piloto ha sido desarrollada empleando algoritmos de redes neuronales de

aprendizaje profundo (*Deep Learning* o DL) y en la actualidad, se dispone de dos versiones diferentes: un modelo basado en redes neuronales convolucionales (*Convolutional Neural Network* o CNN) y un modelo transformador de visión (*Vision Transformer* o VT) auto-supervisado.

Por su parte, los hospitales suelen estar equipados con múltiples aparatos de rayos X, desde donde se obtienen las imágenes de los pacientes a diagnosticar y que, posteriormente, se envían a un sistema de archivo y comunicación de imágenes (*Picture Archiving and Communication System* o PACS) que permite su almacenamiento, acceso, distribución y gestión de forma estructurada. La implementación de las comunicaciones y la definición del formato de los datos de estas imágenes viene determinada por el protocolo universal de intercambio de imágenes médicas digitales (*Digital Imaging and Communications in Medicine* o DICOM). Arquitecturas con estos componentes son muy comunes en el ámbito de la medicina, pero poco frecuentes fuera de él, por lo que las soluciones disponibles en el mercado son limitadas.

Este proyecto surge de la necesidad de disponer de un software compatible con los sistemas más habituales en la medicina actual, capaz de manejar los dos modelos de redes neuronales de los que se dispone en la actualidad, y que también esté preparado para incluir modelos que se desarrollen en el futuro. Además, el software desarrollado debe ser configurable para conseguir el máximo rendimiento posible en distintos entornos sanitarios reales, tanto hospitales como empresas privadas, donde los recursos y necesidades pueden ser muy diferentes.

1.2. Objetivos

El objetivo fundamental del trabajo es desarrollar, optimizar y desplegar de manera eficiente un servidor para la inferencia de redes neuronales de apoyo al radiodiagnóstico, que sea capaz de servir los modelos CNN y VT en un escenario real como es el HGUCS. Este objetivo general se puede desglosar en varios objetivos específicos:

Objetivo 1. Desarrollar el servidor del sistema inteligente sobre el que podrá ejecutarse el módulo de detección de patologías en RXT. La arquitectura de este sistema será de tipo cliente-servidor, que permitirá la recepción de imágenes e incluirá en su respuesta el resultado de la predicción de los hallazgos al centro solicitante.

Objetivo 2. Diseñar y desarrollar los protocolos de comunicaciones seguras y anonimizadas de imágenes DICOM y de informes médicos requeridos por el módulo de detección de patologías en RXT, para llevar a cabo la inferencia de dichas imágenes e informes.

Objetivo 3. Desplegar el servidor en el HGUCS, recibir archivos DICOM del PACS y realizar la inferencia de las imágenes que lleguen. El resultado generado para cada imagen se

almacenará en una DB para su posterior manejo en una validación doble ciego que pondrá a prueba la precisión del sistema desarrollado.

Objetivo 4. Analizar la latencia y el rendimiento de los modelos para distintas configuraciones del servidor, variando el hardware sobre el que se despliega. Se deberá hacer de manera sistemática y automatizada, para que pueda ser fácilmente replicable en diferentes entornos.

El diseño, desarrollo e implementación derivados de estos objetivos se deberá realizar de un modo ágil y estandarizado, lo que permitirá que el trabajo elaborado sea adaptable y extensible a entornos con particularidades variadas. En un futuro, la solución pretende implantarse en otros entornos sanitarios que pueden hacer uso de esta herramienta, como por ejemplo la empresa ActualTec S.L., donde la estructura del sistema está organizada como multicliente, en la que aparecen varios PACS enviando y recibiendo imágenes médicas, y donde los recursos están virtualizados en la nube.

A nivel de formación, el proyecto requiere profundizar en los conocimientos adquiridos durante el máster acerca de la computación de altas prestaciones (High Performance Computing o HPC), y su adaptación a la optimización de software sobre hardware de bajo consumo especialmente diseñado para AI. En esta labor, también ha sido necesario conocer el funcionamiento de las redes neuronales y su modelización, para explotar al máximo las características de las diferentes arquitecturas utilizadas.

1.3. Estado de la cuestión

La AI es la tecnología que está transformando el mundo actual. Calificada ya como la pólvora que impulsa la cuarta revolución industrial [1], su aplicación está tan normalizada en la sociedad que ni siquiera se es consciente de la potencia de las herramientas desarrolladas bajo su ámbito. Un ejemplo de esto es la aplicación Google Maps, utilizada por millones de usuarios cada día para averiguar el estado del tráfico en un momento dado y para ser informados al instante de posibles obras o accidentes en las carreteras.

El crecimiento vertiginoso de las aplicaciones basadas en inteligencia artificial ha sido posible gracias al incremento de la potencia de cálculo de los procesadores y al desarrollo de nuevas arquitecturas de aceleradores de tipo unidad de procesamiento gráfico [2] (*Graphics Processing Unit* o GPU). Las GPUs son procesadores programables potentes y altamente paralelos que brindan una capacidad aritmética máxima y un ancho de banda de memoria que supera considerablemente a su homólogo, la unidad central de procesamiento [3] (*Central Processing Unit* o CPU).

En 1965 Gordon Moore aseguró que el número de transistores en un microchip se duplicaría aproximadamente cada dos años, mientras que el precio de los ordenadores se reduciría a la mitad [4]. Hoy en día, la conocida como *Ley de Moore* está llegando a su fin, los microprocesadores son mucho más rápidos que en el pasado y están tan optimizados que las tendencias actuales de fabricación de procesadores se basan en la integración de un gran número de núcleos y unidades vectoriales de gran anchura [5]. Sin embargo, el progreso de la AI no hubiese sido posible sin el desarrollo de aceleradores de tipo GPU, tales como las tarjetas de NVIDIA A100 [6], o las H100 [7], con un elevado número de núcleos y procesadores tensoriales [8] (*tensor cores*) para la realización eficiente de productos de matrices.

Otro factor decisivo ha sido el desarrollo del Big Data [9] y el auge de la computación en la nube (*Cloud Computing*) [10]. De hecho, múltiples estudios han demostrado que los algoritmos de entrenamiento de redes neuronales para la clasificación de imágenes consiguen mejor precisión y solidez cuando trabajan con grandes volúmenes de datos [11]. Tecnologías como la computación en la nube [12] y el internet de las cosas [13] (*Internet of Things* o IoT) son las que permiten crear escenarios de recolecta y compartición masiva de datos, que más adelante pueden ser utilizados para entrenar las redes neuronales en el ámbito del aprendizaje profundo.

La última circunstancia determinante ha sido la mejora de las técnicas de entrenamiento de modelos. Nuevos algoritmos como el descenso de gradiente estocástico [14] (*Stochastic Gradient Descent* o SGD) han mostrado excelentes resultados para la optimización en el aprendizaje automático (*Machine Learning* o ML) y ahora son la opción dominante frente a sus competidores anteriores.

Una vez los modelos de red neuronal han sido entrenados, quedan preparados para realizar inferencia sobre nuevas muestras. En esta fase de inferencia se requiere una menor capacidad de cómputo y almacenamiento, lo que propicia que los modelos a implementar puedan ser llevados a plataformas especializadas, de menor coste y tamaño, que además suelen ser más eficientes. En los últimos años han ido apareciendo en el mercado placas de computación empotradas [15] limitadas en memoria y capacidad de procesamiento, diseñadas específicamente para hacer inferencia de modelos de redes neuronales. Algunos ejemplos son las computadoras Raspberry Pi y la gama NVIDIA Jetson, a la que pertenecen los modelos Nano, Xavier y Orin, sobre los cuales se validará la herramienta.

La computación en la nube y el IoT se ven complementados por la computación en el borde [16] (*Edge Computing*), cuyo objetivo es que los servicios se ejecuten lo más cerca posible de donde se encuentran los datos y sus usuarios, reduciendo así la latencia y agilizando el procesamiento. Esta tecnología es compatible con la integración de las previamente mencionadas infraestructuras especializadas para la inferencia, resultando en casos de uso como, por ejemplo, el análisis de vídeo con computación en el borde para la supervisión del tráfico en tiempo real en una ciudad inteligente [17].

En la actualidad, grandes empresas como NVIDIA con su entorno de inferencia Triton, y Google con su software TensorFlow Serving, han apostado por desarrollar servidores de inferencia especializados cuyo objetivo básico es el de ejecutar cualquier modelo de red neuronal en cualquier tipo de hardware basado en procesadores de tipo CPU o GPU. El éxito de estos servidores está siendo tal que las empresas líderes que ofrecen infraestructuras de aprendizaje automático están adoptándolas en sus plataformas. Así, Microsoft ha mejorado la calidad y eficiencia de su traductor utilizando Triton [18].

1.4. Estructura de la memoria

El trabajo llevado a cabo a lo largo de este proyecto está organizado en cinco capítulos diferentes. El contenido de cada uno de los capítulos se ha dividido de la siguiente manera:

- En el Capítulo 2 se describen brevemente los conceptos necesarios para la comprensión del desarrollo del trabajo. Tras una primera introducción de las redes neuronales con mención a las CNNs y los VTs, se realiza un análisis de las técnicas de compresión para dar respuesta a las necesidades de rendimiento de las redes neuronales con elevado número de capas como son las redes neuronales profundas. Posteriormente, se realiza una comparación de los entornos de inferencia disponibles en el mercado y se argumenta la elección tomada para este proyecto. A continuación, se introducen los sistemas de información hospitalarios, el formato de imágenes médicas y el protocolo de comunicación DICOM. El capítulo finaliza con una explicación de las arquitecturas de altas prestaciones.
- En el Capítulo 3 se explica en primer lugar la arquitectura global del sistema implantado en el HGUCS. Posteriormente, se describe cada módulo de la misma: el servidor intermedio desarrollado para la captura de las imágenes radiológicas, el servidor de inferencia detallando su funcionamiento interno y la configuración aplicada en este proyecto, así como el diseño de la DB donde se almacenarán las predicciones de inferencia que permitirá la posterior comparativa con el diagnóstico realizado por personal experto en radiología.
- El Capítulo 4 describe los resultados experimentales del proyecto. En este capítulo, se especifican las características hardware y software de las cuatro plataformas que se han empleado para medir el rendimiento del servidor de inferencia. También se muestran los problemas y limitaciones encontrados, así como las implementaciones llevadas a cabo para darles solución. A continuación, se especifica el desarrollo realizado para automatizar las mediciones y conseguir que se puedan lanzar los experimentos sin intervención humana. Más adelante, se detallan las características de las redes CNN y VT, especificando la red base de ambas. Finalmente, se analizan los resultados de los experimentos de carga realizados en cada plataforma hardware, además de mostrar las mejores configuraciones para el entorno del HGUCS en concreto.

- Por último, en el Capítulo 5 se debate sobre el cumplimento de los objetivos marcados, se razonan las conclusiones del ensayo realizado y se presentan las posibles líneas futuras de trabajo.

Capítulo 2

Conceptos previos

En este capítulo de la memoria se describen los conceptos básicos necesarios para la comprensión del desarrollo del trabajo, comenzando con una introducción a las redes neuronales, las técnicas de compresión de modelos y los entornos de inferencia. El capítulo finaliza describiendo los sistemas de información hospitalarios, el estándar DICOM y las arquitecturas de altas prestaciones.

2.1. Introducción a las redes neuronales

El DL es un subconjunto del ML [19] que se basa en el uso de redes neuronales artificiales (*Artificial Neural Network* o ANN) o redes neuronales simuladas (*Simulated Neural Network* o SNN), para la resolución de problemas en múltiples disciplinas, como la ingeniería, medicina y empresariales, entre otros [20]. El nombre y la estructura de las ANNs se inspira en el funcionamiento del cerebro humano e imitan la forma en que las neuronas biológicas trabajan y se comunican entre sí para procesar la información de entrada que generan los sentidos.

Las ANNs son una de las múltiples herramientas que se utilizan en el ámbito del aprendizaje automático para transformar entradas de datos complejas en datos procesados que las computadoras pueden entender. Es decir, son sistemas de aprendizaje computacional que utilizan una red de funciones para procesar una entrada de datos definida de forma determinada, y generar una salida que normalmente suele estar definida de forma diferente.

Las neuronas artificiales, también llamadas perceptrones, son los elementos que componen las ANNs y se modelan como una versión simplificada de las neuronas que se encuentran en el cerebro humano [21]. Todas las neuronas artificiales están conectadas a otras neuronas y la

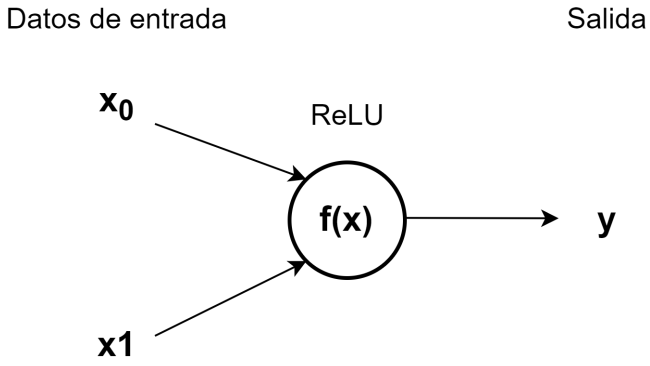


Figura 2.1: Neurona artificial con función de activación ReLU.

densidad y cantidad de las conexiones existentes es diferente dependiendo del tipo de ANN que se esté utilizando. Las conexiones entre neuronas tienen un peso asociado que representa la influencia que tiene la neurona de entrada sobre la de salida; cuanto mayor sea la magnitud del peso, mayor será su influencia.

Cada neurona (Figura 2.1) recibe entradas de otras neuronas, realizando una suma ponderada de las salidas de esas neuronas y los pesos de las conexiones y el resultado se pasa a la función de activación (no lineal). La no linealidad de la función de activación es lo que permite que las redes neuronales puedan aprender representaciones complejas [22]. Por ejemplo, la función de activación no lineal (*Rectified Linear Unit* o ReLU), comprueba si el valor de la neurona individual está por encima de su valor sesgo específico y, si lo está, la neurona se activa y envía el resultado del cálculo anterior a la siguiente capa de la red. De lo contrario, la información no se transmite a la siguiente capa.

La red neuronal suele organizarse en agrupaciones de neuronas, comúnmente conocidas como capas. Aunque el número de neuronas que se puede encontrar en cada capa puede ser diferente al de otra capa, todas las ANNs mantienen el mismo esquema: una capa de entrada, una o más capas ocultas y una capa de salida. Si aparece más de una capa oculta, la arquitectura resultante se denomina red multicapa. Las redes multicapa son mucho más sofisticadas que las redes artificiales monocapa y permiten dar solución a problemas más complejos. Hoy en día, la mayoría de redes neuronales que se utilizan siguen un modelo multicapa.

La Figura 2.2 muestra un ejemplo de una ANN de dos capas ocultas. La capa de entrada tiene tres neuronas, mientras que la de salida tiene dos, por lo que la red espera datos de entrada de dimensión tres e inferirá dos etiquetas de salida. El número de capas ocultas es lo que diferencia una red neuronal simple de una profunda (*Deep Neural Network* o DNN) [23], la red debe contener una gran cantidad de capas ocultas para ser catalogada como DNN. Dentro del ámbito del DL existen varios tipos de algoritmos de aprendizaje, en este trabajo se cubren

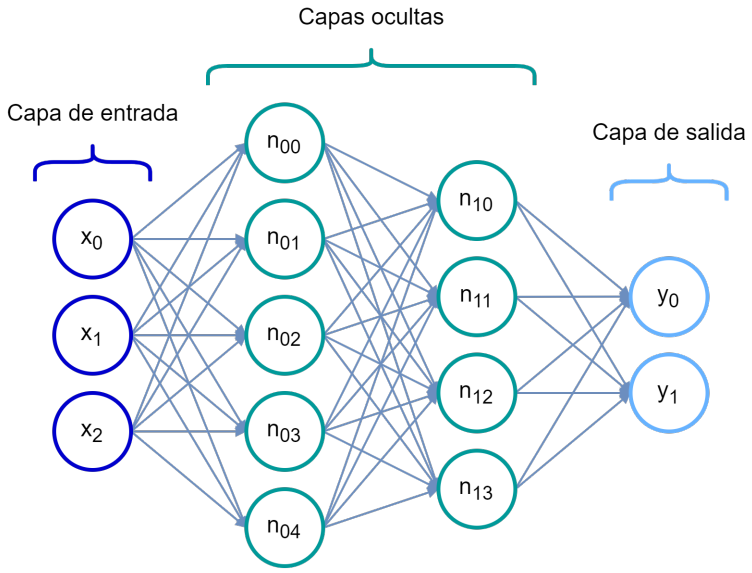


Figura 2.2: ANN multicapa totalmente conectada simple.

dos de ellas: las CNNs y las VTs.

2.1.1. Redes neuronales convolucionales

Las CNNs son un tipo de ANN en el que las neuronas se asemejan en gran medida a las que se pueden encontrar en el lóbulo frontal del cerebro de los seres humanos y los animales, que es donde se procesan los estímulos visuales. Surgieron como resultado del trabajo realizado por Hubel y Wiesel en 1959 para analizar cómo funciona la corteza visual, por lo que el propósito principal de las CNN es, hoy en día, el procesamiento de imágenes [24].

Las CNN se componen de múltiples capas convolucionales con filtros (*kernel*) de una o más dimensiones, cada una de las cuales es definida por un conjunto de parámetros de aprendizaje aplicados en las operaciones de convolución [25]. Cada *kernel* consiste normalmente en una matriz, con una serie de coeficientes numéricos que se aplican mediante una operación matemática llamada convolución.

Suponiendo que a una imagen de 28 x 28 x 1 píxeles se le aplican en la primera convolución 32 filtros de 1 canal, se pasaría a disponer de 28 x 28 x 32 neuronas [26]. Para reducir la cantidad de datos que se necesitan como dato de entrada en las sucesivas capas, estas capas de convolución se van alternado con otras cuya función es simplificar y extraer los datos más representativos obtenidos en el filtro utilizado anteriormente, en un proceso denominado *subsampling*.

El método más conocido de *subsampling* es el de *max pooling* [27], en la que se calcula el valor máximo de una matriz de características, y se utiliza para crear un mapa de características reducido. En la Figura 2.3 se ilustra un ejemplo en el que una matriz de características de dimensiones 4 x 4 es reducido a una 2 x 2 mediante *max pooling*. Otros métodos son el *average pooling* [28], que calcula la media de la matriz de características y el *stochastic pooling* [29], que hace la agrupación de manera aleatoria.

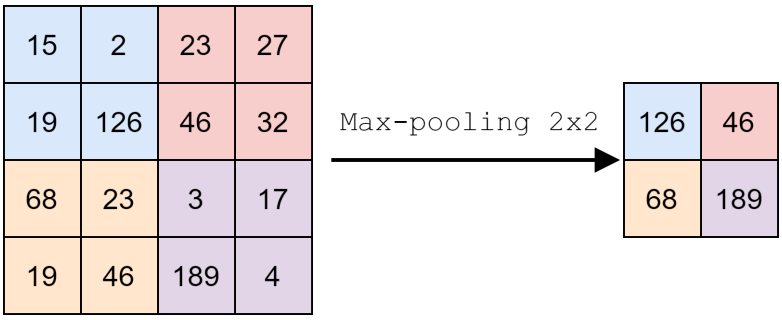


Figura 2.3: *Max pooling* de un mapa de características de 4x4 pixeles a 2x2.

Las CNNs se van construyendo mediante la repetición de estos procedimientos de filtrado y muestreo parcial (Figura 2.4). El tamaño de las imágenes resultantes en las capas más profundas es cada vez menor y la cantidad de filtros utilizada se va incrementando hasta el punto que las neuronas en capas muy profundas son mucho menos sensibles a variaciones básicas de las imágenes, y únicamente son activadas por características cada vez más complejas de las mismas.

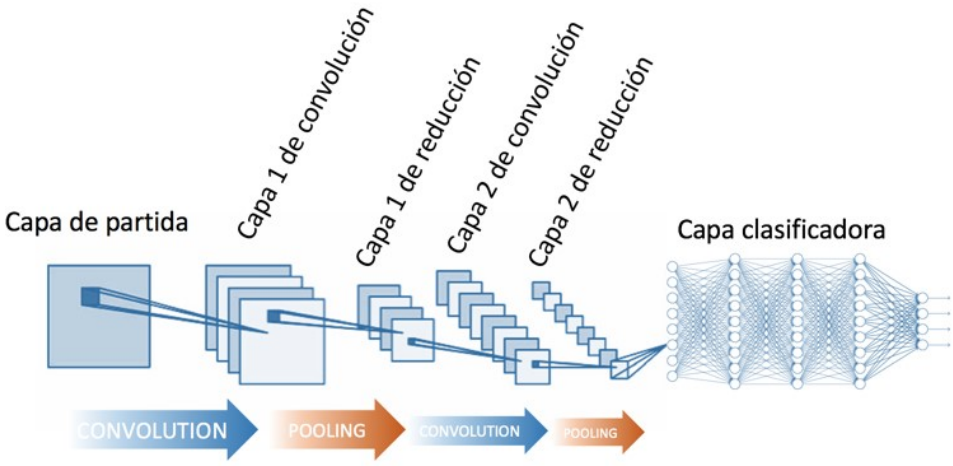


Figura 2.4: Construcción de una CNN.

De esta manera, a medida que aumenta la profundidad, los filtros de las capas convolucionales van aprendiendo las características y patrones relevantes de la imagen: la primera capa está

especializada en detectar patrones muy simples, como líneas o bordes, las posteriores son capaces de interconectar estos patrones y detectar diferentes formas, y así lograr tener una información que les permite reconocer la imagen.

Habitualmente, las CNNs disponen al final de una red neuronal con capas completamente conectadas, cuyo objetivo es interpretar y hacer predicciones correctas en base a las muestras de entrada. Por ello, la última capa de la red neuronal [30], la capa de clasificación, suele emplear la función de clasificación *softmax* [31] que asigna un vector de variables de características a una distribución de probabilidad. Es una de las funciones de clasificación más populares hoy en día y, además de en el ML, se ha utilizado para resolver problemas en campos como la teoría de juegos [32].

Para determinar los parámetros que definen los filtros de sus capas de convolución, los modelos que utilizan estas redes se someten a una fase de entrenamiento, con la ayuda de una gran cantidad de imágenes, que permiten ajustar iterativamente los pesos a aplicar en cada filtro, hasta que son capaces de predecir correctamente las imágenes utilizadas como entrada. El procedimiento de ajuste de pesos se realiza en un algoritmo de aprendizaje conocido como *retropropagación* [33].

2.1.2. Redes basadas en transformadores de visión

En el año 2017 aparecieron las arquitecturas basadas en la auto-supervisión, en particular los transformadores (*Transformers*) [34], con mejoras significativas en calidad, a la vez que en el tiempo de entrenamiento requerido frente a las redes predecesoras basadas en mecanismos de recurrencia. Estas nuevas arquitecturas proponen prescindir por completo de la recurrencia y basarse únicamente en mecanismos de atención que se asemejan al comportamiento de los seres humanos para percibir la información [35]: la tendencia es no procesar la información en su totalidad a la vez, concentrándose selectivamente en una parte, e ignorando otras. Así, si se descubre que una escena tiene a menudo una característica específica, se aprende a buscar esa característica cuando vuelvan a aparecer escenas similares.

Mientras que los transformadores habían establecido su predominancia en el campo del lenguaje natural (*Natural Language Processing* o NLP) [36], sus aplicaciones en el campo de la visión por ordenador eran limitadas. En particular, se aplicaban para reemplazar ciertos componentes de las CNNs pero manteniendo su estructura general [37]. En 2020 se demostró que un *Transformer* puro también puede tener un buen comportamiento en tareas de clasificación de imágenes [38]. Estas redes, denominadas transformadores de visión (*Vision Transformers* o VT), mantienen las ventajas sobre las CNNs de última generación, ya que logran una mejor calidad y necesitan muchos menos recursos computacionales para su entrenamiento.

La propuesta de los VTs consiste en descomponer las imágenes de entrada como una serie de

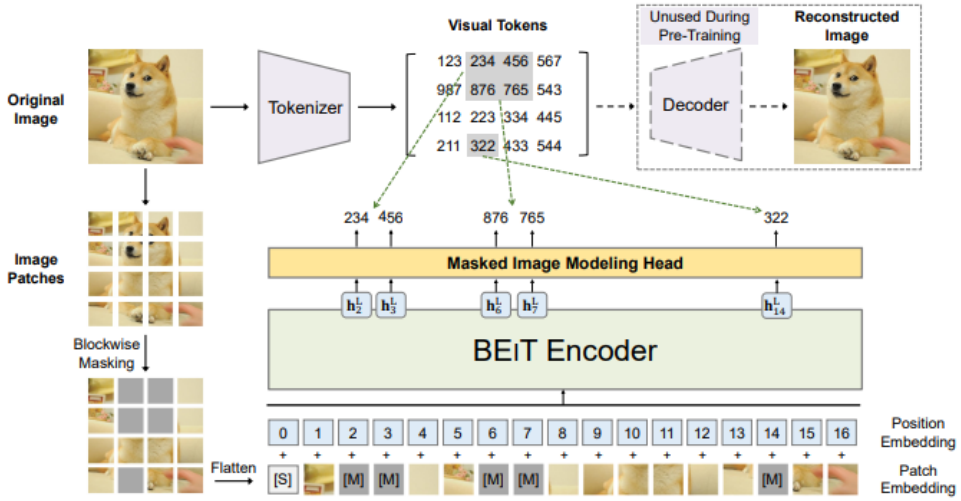


Figura 2.5: Preentrenamiento BEiT.

muestras visuales que, una vez transformados en vectores, se interpretan como palabras en un transformador normal. Si en el campo del NLP el mecanismo de atención de los transformadores captura las relaciones entre diferentes palabras del texto a analizar, los VTs intentan, en cambio, capturar las relaciones entre diferentes muestras de una imagen. Artículos recientes reflejan que esta diferencia es especialmente notoria en escenarios en los que se poseen abundantes datos de preentrenamiento [39].

Siguiendo este enfoque, el modelo BEiT (*Bidirectional Encoder representation from Image Transformers*) [40] fue presentado en la conferencia de *The International Conference on Learning Representations* (ICLR) [41] del año 2022, la principal reunión de profesionales dedicada al avance de la AI y el DL. Inspirado en BERT, la red neuronal BEiT de código abierto creada por Google para la compresión y NLP, plantea la novedad del modelado de imágenes enmascaradas para preentrenar los VTs. Los modelos BEiT se tratan, por lo tanto, de VTs preentrenados de forma autosupervisada.

Como se puede observar en la Figura 2.5, antes del entrenamiento las imágenes se dividen en representaciones visuales discretas (*visual tokens*). Durante el preentrenamiento, cada imagen tiene sus representaciones visuales (*image patches*) y sus *visual tokens* creados en la fase previa y, aleatoriamente, una parte de los *visual tokens* es sustituida por un valor especial (M). A continuación, las representaciones visuales son introducidas en un transformador de visión. Así, el objetivo principal del preentrenamiento es el de predecir las *image patches* de la imagen original, basándose en la imagen corrupta.

El hecho de que las DNNs estén compuestas por un gran número de capas también quiere decir que es necesario disponer de más recursos para gestionarlas que en aquellas redes con

menor número de capas. En concreto, el espacio de almacenamiento para guardar la red y el tiempo requerido para ejecutarlo en tiempo de ejecución puede aumentar considerablemente. Para casos en los que se dispone de plataformas de recursos limitados, existen técnicas como la compresión de modelos [42] para reducir el tamaño y el tiempo de inferencia de las redes neuronales.

2.2. Técnicas de compresión de modelos de red neuronal

La compresión de modelos de red neuronal es una técnica que permite convertir grandes y complejos modelos de redes neuronales en una versión reducida y de mayor velocidad, por lo general sin una pérdida significativa en su desempeño. Esta técnica se suele emplear para poder desplegar algún modelo de red neuronal en dispositivos con escasos recursos de memoria o en contextos donde la red neuronal sin comprimir no llega a cumplir con los mínimos requisitos de latencia [43]. Existen diversas técnicas de compresión, aplicables tanto en la fase de aprendizaje de los modelos como una vez éstos han sido ya entrenados. Entre los primeros, se destacan estos dos procedimientos:

- **La factorización de rango bajo [44] (*Low-Rank Factorization* o LRF):** consiste en tratar de identificar los parámetros redundantes de la red neuronal aplicando la descomposición matricial y tensorial, y factorizar cada matriz original en dos matrices más estrechas minimizando el error de Frobenius. Algunos artículos han mostrado que implementando esta técnica en la última capa de su modelo consiguen una reducción de tiempo de entrenamiento del 30-50 % en comparación con la representación en rango completo [45].

- **Destilación del conocimiento (*Knowledge Distillation* o KD) [46]:** este procedimiento sigue una arquitectura profesor-alumno, donde los conocimientos de una red neuronal compleja (profesor) son segregados para tratar de obtener una más simple (alumno). Esta división se lleva a cabo mediante métodos fundamentados en las técnicas de transferencia de conocimiento que se dan entre profesores y alumnos. En definitiva, la red alumno imita a la original tratando de obtener un rendimiento similar o incluso superior.

Dado que los modelos de red neuronal empleados en este trabajo están ya entrenados, este documento hará especial mención a los métodos de compresión que se aplican una vez la fase de entrenamiento está completada. Concretamente, se centrará en la poda de modelos y la cuantización, aunque estos también pueden aplicarse durante la fase de entrenamiento.

2.2.1. Poda

La poda de modelos elimina de forma sistemática determinados parámetros de una red existente [47]. El objetivo es aumentar la dispersión en las matrices de conexión de la red, incrementando así el número de parámetros con valor igual a cero en el modelo [48]. Generalmente, la red original es de gran tamaño y precisión, por lo que el propósito es producir una red más pequeña con una precisión equivalente. Existen distintos métodos de poda, diferenciándose, principalmente por las siguientes características:

- **Estructura:** en cuanto a la estructura de la dispersión, los parámetros pueden ser podados individualmente (desestructurado) o bien agrupados y eliminados en grupo (estructurado).
- **Clasificación:** los parámetros pueden clasificarse de muchas maneras, por ejemplo, por sus valores absolutos o por su contribución a las activaciones, entre otras.
- **Planificación:** la cantidad que se poda en cada etapa, por ejemplo, eliminando todos los pesos seleccionados en un solo paso o iterativamente en distintos pasos.
- **Reajuste:** en la característica del reajuste o *fine-tuning* [49] normalmente se establece una tasa de aprendizaje que se mantiene invariable. Cada peso no podado se ajusta a partir de este valor fijo y de su valor final entrenado.

Estudios recientes han mostrado una reducción de tamaño del 9-13 % en el modelo podado respecto al original, ambos sin pérdida de precisión [50]. Estos resultados pueden ser mejorados si se incluye una etapa de compresión que ejecuta distintas técnicas secuencialmente. Así, se comienza podando el modelo para reducir el número de pesos, seguido de una cuantización para disminuir el número de bits de cada peso y finaliza con la codificación de Huffman para una mayor compresión. Con esta secuencia de pasos el artículo consigue una disminución en el espacio que el modelo necesita de un 35-49 %.

2.2.2. Cuantización

La técnica de la cuantización se refiere a minimizar el número de bits necesario para almacenar un peso preservando la exactitud del modelo asociado [51]. Para ello, las representaciones en coma flotante suelen ser transformadas en valores enteros fijos o representaciones en coma flotante de menor precisión. En pocas palabras, se reduce el rango de representación de los pesos asociados al modelo con el propósito de obtener otras ventajas. El objetivo de cuantizar una red neuronal es el de reducir el tamaño necesario para almacenarla, acelerar el tiempo de inferencia y reducir el consumo de energía, maximizando su precisión.

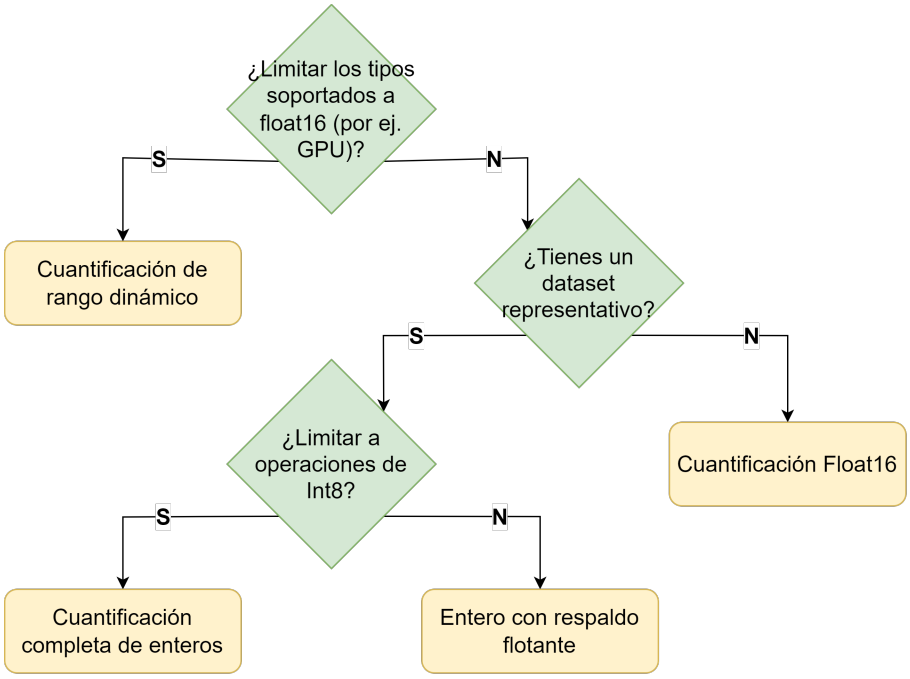


Figura 2.6: Árbol de decisión para cuantizar con TfLite.

En general, hay dos aproximaciones para aplicar la cuantización: convertir un modelo de coma flotante preentrenado en uno de coma fijo sin entrenamiento o llevar a cabo el entrenamiento del modelo imponiéndole restricción de coma fijo. Diferentes estudios demuestran que la segunda aproximación puede alcanzar una precisión superior a la primera [52]. No obstante, tiene la desventaja de ser mucho más complicado de implantar, ya que, requiere de una estrecha integración con el diseño y el entrenamiento de la red.

Los pesos de las redes neuronales se almacenan normalmente como números de coma flotante de 32-bits. Estos pueden ser cuantizados a precisiones de 16-bit, 8-bit, 4-bit y hasta 1-bit, lo que se conoce como binarización de pesos o *weight binarization* [53]. El árbol de decisión que proporciona TensorFlow Lite (TfLite) para cuantizar los modelos se puede ver en la Figura 2.6 y en la Tabla 2.1.

Asimismo, recientes publicaciones muestran pérdidas inferiores al 1 % al aplicar la cuantización de coma fija sobre CNNs [54]. El mismo artículo reporta que utilizando cuantización por capas se consigue un 53 % menor consumo de memoria y un ahorro del 77,5 % del coste de ejecución de las multiplicaciones comparados con herramientas que realizan una cuantización a 8-bits de todos los parámetros.

Tabla 2.1: Hojas del árbol de decisión de cuantización de TfLite.

Cuantización	Beneficios	Hardware
Rango dinámico (16-bit)	4x más pequeño 2x-3x de aceleración	UPC
Float16 (16-bit)	4x más pequeño 3x+ de aceleración	CPU, Edge TPU, Microcontroladores
Completa de enteros (8-bit)	2 veces más pequeño Aceleración de GPU	CPU, GPU
Entero con respaldo flotante (8-bit)	2 veces más pequeño Aceleración de GPU	CPU, GPU

2.3. Entornos de inferencia

Un servidor de inferencia es un software que se emplea para estandarizar y facilitar la implementación y ejecución de modelos de redes neuronales entrenados de manera rápida y escalable en producción. El abanico de opciones disponibles en el mercado actual es muy amplio, se enuncian a continuación los más representativos:

TorchServe [55]: TorchServe es una herramienta de código abierto flexible y fácil de usar para servir modelos PyTorch, creado por Facebook. Proporciona una herramienta sencilla para empaquetar modelos proporcionando un fácil versionado. Es capaz de manejar múltiples modelos en una instancia y es muy fácil de escalar. Incorpora una API REST sencilla tanto para la inferencia como para la gestión de modelos y ofrece métricas que pueden cargarse fácilmente. Soporta no sólo la API HTTP sino también la API gRPC y es compatible con el procesamiento por lotes.

TensorFlow Serving [56]: creado por Google, es un sistema caracterizado por su flexibilidad y alto rendimiento. Al igual que TorchServe, soporta las APIs HTTP y gRPC, tanto para la inferencia como para la gestión de modelos. Puede manejar múltiples modelos o múltiples versiones del mismo modelo simultáneamente. A diferencia de TorchServe, no está limitado al uso de controladores de Python para poder manejar un modelo.

NVIDIA Triton [57]: Triton, el servidor de inferencia de NVIDIA, es un servidor de código abierto que destaca por ser compatible con múltiples marcos de aprendizaje profundo (TensorRT, TensorFlow SavedModel, ONNX y TorchScript) e incluso con scripts de Python o aplicaciones C++. Triton admite redirigir la salida de un modelo a la entrada de otro, creando tuberías entre modelos, cada uno de los cuales puede utilizar un entorno de desarrollo diferente. También es capaz de proporcionar métricas que indican la utilización de la GPU, el rendimiento del servidor y la latencia del servidor en formato de datos

Prometheus [58], un sistema de recuperación de datos basado en operaciones de tipo MapReduce [59]. Soporta APIs REST y gRPC que permiten a los clientes remotos solicitar inferencias para cualquier modelo que esté siendo gestionado por el servidor. Además, proporciona una solución de inferencia para la nube y otra para entornos de *Edge Computing* optimizada tanto para las CPU como para las GPU. Para estos últimos entornos, Triton puede instalarse como una biblioteca compartida con una API de C que permite incluir su funcionalidad completa directamente dentro de una aplicación.

Multi Model Server [60]: este entorno fue creado por Amazon Web Services (AWS), y está optimizado para ejecutarse en la infraestructura de AWS, pero puede funcionar en cualquier sistema basado en UNIX. Está fuertemente ligado a TorchServe, por lo que soporta los mismos entornos de desarrollo y utiliza la misma herramienta de empaquetado de modelos. Al igual que los demás entornos, también proporciona una API REST tanto para la inferencia como para la gestión de los modelos.

La elección del servidor de inferencia a la hora de poner en producción un modelo de red neuronal concreto es una decisión crítica, condicionada por el entorno de desarrollo utilizado en la creación de la red neuronal que se va a servir. Debido a ello, en la mayoría de los casos resulta beneficioso utilizar el mismo entorno de desarrollo que el del modelo de red neuronal.

En este proyecto, se trabajará con un modelo construido en TensorFlow y otro en PyTorch, y puede que en el futuro se utilicen otros entornos. La posibilidad de utilizar modelos desarrollados en diferentes entornos ha sido un factor determinante para seleccionar Triton como el servidor de inferencia. Además, Triton es un servidor que se está posicionando líder en el mercado y ha conseguido un rendimiento récord en todo el catálogo de categorías de la última versión de MLCommons (antiguo MLPerf) [61]. Esta herramienta de medición de prestaciones incluye un conjunto de pruebas de inferencia de ML estándar con métricas correspondientes y un método de evaluación de análisis comparativo para medir la velocidad y el rendimiento del software/hardware ML.

2.4. Sistema de información hospitalario. Estándar DICOM

La información base de este proyecto la constituyen las imágenes médicas [62], en concreto las RXT, a partir de las cuales se debe emitir un diagnóstico clínico de manera similar a lo que haría un especialista humano en radiología. Estas imágenes son generadas en diferentes aparatos o equipos médicos y pueden ser visualizadas en estaciones clínicas conectadas a ellos, aunque lo habitual es que sean enviadas a un PACS [63]. Un PACS es un sistema que permite almacenar y acceder de manera homogénea a una gran variedad de imágenes médicas, siendo ampliamente utilizado en los hospitales y en general en todo tipo de centros de atención médica. Los PACS almacenan no sólo RX, sino también imágenes de diferentes modalidades [64], como es el caso

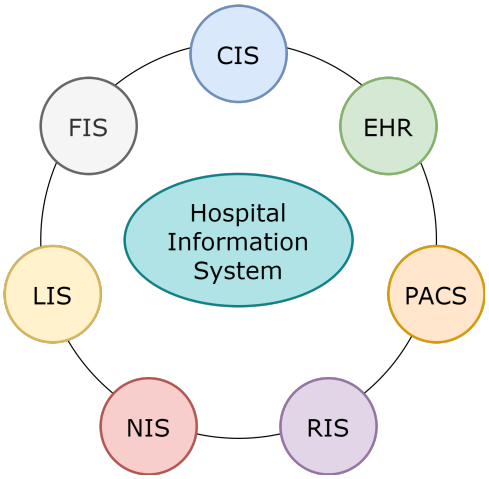


Figura 2.7: Posibles componentes de un HIS.

de mamografías (*mammography* o MG), resonancias magnéticas (*magnetic resonance* o MR), ultrasonidos (*ultrasound* o US) y tomografías computarizadas (*computerized tomography* o CT).

Los PACS comparten información e interactúan con el resto del sistema de información del hospital (*Hospital Information System o HIS*) [65], un subsistema que gestiona todos los sistemas de procesamiento de la información, tanto los aspectos clínicos como los administrativos y financieros. El HIS se compone de varios componentes (Figura 2.7) entre los que se encuentra el sistema de información clínica (*Clinical Information System* o CIS), el sistema de información financiera (*Finantial Information System* o FIS), el sistema de información de laboratorio (*Laboratory Information System* o LIS), el sistema de información de enfermería (*Nursing Information System* o NIS), el sistema de información de farmacia (*Pharmacy Information System* o PIS), el PACS, el sistema de información de radiología (*Radiology Information System* o RIS) y la historia clínica electrónica (*Electronic Health Records* o EHR).

El RIS es el sistema de información diseñado para dar respuesta a las necesidades del servicio de radiología y a través del cual se pueden realizar tareas como citar a un paciente, realizar un seguimiento de su tratamiento o acceder a otro tipo de información del mismo. En la Figura 2.8 se ilustra una posible integración de un RIS en un entorno HIS con un PACS para la gestión de imágenes médicas.

El envío y almacenamiento de las imágenes médicas se realiza utilizando el estándar DICOM, un estándar de transmisión y almacenamiento de imágenes y datos de carácter médico que nació con el propósito de permitir la interoperabilidad entre dispositivos de diferentes fabricantes. Su antecedente es el grupo de normas de interconexión de dispositivos ACR-NEMA [66], concebido en la década de 1980 por el Colegio Americano de Radiología (*American Asociation of Radiology* o ACR) y la Asociación Nacional de Fabricantes de Material Eléctrico (*National Electrical*

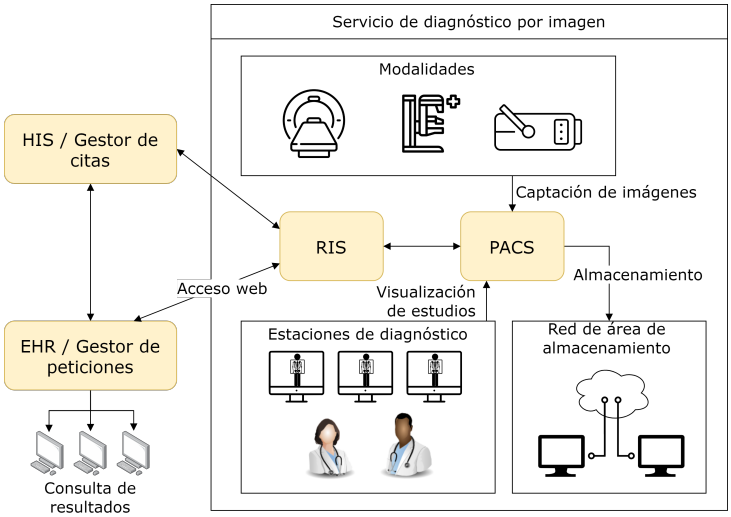


Figura 2.8: Integración de sistemas y equipos para la gestión de imágenes médicas.

Manufacturers Association o NEMA). Las versiones 1.0 y 2.0 de este conjunto de reglas se limitaron a especificar un modelo de red de comunicación entre iguales (*peer-to-peer*) para llevar a cabo la transmisión de imágenes médicas. Fue la versión 3.0, la que se conoce como DICOM, la que, además de mantener la interoperabilidad con los anteriores modelos de comunicación, fomentó el intercambio de imágenes médicas a través de redes estándar [67].

Desde su lanzamiento en 1993, grandes proveedores de equipos de electromedicina como AGFA [68], General Electric [69] y Siemens [70] lo han adoptado de manera general, y hoy en día es uno de los estándares de mensajería sanitaria más extendidos en el mundo. En la actualidad, existen cientos de miles de dispositivos de imagen médica y millones de imágenes DICOM para la atención sanitaria [71]. El estándar DICOM sigue estando respaldado y mantenido por NEMA, y está reconocido por la organización internacional para la normalización (*International Organization for Standardization* o ISO) como el estándar ISO 12052.

2.4.1. Formato de almacenamiento DICOM

Una imagen médica por sí misma no aporta suficiente información. Para que sea correctamente interpretada es necesario que vaya acompañada de datos adicionales, incluyendo información del paciente, del estudio realizado y del equipo radiológico y magnitudes dosimétricas. Por esto, formatos de imágenes tradicionales como JPEG o PNG fallan en dar respuesta a las necesidades concretas del ámbito de las imágenes médicas.

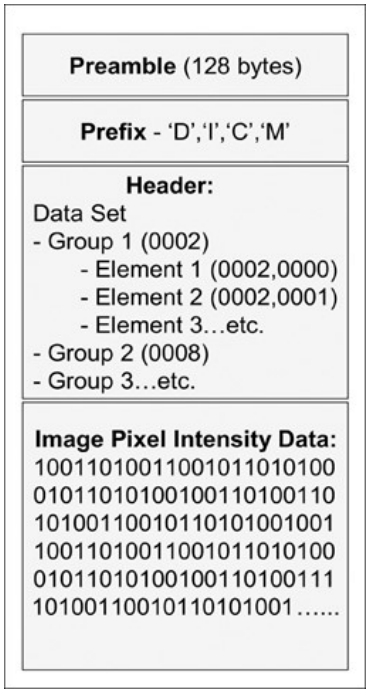


Figura 2.9: Estructura de un archivo DICOM.

La ventaja del estándar DICOM es que toda esta información relevante se reúne en un único fichero, en el cual los datos se definen mediante un modelo que intenta reflejar el mundo real [72]. El núcleo de información de un archivo DICOM es la imagen, pero también contiene información organizada por etiquetas para que los expertos sanitarios tengan todo lo que necesitan para entender el relato que la imagen está tratando de transmitir: información sobre el paciente como datos demográficos (*Patient's Sex*) y de identificación (*Patient's Name*), el estudio en el que se encuadra la toma de la imagen (*Study Instance UID*), la serie a la que pertenece la imagen (*Series Instance UID*), el dispositivo donde ha sido obtenida (*Device UID*), etc.

Un objeto DICOM puede contener imágenes con múltiples fotogramas, y la imagen puede estar comprimida usando gran variedad de estándares, incluidos JPEG, JPEG Lossless, JPEG 2000, LZW y *Run Length Encoding* (RLE). La estructura básica de un archivo DICOM (Figura 2.9) consiste en una parte de cabecera (*Header*) y una parte de conjuntos de datos de imagen (*Data Set*), que a su vez está compuesto por una serie de elementos (*Data Elements*).

En la Figura 2.10 se muestran los elementos de un archivo DICOM y se observa que cada *DataElement* contiene un elemento etiqueta (*DataElement Tag*) que lo identifica. Estas etiquetas, numeradas con un identificador único y ordenadas dentro del archivo DICOM de menor a mayor, se describen del siguiente modo:

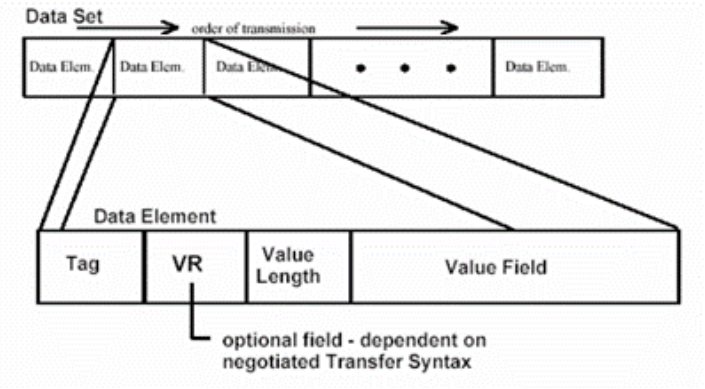


Figura 2.10: Elementos de un archivo DICOM.

- ***Tag*:** etiqueta de identificación.
- ***Value Representation* (VR):** valor de representación que indica el tipo de dato almacenado.
- ***Value Length*:** indica la longitud del dato.
- ***Value Field*:** contiene el dato en sí.

Debido al formato especial de los ficheros DICOM, es necesario disponer de un visor capaz de interpretar las informaciones del archivo y mostrarlas como una imagen. Existen visores DICOM en línea de código abierto que pueden ser utilizados libremente como OHIF Viewer [73] o Micro Dicom [74], si bien su utilidad profesional suele ser limitada comparada con los visores comerciales, normalmente de pago y código propietario como OsiriX DICOM Viewer [75], PostDICOM [76] y Sante DICOM Viewer [77].

En la última década, el tamaño de los datos digitales manejados por los servicios sanitarios ha aumentado drásticamente. El volumen de imágenes médicas que se generan es cada vez más alto y se ha observado que las necesidades de almacenamiento de los PACS crecen entre un 20 % y un 40 % al año [78]. Por ello, los sistemas software y hardware del pasado pueden ser insuficientes para afrontar las nuevas necesidades impuestas por el actual volumen de datos. Por ejemplo, la eficacia de la recuperación de información de los PACS ante un fallo se deteriora al afrontar cargas de trabajo elevadas.

Una posible solución para afrontar este problema es trasladar los sistemas PACS a la nube, donde se alojarían en servidores gestionados por proveedores y los hospitales delegarían gran parte del trabajo de mantenimiento de hardware [79]. Otra ventaja es que se facilitaría la interoperabilidad entre los sistemas informáticos de los hospitales e incluso entre organismos de salud [80].

En este sentido, pensando en situaciones en las que el tiempo de respuesta puede ser crítico y los recursos para disponer de grandes PACS son limitados (por ejemplo, pequeñas clínicas privadas), es importante obtener el máximo rendimiento de los recursos de los que se dispone para dar la mejor calidad de servicio. Tanto para diseños enfocados en la nube, como para los tradicionales basados en recursos físicos, los avances en el hardware de altas prestaciones son clave para poder responder a los retos que suponen estos escenarios.

2.5. Arquitecturas de altas prestaciones

La HPC es un ámbito de trabajo relacionado con todas las facetas de la tecnología, la metodología y la aplicación asociados a la obtención de la máxima capacidad de cálculo posible para cualquier momento y tecnología [81]. Se puede encontrar en una amplia gama de sistemas, desde ordenadores de sobremesa comunes hasta los grandes sistemas de procesamiento paralelo [82]. Los dispositivos electrónicos digitales empleados son denominados "supercomputadores" y su propósito es el de completar una gran cantidad de problemas computacionales lo más rápido posible.

La mayoría de los sistemas de alto rendimiento se basan en procesadores de conjuntos de instrucciones reducidos (*Reduced Instruction Set Computing* o RISC) [83], basados en pocas y simples instrucciones que generalmente se ejecutan en un único ciclo, lo que permite acelerar al máximo los ciclos de instrucción. Estos procesadores están diseñados para insertarse fácilmente en un sistema de múltiples procesadores que comparten sistema operativo y memoria, y utilizan el procesamiento informático conocido como el multiprocesamiento simétrico (*Symmetric Multi-Processing* o SMP) para acceder a una única memoria.

Estas máquinas son especialmente útiles para resolver problemas que requieren cargas de trabajo muy grandes, como por ejemplo, el procesamiento de las DNNs. Las DNNs tienen una elevada complejidad computacional y hasta el día de hoy han dependido de los motores de propósito general, especialmente las GPUs, para su procesamiento [84]. Su amplio uso ha despertado el interés en estudiar técnicas que permitan un procesamiento eficiente de estas redes, con el fin de mejorar la eficiencia energética y el rendimiento sin deteriorar la precisión.

Los sistemas integrados basados en placas NVIDIA Jetson son ejemplos de arquitecturas especializadas para conseguir un bajo consumo energético y una alta precisión [85]. Estas arquitecturas se caracterizan porque los distintos elementos de computación que componen la placa, una CPU y una GPU, comparten el mismo bus de memoria. Además, poseen un amplio rango de escalado de frecuencia, lo que les permite aumentar la frecuencia de los procesadores para mejorar el rendimiento del sistema. Estas características, así como su bajo consumo de energía, los convierten en opciones de alto rendimiento fundamentales para las DNNs, aunque en algún caso los recursos computacionales y de almacenamiento estén limitados.

Capítulo 3

Desarrollo del servidor de inferencia

Este capítulo describe la arquitectura global del sistema desarrollado e implantado en el HGUCS. Para ello, se detalla cada módulo de la arquitectura: el servidor intermedio construido para la captura de las imágenes radiológicas, el servidor de inferencia, describiendo su funcionamiento interno y la configuración aplicada en este proyecto, y el diseño de la DB donde se almacenarán los resultados de inferencia y que permitirá la comparativa con el personal experto.

3.1. Diseño del servidor de inferencia

La arquitectura sirve de punto de partida en una aplicación y es lo que define la estructura, la escalabilidad, el comportamiento y demás características del sistema. Por eso, es imprescindible realizar un diseño que tenga en cuenta no sólo las necesidades actuales, sino también las futuras. Los factores claves que se han tenido en cuenta a la hora de diseñar el sistema son los siguientes:

Portabilidad: La implantación del sistema en el hospital, su posible futura expansión a otros centros en los que las necesidades y recursos serán inevitablemente diferentes, y los experimentos de carga previstos en el proyecto, requieren la instalación de la arquitectura en un mínimo de cinco máquinas diferentes. Por ello, la primera decisión tomada ha sido la de crear una imagen Docker que albergue la arquitectura completa.

Esto ha conllevado dos grandes ventajas: por un lado, se evitan problemas entre los escenarios de desarrollo y producción y, además, la gestión de actualizaciones es más fiable, segura y rápida. Por otro lado, el hecho de empaquetar en contenedores garantiza una capa de abstracción frente al hardware en el que se ejecuta, puesto que la misma imagen Docker con Triton puede ser desplegada igualmente en un sistema operativo CentOS,

Ubuntu, o Windows.

Seguridad y cifrado: Las imágenes médicas son datos de salud, considerados sensibles y protegidos especialmente por la ley. Para garantizar la confidencialidad, es necesario utilizar comunicaciones cifradas entre todos los componentes de la arquitectura. Triton no soporta la versión cifrada de HTTP, el protocolo seguro de transferencia de hipertexto (*Hypertext Transfer Protocol Secure* o HTTPS), por lo que se ha utilizado el protocolo gRPC bajo TLS 2.0.

Interoperabilidad: El núcleo de Triton no soporta las comunicaciones vía estándar DICOM, lo que obliga a incluir en la arquitectura un componente que haga de puente entre los equipos de RX del HGUCS y el servidor de Triton.

Validación doble ciego: El modelo AI desplegado para la inferencia radiodiagnóstica se encuentra en fase de validación. El sistema debe ser capaz de almacenar el resultado de las inferencias que va generando el modelo para, en un futuro, comparar los diagnósticos del modelo con los realizados por un experto y así medir su tasa de acierto.

Bajo estas premisas, se ha diseñado una arquitectura en la que intervienen los siguientes actores:

1. **Equipo de RX:** aparato médico del servicio de urgencias que obtiene las imágenes radiológicas.
2. **Middleware:** servidor que comunica los aparatos de RX con Triton.
3. **NVIDIA Triton:** servidor que realiza la inferencia.
4. **MariaDB:** DB relacional que almacena los resultados de inferencia.

La Figura 3.1 ilustra el diseño de la arquitectura del sistema y el comportamiento más básico del sistema. Los actores interactúan entre ellos siguiendo este flujo de trabajo:

1. El middleware implementa un servidor de intercambio de DICOMs y queda a la escucha en el puerto 11112 (puerto DICOM). Cuando alguno de los equipos de RX del hospital obtiene una imagen nueva, envía el DICOM creado tanto al PACS del HGUCS como a este middleware.
2. En el middleware, se extrae la imagen y se preprocesa, para que los datos que forman la imagen concuerden con el tipo de dato y dimensiones que espera el modelo desplegado en Triton.

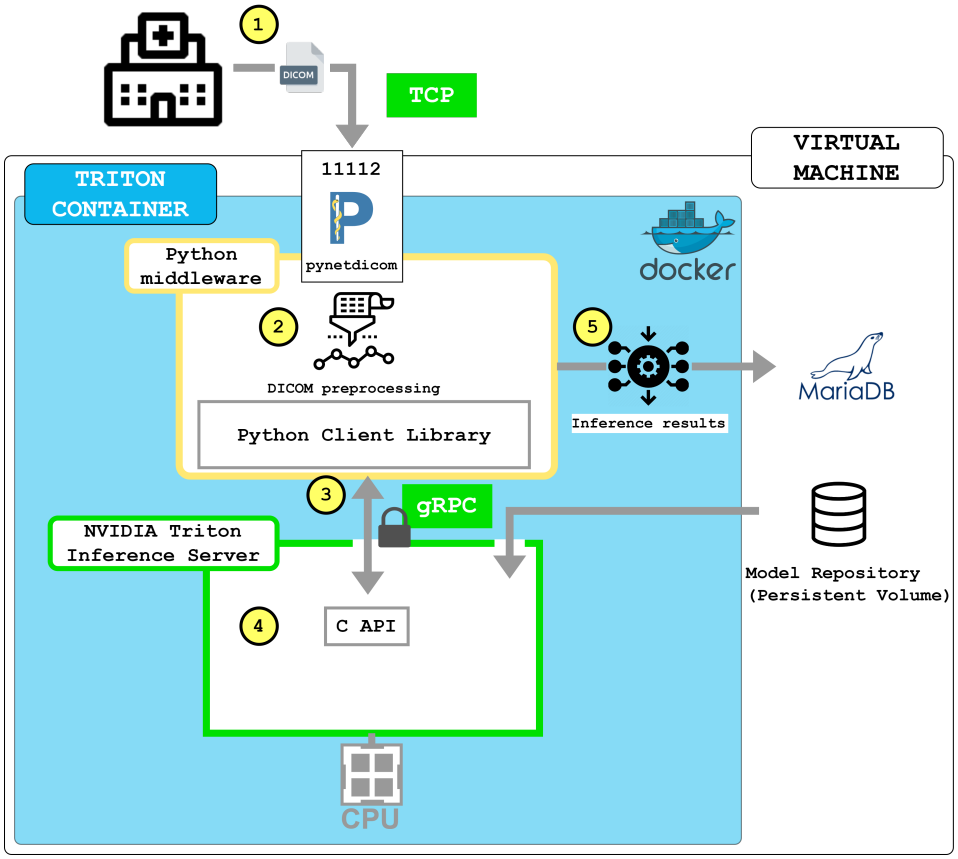


Figura 3.1: Diseño de la arquitectura del sistema.

3. Se abre un canal seguro utilizando el protocolo gRPC entre el middleware y el núcleo de Triton, por el que se envía la imagen preprocesada y se especifica con qué modelo se quiere hacer inferencia.

4. El núcleo de Triton recibe la petición, la procesa, hace inferencia y devuelve los resultados al middleware.

5. El middleware guarda los resultados en MariaDB.

6. Se responde mediante un simple *Acknowledgement DICOM* al equipo de RX del HGUCS que envió la imagen para dar por finalizada la transacción.

3.2. Middleware

El middleware consiste en un programa servidor implementado en Python. Se ha seleccionado el lenguaje de programación de Python por la simplicidad de su sintaxis, que facilita y acelera el desarrollo software. Es también uno de los lenguajes de programación más utilizados y con una comunidad de desarrolladores inmensa, lo que puede ayudar como soporte en el desarrollo. Además, los modelos de predicción CNN y VT han sido implementados utilizando las bibliotecas Python de TensorFlow y PyTorch, por lo que todo el desarrollo del proyecto puede realizarse en Python.

El propósito principal del middleware es el de recibir los archivos DICOM y transformar los datos para que puedan ser procesados por Triton. Para poder manejar los archivos DICOM, se hace uso de la biblioteca pydicom de Python, que permite leer, modificar y escribir datos de ficheros DICOM.

El servidor hace uso también de la biblioteca de Pynetdicom para implementar las comunicaciones necesarias. Permite permanecer a la escucha en un puerto en concreto cumpliendo el rol de proveedor de clases de servicio (*Service Class Provider* o SCP) del protocolo DICOM, es decir, su función es la de receptor de ficheros DICOM. Este hecho implica que el SCP no va a iniciar las comunicaciones, éste permanecerá a la espera hasta que otra máquina que implemente el protocolo DICOM y cuyo rol sea usuario de clases de servicio (*Service Class User* o SCU) lo contacte.

En el ámbito del proyecto RELIANCE, los modelos generados pueden inferir en RXT PA y AP, por lo que únicamente son interesantes los estudios que contienen este tipo de imágenes. Cada vez que llega un DICOM nuevo, se verifica si la imagen corresponde a un tórax PA o AP, comprobando las etiquetas de modalidad (CR o DX) y descripción. Aquellos estudios que hayan pasado las verificaciones son enviados al servidor de inferencia Triton. Para ello se utilizan los clientes *tritonclient* de Python, en concreto el módulo especializado para comunicaciones gRPC. Este mismo módulo permite lanzar la inferencia en Triton y recibir su respuesta.

Tras recibir la respuesta, el middleware abre un canal con la DB MariaDB, utilizando para ello un usuario con los permisos mínimos necesarios para ejecutar ciertos procedimientos almacenados. Haciendo uso de estos procedimientos almacenados, se insertan los datos recibidos en la DB. Una vez terminado, se cierra el canal. La información relevante que se ha decidido almacenar en la DB es la siguiente:

- **Accesion number**, obtenido de la etiqueta con la dirección (0008, 0050) del DICOM recibido. Este texto se utiliza como identificador único del estudio, y por lo tanto como clave primaria de la tabla.
- **IP del emisor**, se lee en el formato X.X.X.X, pero se transforma a un número de un

máximo de 11 dígitos cuando se almacena.

- **Resultados de la predicción del modelo de Triton**, cada etiqueta con el valor de su predicción.
- **Fecha en la que se recibió el estudio**.
- **Fecha en la que se terminó de procesar el estudio**, esta fecha indica el momento en el que se recibieron los resultados de la predicción de Triton.

La implementación del SCP del middleware también tiene la necesidad de responder al emisor. Tanto si el DICOM ha pasado el filtro y se ha podido procesar como si se ha descartado, el trabajo entre emisor y receptor ha sido completado satisfactoriamente, por lo que en ambos casos se devuelve un estado de éxito (código 0x0000 en las comunicaciones DICOM).

3.2.1. Configuración

Por cada modelo de AI distinto que se quiera desplegar en Triton, se crea una clase envoltorio equivalente en Python. Esta clase contiene todas las configuraciones específicas del modelo, como son su nombre, los datos acerca de sus entradas y salidas (nombre, tipo de dato y formato) y los pasos necesarios para preprocesar la imagen del DICOM. En el Listado 3.1 se puede consultar la clase envoltorio de la red CNN.

Gracias a este enfoque de programación orientada a objetos, es posible indicar al middleware cualquier número de modelos que se quiera levantar, siempre que éstos tengan su envoltorio definido. Al desplegarse por primera vez, éste cargará todos los objetos envoltorio equivalentes a los modelos.

La modalidad del SCP se ha configurado de forma que el middleware quede a la escucha en la dirección IP "0.0.0.0" y puerto 11112, con el *AE_Title* "*RELIANCE_DICOM*". El servidor también puede aceptar dos tipos de eventos, *C_ECHO* y *C_STORE*:

- ***C_ECHO*** es el equivalente DICOM al *ping*, empleado para que otros dispositivos que utilicen comunicaciones DICOM puedan comprobar que el middleware está activo. Se ha habilitado el contexto *Verification* para que el protocolo DICOM acepte estas conexiones cuando negocia con el SCU.
- ***C_STORE*** es la instrucción que se debe utilizar para que un SCU externo haga llegar al middleware un fichero DICOM, y se ha habilitado el contexto *AllStoragePresentation-Contexts* para que esto sea posible.

```
    import pathlib
    from torax.TensorFlow.tf_ds_preprocessor import load_data_from_dicom

    class ToraxTensorFlowModel:
        model_name = "torax_model"
        input_name = "input"
        input_data_type = "FP32"
        output_name = "MultiLabelSmartResizeDenseNet"
        output_data_type = "FP32"

        def __init__(self):
            self.__read_mean_std()

        def __read_mean_std(self):
            """Reads the file containing configuration values for the model."""
            normalize_mean_file=pathlib.Path(pathlib.Path(__file__).parent.
    resolve().as_posix(),"means/dataset_mean.txt")
            # Mean and std reading
            self._mean = 1.0
            self._std = 1.0
            with open(normalize_mean_file, "r") as f:
                self._mean = float(f.readline())
                self._std = float(f.readline())
       def preprocess_image(self, dicom):
            return load_data_from_dicom(dicom, self._mean, self._std)

```

Listado 3.1: Clase envoltorio del modelo de la CNN.

El Listado 3.2 muestra cómo se cargan los objetos envoltorio de los modelos, configuraciones básicas del servidor y se inicializa el servidor. La conexión con el núcleo de Triton se realiza mediante el módulo *tritonclient.grpc*, el cual es parte de las bibliotecas que ofrece NVIDIA para que los clientes puedan comunicarse con el servidor de Triton. Se crea un objeto de tipo *InferenceServerClient* y se lanza una inferencia a partir de la función incorporada *infer*. Dado que se debe cifrar la conexión con TLS, hay que especificar la clave pública de la autoridad de certificación (*Certification Authority* o CA), la clave privada del middleware y la clave pública del middleware en la creación de este objeto *InferenceServerClient*.

Las transacciones con la DB de MariaDB, se implementan utilizando el conector para bases de datos de tipo MySQL *mysql.connector*. En el Listado 3.3 se muestra la implementación de la función que recibe los datos preprocesados y el objeto envoltorio del modelo y se comunica con Triton para lanzar la inferencia.

3.3. NVIDIA Triton

El servidor de inferencia Triton soporta una amplia variedad de entornos de desarrollo, optimizaciones y configuraciones. Esto es posible porque su arquitectura de alto nivel, ilustrada en la Figura 3.2, se compone de componentes software más pequeños y especializados que ofrecen una mayor personalización independiente. Los elementos que intervienen en la arquitectura de Triton son los siguientes:

```
    def main():
        models = [ToraxTensorFlowModel(), ToraxPyTorchModel(),
    ToraxEnsembleModel()]
        labels = read_labels_file()
        modality = Modality(('0.0.0.0', 11112), 'RELIANCE_DICOM')
        contexts = AllStoragePresentationContexts
        contexts.append(build_context(Verification))
        handlers = [
            (evt.EVT_CONN_OPEN, handle_open),
            (evt.EVT_C_STORE, handle_store, [models, labels]),
            (evt.EVT_C_ECHO, handle_echo)
        ]
        with ServerInit(modality, contexts, handlers) as HospitalServer:
            signal.signal(signal.SIGINT, close_hospital_server)
            signal.pause()
     if __name__ == '__main__':
        main()

```

Listado 3.2: Inicializar servidor NVIDIA Triton.

- **Puntos de entrada (endpoints):** las peticiones para hacer inferencia pueden llegar por cualquiera de estas tres vías: HTTP, gRPC, y la API de C. Además, los modelos manejados por Triton pueden ser consultados y controlados por una API de gestión de modelos dedicada, disponible por estas mismas tres vías.

- **Repositorio de modelos:** directorio bajo el que se guardan todos los modelos que va a manejar el servidor. Debe seguir una estructura de archivos concreta que varía dependiendo del entorno en el que el modelo está desarrollado.

- **Modelos de inferencia:** pueden ser con estado (tipo *stateful*) y sin estado (tipo *stateless*). Los modelos para la clasificación de imágenes y detección de objetos (los dos de este proyecto) son *stateless*. Una de las características de los modelos sin estado es que cada inferencia realizada es independiente de todas las demás.

- **Planificadores de colas:** permiten gestionar la cola de las peticiones de inferencia que llegan a cada modelo del servidor. Cada modelo debe tener un planificador de cola, cuyo tipo depende a su vez del tipo de modelo asociado. En el caso de los modelos *stateless* objeto de este proyecto se pueden utilizar dos:

  - Planificador por defecto (*default scheduler*): distribuye las peticiones de inferencia entre las instancias de modelo configuradas.
  - Planificador dinámico (*dynamic batcher*): el servidor se ocupa de crear dinámicamente grupos de peticiones de inferencia (lotes) para después distribuir los grupos sobre las instancias de modelo configuradas. Esto normalmente da una productividad (inferencias/segundo) mayor que el planificado por defecto.

- **Entornos de desarrollo (*backends*):** un *backend* de Triton es la implementación que ejecuta un modelo, y habitualmente es una envoltura alrededor de un entorno de DL, como PyTorch, TensorFlow, TensorRT, ONNX Runtime u OpenVINO. Cada modelo debe

```
    import numpy as np
    import tritonclient.grpc
    import pathlib

    def infer_in_triton_tls(data, model, url="localhost:8001", verbose=False, labels=None):
        """Sends an inference request using grpc protocol to the Triton server. Communication is encrypted. Returns the output of the inference made by Triton.

        Arguments:
        data -- data that Triton will infer for
        model -- model object as it is exposed in Triton

        Keyword arguments:
        url -- url where Triton server is listening for grpc connections (default "localhost:8001")
        verbose -- bool to activate/deactivate verbose logging from the grpc connection (default False)
        labels -- list containing the labels for a prediction (default None)
        """
        root_certificates=pathlib.Path(pathlib.Path(__file__).parent.resolve().as_posix(),"certs/ca/ca-cert.pem")
        private_key=pathlib.Path(pathlib.Path(__file__).parent.resolve().as_posix(),"certs/client/client-key.pem")
        certificate_chain=pathlib.Path(pathlib.Path(__file__).parent.resolve().as_posix(),"certs/client/client-cert.pem")

        triton_client = tritonclient.grpc.InferenceServerClient(
            url=url, verbose=verbose, ssl=True,
            root_certificates=root_certificates,
            private_key=private_key,
            certificate_chain=certificate_chain)

        inputs = []
        outputs = []
        input_name = getattr(model, 'input_name')
        output_name = getattr(model, 'output_name')

        inputs.append(
            tritonclient.grpc.InferInput(input_name, data.shape, getattr(model, 'input_data_type')))
        outputs.append(tritonclient.grpc.InferRequestedOutput(output_name, class_count=0))
        # Initialize the data
        inputs[0].set_data_from_numpy(data)
        # Run the inference
        results = triton_client.infer(model_name=getattr(model, 'model_name'),
                                      inputs=inputs,
                                      outputs=outputs)
        output0_data = results.as_numpy(output_name)
        maxs = np.argmax(output0_data, axis=1)
        return build_response_json(output0_data[0], maxs[0], labels)

```

Listado 3.3: Comunicación con el núcleo de NVIDIA Triton.

estar asociado a un *backend*, y se especifica en la configuración del modelo mediante el ajuste "*backend*". Triton pone a disposición de los desarrolladores una API para poder implementar *backends* propios que realicen cualquier funcionalidad que se desee.

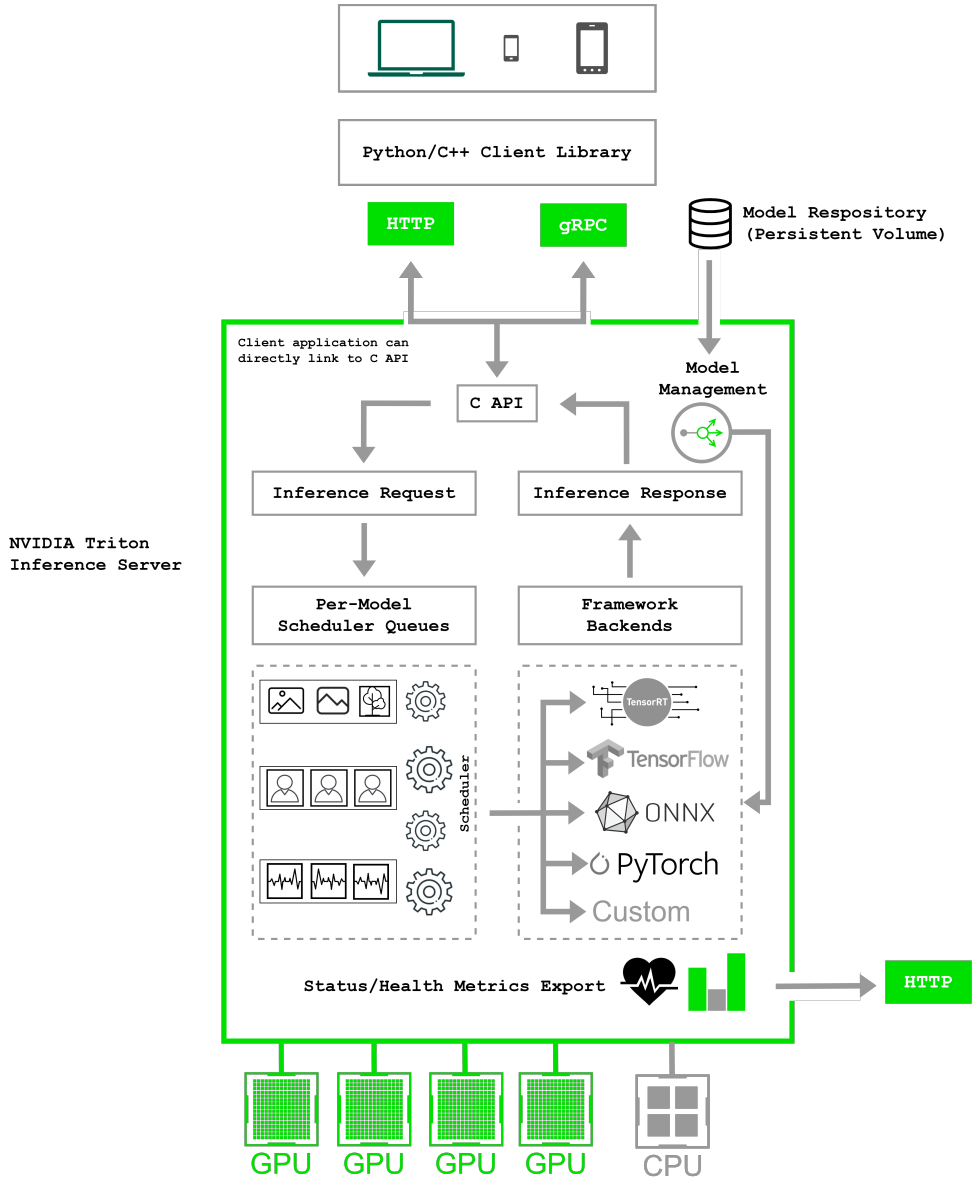


Figura 3.2: Arquitectura del servidor de inferencia Triton.

Las peticiones de inferencia llegan al servidor a través de una de las tres vías posibles, HTTP/REST, gRPC o la API de C, y se dirigen al planificador asociado a cada modelo. El planificador realiza opcionalmente la agrupación de las solicitudes de inferencia y pasa las solicitudes al *backend* correspondiente al tipo de modelo. El *backend* realiza la inferencia utilizando las entradas proporcionadas en las solicitudes por lotes para producir las salidas solicitadas. Finalmente, se devuelven los resultados inferidos al emisor.

La arquitectura de Triton permite que múltiples modelos y/o múltiples instancias del mismo modelo se ejecuten en paralelo en el sistema. Esto es especialmente útil en escenarios con modelos diferentes, o en los que haya más instancias de modelo que recursos GPU del sistema. A lo largo de las pruebas, esta opción se ha mantenido en una única instancia de modelo por red neuronal desplegada para acotar el abanico de pruebas a realizar. En el Anexo D se puede ampliar la información acerca de esta configuración.

3.3.1. Configuración

Conversión de modelos. El primer trabajo de configuración a realizar a nivel de Triton ha sido el de convertir los modelos de predicción desarrollados en los proyectos RADIANT y RELIANCE a formatos soportados por Triton:

El modelo CNN fue construido en TensorFlow 2 y almacenado en formato Keras, un único archivo con extensión *.h5*. Triton soporta el marco de aprendizaje TensorFlow, pero necesita que los modelos de TensorFlow sigan una estructura de datos jerárquica y estén en formato *SavedModel*.

El modelo VT fue implementado en PyTorch y guardado en dos partes: una clase de Python que define el esqueleto del modelo, y un archivo con extensión *.ckp* que define un punto de control general. Para desplegar este modelo en Triton, ha sido necesario convertirlo al formato TorchScript, formato recomendado por PyTorch para inferencia y despliegue a escala, y guardarlo como un archivo autocontenido de extensión *.pt*. Por temas de versiones de Pytorch soportadas por Triton 21.06-py3, se utilizó la versión de torch 1.10.1 y torchvision 0.11.1 para hacer la conversión del formato *ckp* a *pt*.

Configuración SSL: al no necesitar acreditación de CA externo, se optó por certificados auto firmados. Para generar los certificados se utiliza la herramienta OpenSSL, tal y como se muestra en el script de configuración de certificados que aparece en el Anexo C.

Control de modelo explícito: con este modo de gestión de modelos seleccionado para Triton es necesario especificar explícitamente cada uno de los modelos que estarán disponibles para uso de los clientes. En el HGUCS se ha desplegado únicamente la red CNN, mientras que para los experimentos de carga se han desplegado los modelos CNN y VT.

Configuración del modelo: por cada modelo que se quiera desplegar, es necesario crear un archivo de configuración que defina parámetros básicos del modelo. Un ejemplo de archivo de configuración es el Listado 3.4, que refleja la configuración del modelo CNN.

Esta configuración se debe guardar en una ubicación concreta, el repositorio de modelos, que contiene todos los modelos y sus archivos de configuración bajo una estructura de archivos

```
name: "torax_model"
platform: "tensorflow_savedmodel"
max_batch_size: 16
instance_group [ { count: 1 }]
input [
  {
    name: "input"
    data_type: TYPE_FP32
    dims: [ 1024, 1024, 1 ]
  }
]
output [
  {
    name: "MultiLabelSmartResizeDenseNet"
    data_type: TYPE_FP32
    dims: [ 8 ]
  }
]
version_policy: { latest { num_versions : 1 }}
optimization {
  graph { level: 1 }
}
```

Listado 3.4: Archivo de configuración CNN.

```
+---torax_model
|   |   config.pbtxt
|   |   labels.txt
|   |
|   \---1
|       \---model.savedmodel
|           |   keras_metadata.pb
|           |   saved_model.pb
|           |
|           \---variables
|                   variables.data-00000-of-00001
|                   variables.index
|
\---torax_pytorch
    |   config.pbtxt
    |   labels.txt
    |
    \---1
            model.pt
```

Figura 3.3: Estructura del repositorio de modelos.

concreta predefinida por Triton. En la Figura 3.3 se ilustra un repositorio que contiene un modelo en formato *SavedModel* de TensorFlow y uno en TorchScript de PyTorch. Asimismo, se dispone de un archivo *labels.txt* que define las etiquetas de salida.

3.4. MariaDB

La DB seleccionada para albergar los resultados de las inferencias ha sido MariaDB. MariaDB Server 10.5 es en la actualidad la principal DB relacional de código abierto. Su lista de características incluye:

- Tablas temporales, en tiempo de sistema y en tiempo de aplicación.
- Cambios instantáneos del esquema.
- Compatibilidad con PL/SQL y JSON.
- Ofuscación de datos y enmascaramiento total/parcial de dato.

El esquema relacional de la DB diseñada para el proyecto está ilustrada en la Figura 3.4. La tabla *Labels* contiene todas las etiquetas de salida posibles de las redes, *Models* almacena información acerca de las redes neuronales, *ToraxPredictions* alberga metadatos acerca de una predicción concreta y *Dicoms* solamente guarda el identificador único de la imagen DICOM, lo estrictamente necesario para poder identificar la imagen y mantener la confidencialidad de los datos médicos.

Las tablas *Labels* y *ToraxPredictions* tienen una relacionan de muchos a muchos, puesto que una predicción concreta puede tener más de una etiqueta de salida y viceversa. La tabla *ToraxPredictionsLabels* se crea a partir de esta relación con el objetivo de guardar los valores inferidos de cada etiqueta para las distintas predicciones. Cada vez que el servidor Triton desplegado en el HGUCS infiere una imagen médica, añade una entrada a la tabla *ToraxPredictions* y tantas entradas como etiquetas de salida tenga el resultado en *ToraxPredictionsLabels*. También se introduce el identificador de la imagen en *Dicoms*.

Las tablas *Users*, *RadiologistPredictions* y *RadiologistPredictionsLabels* son utilizadas para guardar datos acerca de la validación doble ciego. *Users* almacena los usuarios de las personas expertas en radiología y *RadiologistPredictions* es equivalente a *ToraxPredictions* y contiene metadatos del diagnóstico de usuarios. *RadiologistPredictionsLabels* es la tabla intermedia que se genera como consecuencia de la relación de las dos anteriores.

Además, con el objetivo de garantizar la confidencialidad requerida por el proyecto, el registro en la DB se restringe a procedimientos almacenados manejados únicamente por el middleware. Los scripts de creación de los procedimientos almacenados pueden consultarse en el Anexo B.

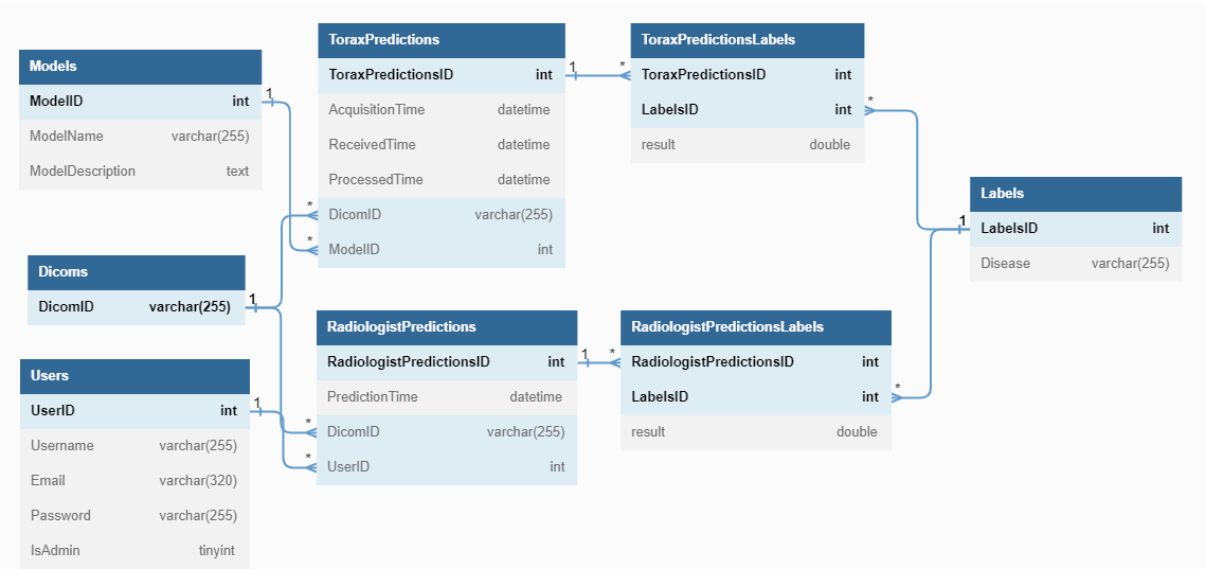


Figura 3.4: Esquema relacional de la DB.

Capítulo 4

Resultados experimentales

En este capítulo, se especifican las características hardware y software de las máquinas utilizadas para los experimentos de carga, así como la herramienta desarrollada para automatizar su ejecución y el software desarrollado para extraer los mejores resultados en función de los datos recogidos. Asimismo, se presentan en detalle las características de las redes CNN y VT que ha manejado el servidor de inferencia Triton implementado en el presente proyecto. Por último, se analizan y muestran los resultados de los experimentos de carga, explicando las métricas recogidas y los parámetros que se han variado para realizar las pruebas.

4.1. Configuración hardware y software

4.1.1. Hardware

Las pruebas para medir el rendimiento de las redes en Triton se han realizado en cuatro máquinas diferentes, un servidor de HPC equipado con aceleradores de tipo GPU y tres plataformas de bajo consumo basadas en sistemas NVIDIA Tegra Jetson, también equipadas con GPUs.

Hardware tradicional

La primera máquina donde se han realizado las pruebas ha sido un nodo computacional del clúster nasp de la UJI de Castellón que incluye 11 nodos, correspondiendo las pruebas a uno de los nodos equipado con 2 GPUs NVIDIA RTX A6000. Las características de esta GPU se recogen en la Tabla 4.1.

Tabla 4.1: Especificaciones de la GPU NVIDIA RTX A6000.

Características de la GPU	NVIDIA RTX A6000
Memoria GPU	48 GB GDDR6
Interfaz de memoria	384-bit
Ancho de banda de memoria	786 GB/s
Código de corrección de errores	Sí
Núcleos CUDA basados en la arquitectura NVIDIA Ampere	10.752
Tensor Cores de tercera generación de NVIDIA	336
RT Cores de segunda generación de NVIDIA	84
Rendimiento en precisión simple	38,7 TFLOPS
Rendimiento en RT Core	75,6 TFLOPS
Rendimiento Tensor	309,7 TFLOPS
NVIDIA NVLink	Conecta dos GPUs NVIDIA RTX A6000
Ancho de banda NVIDIA NVLink	112,5GB/s (bidireccional)
Interfaz del sistema	PCI Express 4.0 x16
Consumo de energía	Potencia total de la placa: 300 W

Hardware de bajo consumo

Adicionalmente al servidor anteriormente mencionado, y teniendo en cuenta las tendencias actuales de mercado, se ha trabajado con tres máquinas de altas prestaciones de la marca NVIDIA. De características diferentes en cuanto a potencia de cálculo y capacidades, comparten la particularidad de su pequeño tamaño y un consumo energético increíblemente bajo. Las especificaciones de las plataformas NVIDIA utilizadas se pueden consultar en la Tabla 4.2.

NVIDIA Jetson Xavier: compuesta por siete chips independientes: una CPU ARM de 8 núcleos, una GPU Volta, tres procesadores para gestionar la visión, el vídeo y la pantalla, y dos chips dedicados al DL. Está pensado para sistemas de AI de alto rendimiento, tales como dispositivos médicos portátiles, robots comerciales, cámaras inteligentes, sensores de alta resolución, drones, inspección óptica automatizada y otros sistemas integrados de IoT. Ofrece hasta 21 TOPS (Tera Operaciones Por Segundo) y 59,7 GB/s de ancho de banda de memoria con un consumo de sólo 30W. Admite múltiples modos de potencia y brinda hasta 14 operaciones de tensor por segundo para aplicaciones de AI en tan solo 10 W.

NVIDIA Jetson Nano: pensado para áreas de menor necesidad de computación que la GPU anterior, como pueden ser las grabadoras de vídeo de red (*networking video recorder* o NVR) básicos, robots domésticos, etc. Ofrece 472 GFLOPs para enfrentarse a los modernos algoritmos de AI, consumiendo solamente entre 5 y 10 W. Tiene la posibilidad de ejecutar varias redes neuronales en paralelo y procesar varios sensores de alta resolución simultáneamente.

NVIDIA Jetson Orin: mejora la capacidad de su predecesor Jetson Xavier sin renunciar

Tabla 4.2: Especificaciones de las plataformas NVIDIA Jetson seleccionadas.

Plataforma	Componente	Descripción
Orin	CPU	12×ARM Cortex-A78AE ARMv8.2 cores @ 2.20 GHz (grouped in 3 clusters)
	L1 Cache	64 KiB ICache; 64 KiB DCache (per core)
	L2 Cache	256 KiB (per core)
	L3 Cache	2048 KiB (per cluster)
	Memory	64 GiB LPDDR5 @ 204.8 GB/s
Xavier	CPU	8×NVIDIA Carmel ARMv8.2 cores @ 2.26 GHz (grouped in 4 clusters)
	L1 cache	128 KiB ICache; 64 KiB DCache (per core)
	L2 cache	2048 KiB (per cluster)
	L3 cache	4096 KiB (total)
	Memory	32 GiB LPDDR4 @ 137 GB/s
Nano	CPU	4×ARM Cortex-A57 ARMv8-A cores @ 1.43 GHz
	L1 cache	48 KiB ICache; 32 KiB DCache (per core)
	L2 cache	2048 KiB (total)
	Memory	4 GiB LPDDR4 @ 25.6 GB/s

a un consumo bajo y optimizado, máximo de 60W. Su rendimiento puede llegar a ser 8 veces superior, alcanzando los 275 TOPS en la versión de 64 GB de memoria frente a los 30 TOPS de la versión de 32 GB de la Xavier. Es el primer modelo que deja atrás la tecnología Fast Ethernet (10/100/10000 MB) y se decanta por optimizar su soporte de conexión con múltiples sensores con Gigabit Ethernet (1/10 GB).

Las tres plataformas mencionadas permiten modificar la configuración de su CPU y GPU para ajustar su rendimiento y consumo de energía máximo. Cada una de ellas viene con varias configuraciones predeterminados, conocidas como modos NVP, que especifican distintos valores para parámetros como la frecuencia y el número de CPUs habilitadas, entre otros. En la Tabla 4.3 se muestran los modos utilizados en este proyecto.

El hardware de bajo consumo está creciendo y mejorando a gran velocidad y el software que gestiona su funcionamiento debe estar a su altura. Los avances en rendimiento general y reducción de costes energéticos del hardware pueden verse afectados negativamente por un software inadecuado.

4.1.2. Software

Los hardware de bajo consumo y el nodo del servidor nasp son plataformas heterogéneas por diseño, por lo que es difícil garantizar la compatibilidad entre ellas. Es por ello que, a la hora de elegir las versiones, se han priorizado las más recientes que la plataforma permitiese. En la Tabla 4.4 se pueden consultar las versiones y el software utilizado para los experimentos.

Tabla 4.3: Configuraciones de modos NVP seleccionados para las plataformas NVIDIA Jetson.

Plataforma	Modo NVP ID	Modo NVP	CPUs habilitadas	Consumo energético (W)	Frecuencia (GHz)
Orin	0	Max-N	12	–	2.20
	1	15W	4	15	1.11
	2	30W	8	30	1.72
	3	50W	12	50	1.49
Xavier	0	Max-N	8	–	2.26
	1	10W	2	10	1.20
	2	15W	4	15	1.20
	3	30W-All	8	30	1.20
Nano	0	Max-N	4	–	1.48
	1	5W	2	5	0.92

Tabla 4.4: Software utilizado en los experimentos de carga.

Característica	Nodo en nasp	Jetson Orin	Jetson Xavier	Jetson Nano
SO	Ubuntu 20.04.2	Ubuntu 20.04.5	Ubuntu 18.04.6	Ubuntu 18.04.5
Docker	20.10.7	20.10.12	20.10.7	20.10.2
NVIDIA JetPack	–	5.0.1	5.0.2	4.6.2
Triton Container	21.06	–	–	–
ML Container (l4t-ml)	–	r34.1.1-py3	r35.1.0-py3	r32.7.1-py3
Triton Inference Server	2.23.0		2.26.0	2.19.0
CUDA Toolkit	11.3	11.4	10.2.89	10.2.89
TensorRT	7.2.3.4	8.4.0.11	7.2.2	7.2.2
nvidia-smi	465.19.01	–	–	–

El servidor de inferencia Triton puede utilizarse en los módulos Jetson a partir de la versión 4.x del NVIDIA JetPack. En la documentación de Triton, siempre recomiendan desplegar el servidor utilizando los contenedores disponibles en catálogo NGC de NVIDIA, sin embargo, no existen imágenes Docker oficiales para módulos Jetson. La única manera oficial de instalar Triton para versiones de JetPack entre la 4.x y la 5.x es hacerlo desde el código fuente y directamente sobre la máquina destino.

En el caso concreto de este proyecto, las máquinas destino son propiedad de la UJI y compartidas por distintas líneas de trabajo. Con el objetivo de facilitar la limpieza de recursos de almacenamiento utilizada, se ha empaquetado el servidor Triton en contenedores Docker para las Jetson Orin, Xavier y Nano. Asumiendo que Docker y Git están instaladas, estos son los pasos para reproducir este procedimiento:

1. **Selección de la imagen base:** se requiere una imagen base compatible con la versión JetPack de Jetson. Las imágenes Docker seleccionadas contienen los entornos de desarrollo TensorFlow y Pytorch, entre otros marcos de trabajo de ML, instalados en un entorno con Python 3.

2. **Dockerfile para montaje del servidor:** se crea un archivo que, a partir de la imagen del paso anterior, instala dependencias, descarga el archivo empaquetado del servidor de inferencia Triton y lo desempaqueta y configura para que esté listo para ser desplegado. Las redes CNN y VT y los *scripts* de alto nivel necesarios para automatizar el lanzamiento de los experimentos de carga se descargan vía Git.

3. **Docker sobre GPU:** se ha instalado el entorno de ejecución *nvidia-container-runtime* para que los contenedores Docker puedan utilizar GPU además de CPU. Adicionalmente, ha sido necesario añadir ciertas bibliotecas para solventar errores del entorno de desarrollo detectados al lanzar la operación de inferencia.

4. **Capacidad de cálculo mínima soportada:** la capacidad de computo mínima admitida por Jetson Xavier y Nano son 7.2 y 5.3, respectivamente, por lo que ha sido necesario indicar estos valores a la hora de desplegar el núcleo de Triton.

Cliente sintético

Analizar el rendimiento del servidor Triton para distintas configuraciones de las redes neuronales desplegadas plantea el reto de arrancar dos aplicaciones independientes que deben trabajar conjuntamente. Concretamente, un cliente enviando peticiones de inferencia y un servidor que las recibe, procesa y devuelve el resultado.

Deben probarse distintas configuraciones de ambos actores, variando a la vez la carga de trabajo que generan/reciben y haciendo un barrido de los parámetros. El número de combinaciones de parámetros es tan alto que es inviable levantar cliente y servidor y probarlas todas manualmente. Por ello, se consideró imprescindible automatizar el lanzamiento de ambos software y obtener los resultados sin intervención humana.

En una primera instancia se optó por utilizar la herramienta *Model analyzer* de Triton [86]. Este software automatiza la gestión del despliegue, el barrido de distintas configuraciones de servidor y cliente, así como la generación de informes a partir de los resultados obtenidos. A su vez, utiliza internamente la herramienta *Performance analyzer* [87] para simular los clientes sintéticos, el cual se ha configurado con los siguientes parámetros:

- **Sincronicidad (*–sync*):** las llamadas API se han configurado síncronas, es decir, se lanzará el mismo número de hilos que clientes especificados por el rango de concurrencia.

- **Rango de concurrencia (*–concurrency-range*):** se emplea para variar el número de clientes a simular, depende de cada plataforma.
- **Distribución de peticiones (*–request-distribution*):** indica el intervalo de tiempo para enviar peticiones de inferencia al servidor, su valor se ha mantenido en "constante" para todas las plataformas probadas. Este valor significa que los clientes enviarán peticiones siempre que hayan recibido respuesta de la anterior.
- **Datos de entrada (*–input-data*):** especifica los datos de entrada que se envían al servidor, se ha definido de manera que los clientes envían todo ceros.
- **Dimensiones (*–shape*):** indica el nombre y tamaño de los datos de entrada que espera la red, para CNN 1 x 1024 x 1024, para VT 1024 x 1024 x 1.
- **Tamaño de lote (*-b*):** tamaño de lote (*batch*) generado en el cliente, se ha configurado para que cada cliente envíe una única inferencia por cada llamada a Triton.
- **Protocolo de comunicación (*-i*):** se utilizará el protocolo de comunicación gRPC para comunicarse con Triton.

Las primeras pruebas concluían en estancamiento al llegar a concurrencias en torno de 15 a 20 clientes. Teniendo en cuenta la potencia del hardware en el que se lanzaban, se esperaba que las redes admitieran concurrencias mucho mayores. Tras analizar el problema, se comprobó que únicamente se producía al utilizar *Model analyzer*, y funcionaba correctamente lanzando exclusivamente el *Performance analyzer*. Además, las plataformas Jetson todavía no admiten el uso de *Model analyzer*, ya que su soporte está en desarrollo y ello impedía probar plataformas de esta familia, que constituyen uno de los objetivos de este trabajo.

Esas razones motivaron a buscar una alternativa a la herramienta. El software seleccionado debía ofrecer las mismas funcionalidades básicas del predecesor, como el despliegue automático de Triton y el barrido de combinaciones de configuraciones tanto en servidor como cliente. Finalmente, se decidió desarrollar una serie de *scripts* bash de alto nivel que cubrieran la automatización de las pruebas de rendimiento y delegaran la gestión de clientes sintéticos en la herramienta *Performance analyzer*. El flujo de trabajo del software para realizar las mediciones se ha ilustrado en la Figura 4.1, y consta de los siguientes pasos:

1. Para cada experimento, se crea un archivo de configuración que contiene los parámetros de la red, y un hipervínculo de referencia al archivo que contiene la red neuronal. Los hipervínculos son preferibles frente a la opción de copiar la red por cada configuración, porque se ahorra espacio, cuestión importante ya que las redes pueden ser de gran tamaño.
2. Para cada archivo de configuración creado, se levanta una instancia Docker con un servidor Triton. El servidor lee el archivo de configuración, carga y despliega la red neuronal.

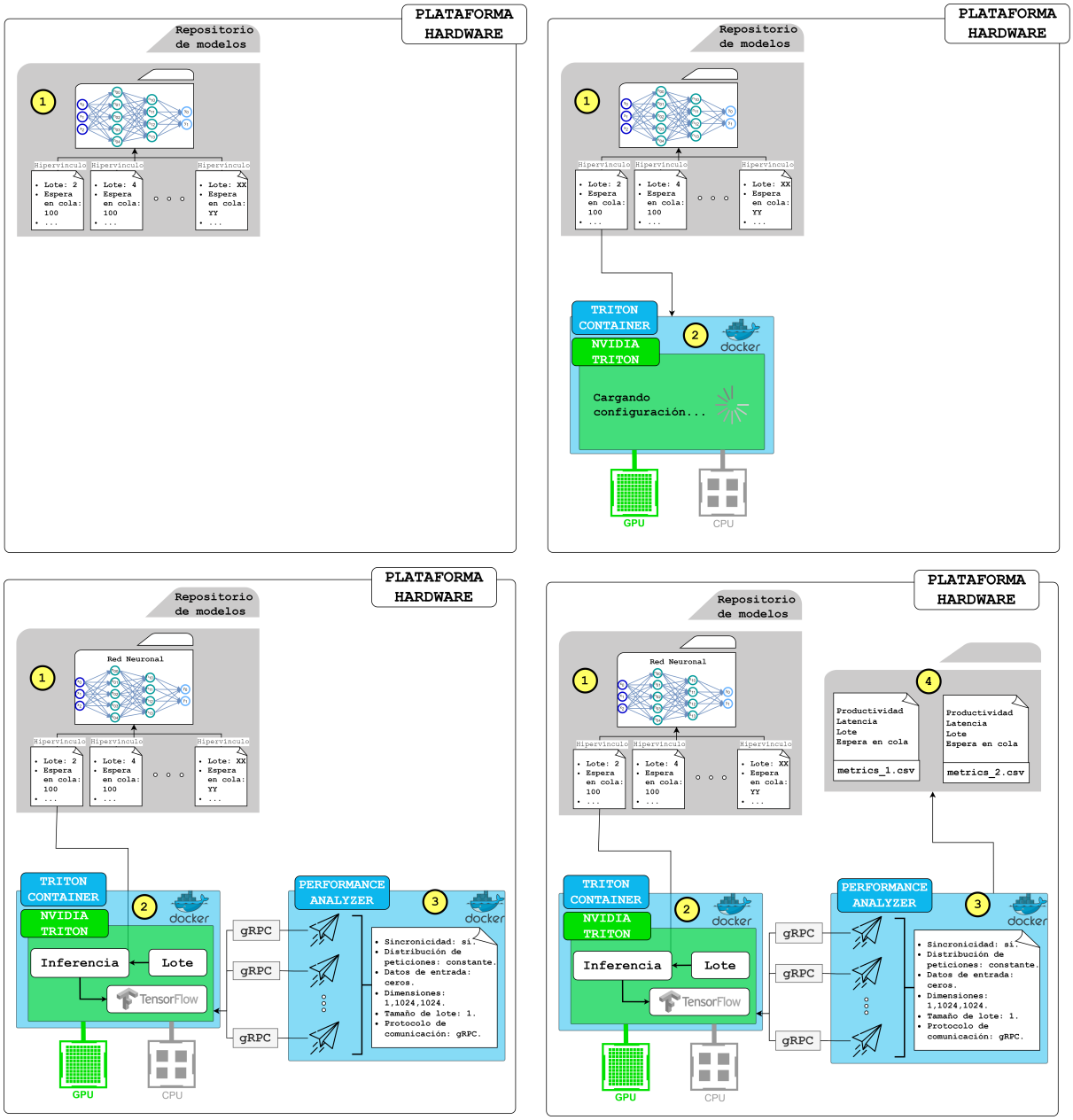


Figura 4.1: Flujo de trabajo del software para hacer las mediciones.

3. Una vez el servidor ha terminado de cargar las configuraciones, se levanta una instancia Docker que contiene el software *Performance analyzer*. Este software despliega tantos clientes sintéticos como concurrencia se haya especificado, que se conectan mediante el protocolo gRPC con Triton.

4. Mientras los clientes sintéticos realizan peticiones de inferencia, la herramienta mide y recoge métricas. Tras la finalización del barrido de concurrencias, se eliminan todos los clientes sintéticos, y las métricas recogidas en los experimentos se almacenan en archivos *CSV*.

5. Ambas instancias Docker, servidor y cliente, se eliminan.

6. Este proceso se repite hasta que se hayan procesado todos los archivos de configuración.

Clasificador de resultados

El lanzamiento de los test de carga da como resultado un fichero en formato *CSV* con multitud de métricas para varios escenarios de prueba. Se debe abordar, llegados a este punto, la problemática de discernir qué valores son los mejores y peores. El criterio de "mejor" o "peor" está condicionado por el escenario al que se refiere: en general, los servicios en línea en tiempo real requieren una baja latencia y los sistemas de procesamiento sin conexión priorizan una alta productividad.

Independientemente del escenario, el número de clientes concurrente que enfrenta el sistema es un factor decisivo. Configuraciones que produzcan un gran rendimiento con cargas de trabajo bajas pueden verse colapsados frente a avalanchas de peticiones. Lo mismo es posible al contrario, configuraciones que no destacan cuando la carga de trabajo es baja pueden rendir especialmente bien al afrontar grandes cantidades de clientes.

En resumen, muchas condiciones diferentes son las que determinan la idoneidad de unos resultados, y hacer este análisis manualmente para todas las mediciones es repetitivo y propenso a errores. Para facilitar esta labor se ha desarrollado un software en Python, aprovechando su biblioteca Pandas, que selecciona la mejor configuración de la red entre todas las obtenidas. Permite, además, hacerlo acotando rangos de concurrencia, plataformas hardware y seleccionando la red neuronal a tener en cuenta a la hora de hacer esta búsqueda, entre otras cosas.

Redes neuronales

Las dos redes neuronales utilizadas para los experimentos son una CNN y un VT. Específicamente, la red CNN es una red totalmente conectada de versión 169 (*DenseNet-169*) desarrollada utilizando el entorno TensorFlow. Por su parte, el VT es una red BEiT versión *Large* (*BEiT-Large*) que utiliza la función de activación ReLU y está implementada en el entorno PyTorch.

Dado que la *DenseNet-169* constituye únicamente la base de la red neuronal completa, se han añadido una serie de capas por encima para que la red final sea capaz de diagnosticar imágenes

médicas de tórax PA y AP. *DenseNet-169* utiliza una resolución de entrada de 512 x 512 con tres canales de color RGB, mientras que la red neuronal completa contiene aproximadamente 14 millones de parámetros y trabaja con una resolución de 1024 x 1024 y un sólo canal de color, es decir, en escala de grises. La dimensión de los datos de salida es de 8, lo que quiere decir que analiza los datos de entrada e infiere ocho diferentes etiquetas.

La red *BEiT-Large*, similar a la *DenseNet-169*, trabaja con una resolución de 512 x 512 y emplea tres canales de color RGB. También se han añadido capas de funciones sobre esta red base para crear la red completa con la que se va a experimentar. El *BEiT-Large* completo está formado por alrededor de 307 millones de parámetros, más de 21 veces el número de parámetros de la red anteriormente mencionada. Por lo demás, es similar a la *DenseNet-169*, y utiliza una resolución de 1024 x 1024 y un único canal de color, infiriendo ocho etiquetas de salida.

Ambas redes, *DenseNet-169* y *BEiT-Large*, experimentan un desajuste en las dimensiones de los datos de entrada de las redes completas en comparación con las redes base. En los dos casos el problema se soluciona con el mismo procedimiento, una de las capas añadidas aplica una convolución 2D para reducir las dimensiones de los datos de entrada y, de esta manera, ajustarlos para que concuerden con lo que las redes base esperan.

Las características más significativas de ambas redes neuronales se resumen en la Tabla 4.5. Las dimensiones (B, H, W, C) significan el tamaño de lote, la altura, la anchura y el número de canales de los datos de entrada respectivamente. Si bien el orden las dimensiones es diferente para las dos redes, los valores son los mismos. Por su parte, las dimensiones (B, E) se refieren al tamaño de lote y número de etiquetas de salida (un valor de -1 indica que se trata de una dimensión dinámica, y por lo tanto la red acepta cualquier valor para esa dimensión). El tamaño de lote es dinámico en las dos redes, por lo que ambas admiten inferir imágenes en grupo y con cualquier número de imágenes por grupo.

Tabla 4.5: Características de las redes neuronales del proyecto.

Red neuronal	Número de parámetros	Dimensiones	Etiquetas de salida
DenseNet-169	14 millones	(B, H, W, C) (-1, 1024, 1024, 1)	(B, E) (-1, 8)
BEiT-Large	307 millones	(B, C, H, W) (-1, 1, 1024, 1024)	(B, E) (-1, 8)

Cuantización de redes

Las plataformas Jetson, aunque especializadas para aplicaciones de inteligencia artificial, tienen la desventaja de poseer recursos limitados. Mientras que las Jetson Orin y Xavier no

presentan obstáculos adicionales, el almacenamiento es una limitación a la hora de incluir las redes de este proyecto en la Jetson Nano. Ninguna de las dos redes con las que se van a hacer los experimentos, *DenseNet-169* y *BEiT-Large*, caben en su memoria.

Para que el despliegue del servidor Triton en la Jetson Nano sea posible, las redes deben modificarse reduciendo sus requisitos de almacenamiento hasta que quepan en memoria. La solución adoptada para reducir el tamaño de las redes ha sido disminuir la precisión de los cálculos mediante la cuantización. Esta técnica puede realizarse en la fase de entrenamiento, sin embargo, lo ideal para este caso es llevarlo a cabo sobre las redes de las que se dispone, sin tener que reentrenarlas.

Las técnicas de cuantización posteriores al entrenamiento pueden ocasionar consecuencias negativas, pérdida de precisión en la mayoría de los casos. En primera instancia se aplicó uno de los métodos más agresivos, consistente en acotar los números de coma flotante a enteros. Al inspeccionar los resultados de las inferencias se confirmó que ambas redes sufrían una disminución considerable de precisión. La solución finalmente adoptada ha sido aplicar una técnica menos agresiva, como es la cuantización a float16.

La *DenseNet-169* no experimenta un impacto significativo en precisión empleando esta técnica, reduciendo además aproximadamente a la mitad su tamaño. Sin embargo, la *DenseNet-169* está desarrollada sobre el marco de trabajo TensorFlow y requiere ser transformada a TfLite para efectuar la cuantización. Esto introduce un nuevo reto, ya que el servidor Triton no soporta desplegar redes en el entorno TfLite. Por tanto, se decidió hacer una última transformación, convirtiendo la red ya cuantizada en TfLite a ONNX, un estándar para la interoperabilidad del aprendizaje automático que sí puede ser desplegado en Triton.

El *BEiT-Large* cuantizado a float16 también mantiene un rendimiento cercano a la red de partida en la Jetson Nano. No obstante, PyTorch, el entorno en el que está desarrollado este VT realiza una cuantización de los pesos pero mantiene el tipo de dato en float32, lo que resulta en que este tipo de cuantización no aporte ningún beneficio al tamaño de la red.

Con estas modificaciones, la *DenseNet-169* ocupa lo suficiente para entrar en la memoria de la Jetson Nano mientras que la red *BEiT-Large* sigue sin caber. Derivado de esto, los experimentos de carga sobre este hardware se limitan a realizar mediciones desplegando únicamente la red *DenseNet-169* en ONNX.

4.2. Experimentos de carga

El cliente sintético recoge una amplia variedad de métricas para proporcionar medidas objetivas del rendimiento del servidor de inferencia Triton. Estudiando estas medidas es posible

hacer una evaluación de su funcionamiento, sacar conclusiones y optimizar la configuración para obtener el máximo provecho del servidor. Los experimentos de carga de este trabajo han tenido en cuenta las siguientes métricas, todas ellas medidas desde el cliente *Performance analyzer*:

- **Concurrencia:** número de clientes simultáneos entre sí enviando peticiones de inferencia al servidor. Si bien los clientes ejecutan sus instrucciones de manera independiente, mantienen un comportamiento secuencial: cada cliente envía una petición de inferencia y se queda a la espera hasta recibir la respuesta, momento en el que envía una nueva petición.
- **Productividad:** número de inferencias por segundo que procesa el servidor. En escenarios en los que se habla de productividad, una configuración que reporte un valor mayor será preferible frente a otras menores.
- **Latencia (p99):** tiempo observado por el cliente desde que envía una petición de inferencia hasta que recibe su resultado. Puesto que se refiere al tiempo de espera del cliente, cuanto menor sea el valor de esta métrica, mejor se calificará la configuración. Se utiliza la latencia del percentil 99 (p99), lo que quiere decir que el 99 % de las peticiones serán más rápidas que el valor de esta métrica.

En lo que respecta al servidor, existe la posibilidad de configurarlo para agrupar peticiones de inferencia en lotes y lanzarlas todas juntas sobre la GPU. Anteriores evaluaciones de Triton concluyeron que hacerlo de esta manera resulta en una mayor productividad y una menor latencia comparado con hacer las inferencias de una en una [88]. Esta característica ha sido probada para ambas redes, *DenseNet-169* y *BEiT-Large*, variando los siguientes parámetros:

- **Tamaño máximo de lote:** número máximo de solicitudes de inferencia que se van a agrupar para ser procesadas en lote. Las peticiones se almacenan en cola hasta completar el lote, que es entonces ejecutado.
- **Tiempo máximo de espera en la cola:** puede haber casos en los que el lote no termine de llenarse y las solicitudes ya encoladas pueden quedarse a la espera demasiado tiempo. Este parámetro especifica el máximo tiempo medido en microsegundos que se demorará la inferencia del lote, y si el intervalo vence, se procesará tal y como está. El tiempo empieza a contar desde que se encola la primera solicitud, y se reinicia al procesar el lote.

Ambas opciones son definidas por números positivos reales y no tienen límite superior alguno, lo que deja un amplio rango de valores posibles. Realizar una búsqueda extensiva a la fuerza bruta no es viable, ya que las pruebas necesitan tiempo para estabilizarse y con tantas combinaciones de valores para medir se tardaría demasiado en completar todas ellas. Para

solventar este problema, se ha definido para cada plataforma una serie de combinaciones de parámetros que representan lo más precisamente los posibles escenarios, y de los cuales se extrae la mejor configuración.

Un factor a tener en cuenta a la hora de analizar los resultados es que las plataformas hardware utilizadas tienen especificaciones y recursos distintos, por lo que una serie de pruebas válidas para un escenario puede no serlo en otra máquina. A continuación, se muestran los resultados para las cuatro plataformas utilizadas donde los parámetros con los que se han lanzado las pruebas se explican en la introducción de su sección.

4.2.1. Nodo servidor nasp

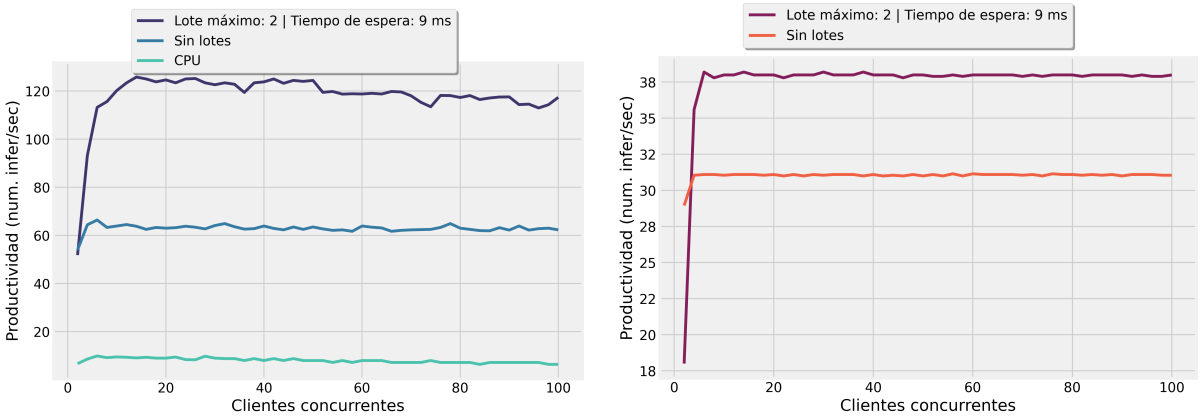
El nodo en el servidor nasp está equipado con dos GPUs NVIDIA RTX A6000 y durante los experimentos el servidor Triton ha utilizado ambas para procesar las peticiones de inferencia. En la Tabla 4.6 se muestran los valores de los parámetros que se han utilizado para definir los experimentos de carga en esta plataforma.

Tabla 4.6: Experimentos con *DenseNet-169* y *BEiT-Large* en el servidor nasp.

Red neuronal	Parámetros	Valores
DenseNet-169	Concurrencia	Rango [2, 100] y saltos de 2.
	Tamaño máximo de lote	[2, 8, 16, 32, 64, 96, 128]
	Tiempo máximo de espera en la cola (µs)	[1000, 10000, 100000, 1000000]
BEiT-Large	Concurrencia	Rango [2, 64] y saltos de 2.
	Tamaño máximo de lote	Rango [2, 64] y saltos de 2. 96 y 128 para mayor amplitud.
	Tiempo máximo de espera en la cola (µs)	[300, 3000, 6000, 9000, 15000]

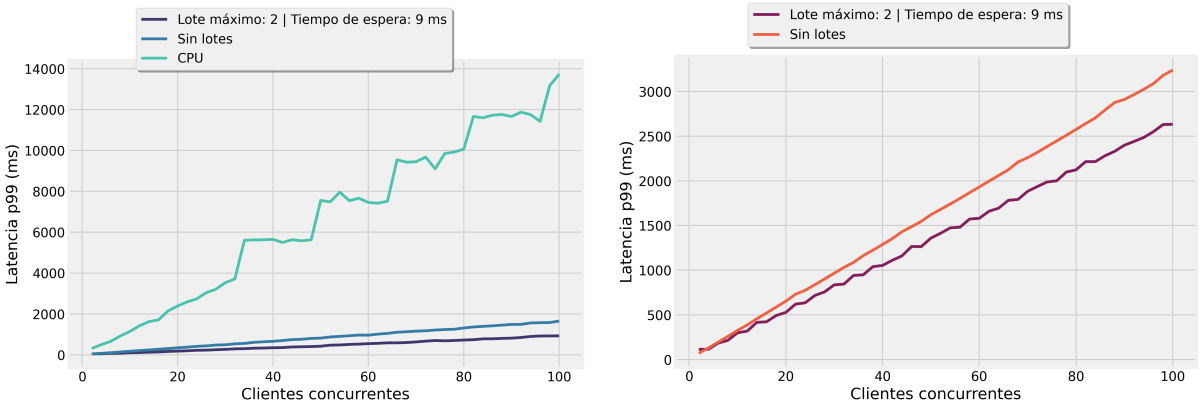
Las Figuras 4.2 y 4.3 muestran el número de inferencias por segundo y la latencia p99 respecto al número concurrente de clientes para las redes *DenseNet-169* y *BEiT-Large*. Para cada red neuronal se dibujan los resultados de rendimiento obtenidos con la mejor configuración sobre GPU agrupando lotes, sin agrupar lotes y sobre CPU. Sobre éste último no se muestran resultados del *BEiT-Large* puesto que pasados los 16 clientes concurrentes el servidor deja de funcionar y la productividad registrada hasta ese punto es del orden de 0,25.

La *DenseNet-169* proporciona, en el mejor de los casos, aproximadamente una productividad de 120 inferencias por segundo y una latencia menor que 0.5 segundos para todas las concurrencias. Por su parte, la red *BEiT-Large* registra una productividad máxima de 40 inferencias/segundo y una latencia aproximada de 2 segundos para las concurrencias cercanas a 100.



(a) Red *DenseNet-169*. (b) Red *BEiT-Large*.

Figura 4.2: Productividad por concurrencia en nodo nasp.



(a) Red *DenseNet-169*. (b) Red *BEiT-Large*.

Figura 4.3: Latencia por concurrencia en nodo nasp.

4.2.2. Plataforma Jetson Orin

Los parámetros que definen los experimentos en la plataforma Jetson Orin vienen reflejados en la Tabla 4.7. En este caso, se han utilizado valores diferentes para las redes *DenseNet-169* y *BEiT-Large*, porque tal y como se ha visto en los resultados del nodo de nasp, sostienen rendimientos muy diferentes y no tiene sentido someterlos a la misma carga de trabajo. Además, debido al gran tamaño que el *BEiT-Large* ocupa en memoria no se pueden emplear tamaños de lote tan grandes, ya que si se sobrepasa la cantidad de memoria disponible de la GPU el servidor puede dejar de funcionar.

Tabla 4.7: Experimentos con *DenseNet-169* y *BEiT-Large* en Orin.

Red neuronal	Parámetros	Valores
DenseNet-169	Concurrencia	Rango [2, 64] y saltos de 2.
	Tamaño máximo de lote	[2, 8, 16, 32, 64, 96, 128]
	Tiempo máximo de espera en la cola (µs)	[1000, 10000, 100000, 1000000]
BEiT-Large	Concurrencia	Rango [2, 64] y saltos de 2.
	Tamaño máximo de lote	[2, 4, 8, 16, 24]
	Tiempo máximo de espera en la cola (µs)	[1000, 10000, 100000, 1000000]

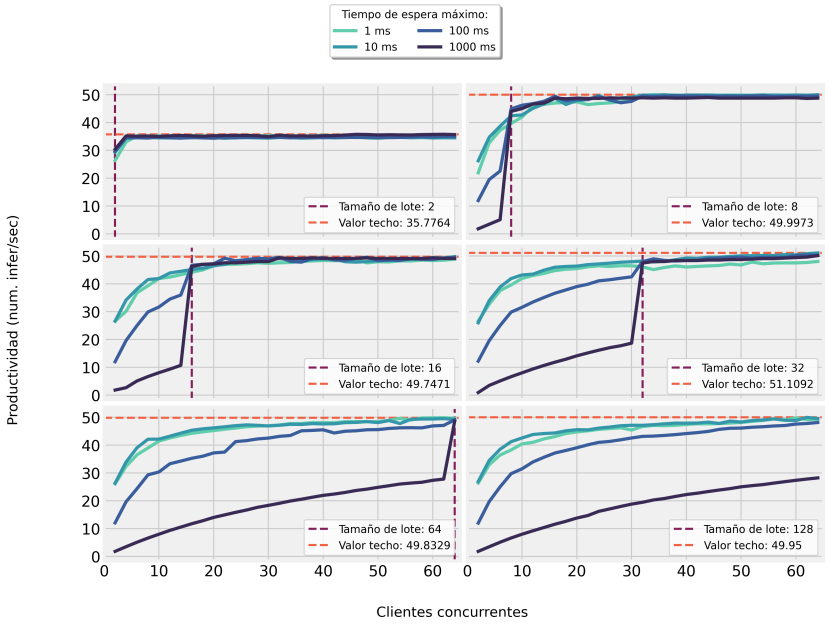


Figura 4.4: Jetson Orin productividad de *DenseNet-169* en modo NVP Max-N.

La Figura 4.4 indica la relación entre productividad y peticiones concurrentes a la red neuronal *DenseNet-169* cuando se utiliza el valor Max-N como modo NVP. El gráfico contiene todos los tamaños de lote mencionados, a excepción del 96, que no aporta información nueva y se ha descartado para que la figura pueda ser interpretada más fácilmente. Por su parte, la línea discontinua vertical delimita el valor de la concurrencia igual al tamaño máximo de lote del experimento, mientras que la horizontal es el valor máximo registrado en el eje Y.

El valor máximo de productividad más común es de 50 inferencias/segundo, aunque el primer gráfico de líneas referente al tamaño de lote 2 alcanza un valor bastante menor, justificado por el reducido tamaño del lote. Para el resto de casos, la clave es conocer para qué número de clientes recurrentes se alcanza el valor máximo.

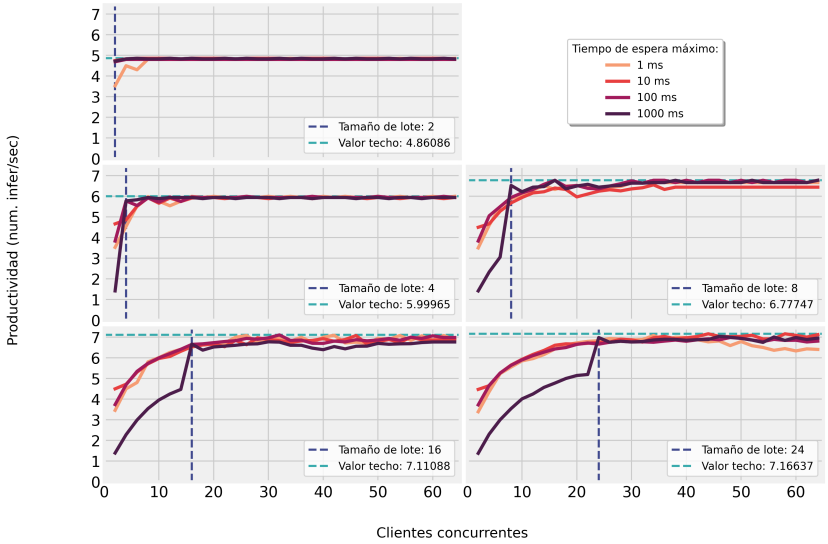


Figura 4.5: Jetson Orin productividad de *BEiT-Large* en modo NVP Max-N.

También se puede observar en la Figura 4.4 que la productividad en el tiempo de espera máximo más alto (1000 ms) queda notoriamente por debajo al resto de configuraciones cuando se pretende alcanzar una concurrencia equivalente al tamaño de lote, reflejado por la línea vertical en la figura. A partir de este punto, se produce un rápido crecimiento de su productividad hasta alcanzar los mismos valores que el resto de líneas. La razón por la que esta situación se produce es porque el número de peticiones concurrentes no rellenan el tamaño del lote y, por consiguiente, es necesario esperar a que el tiempo de espera venza para lanzar las inferencias. Otros tiempos de espera menores también siguen el mismo patrón, se puede observar en el de 100 ms, pero el crecimiento es menos pronunciado.

La Figura 4.5 expone la correlación en el modo NVP de Max-N entre la productividad y el número de peticiones concurrentes a la red neuronal *BEiT-Large*. Igual que en las figuras referentes a la *DenseNet-169*, la línea discontinua vertical muestra el tamaño de lote máximo del experimento y la horizontal el valor máximo registrado en el eje Y.

En el caso de la red *BEiT-Large* la productividad máxima es mucho menor, en torno a las 7 inferencias por segundo. Aún así, el comportamiento es parecido al de la red *DenseNet-169*, los tamaños de lote pequeños no consiguen llegar a este valor máximo y llegados a las 16 inferencias por lote todas las distintas configuraciones lo alcanzan.

En cuanto a los rápidos picos de subida que se han comentado con anterioridad, los datos recogidos para esta red también los reportan. Igual que ocurría con la red *DenseNet-169*, se producen cuando la concurrencia iguala el valor del tamaño de lote máximo.

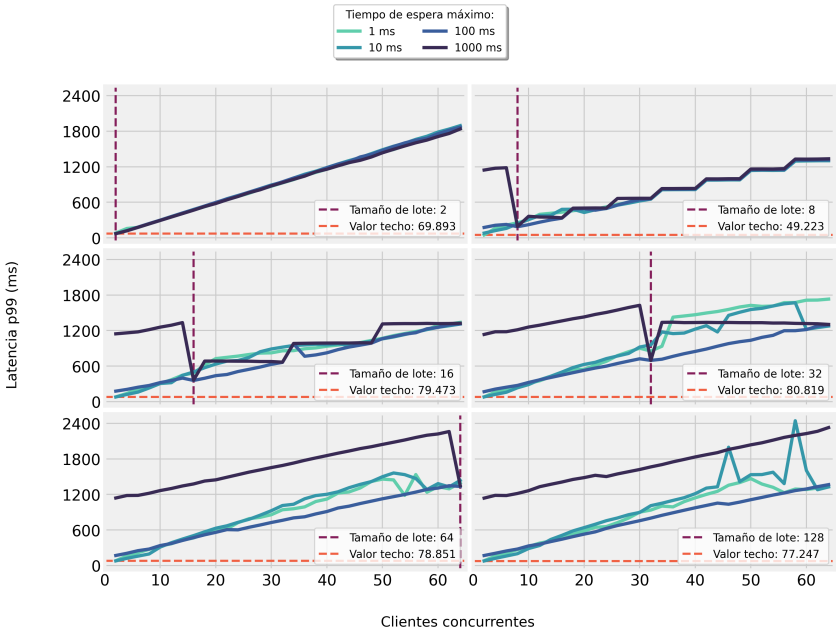


Figura 4.6: Jetson ORIN latencia de *DenseNet-169* en modo NVP Max-N.

Los datos mostrados en las Figuras 4.6 y 4.7 muestran los mismos experimentos realizados sobre las redes *DenseNet-169* y *BEiT-Large* que los de 4.4 y 4.5, respectivamente. Por tanto, ilustran la correlación entre la latencia p99 y el número de peticiones concurrentes, manteniendo el hardware configurado en el modo NVP de Max-N.

La Figura 4.6 también evidencia un comportamiento similar en cuanto a latencia y productividad, aunque los tiempos de espera más altos muestran resultados inferiores al reportar latencias mayores. En este caso, cuando se alcanza el mismo valor de concurrencia que de tamaño de lote máximo se experiencia un decrecimiento repentino.

Por su parte, la latencia del percentil 99 de ambas redes aumenta linealmente, ya que las líneas de las gráficas mantienen una tasa de cambio constante. Esto es especialmente visible en la Figura 4.7 donde los tamaños de lote máximo más bajos muestran líneas casi rectas, pero se produce un crecimiento escalonado al aumentar el tamaño del lote.

A través del programa clasificador de resultados se ha podido determinar que la mejor combinación de parámetros está compuesta por un tamaño de lote de 16 y un tiempo de espera máximo de 100 ms para la *DenseNet-169*. Por su parte, el *BEiT-Large* logra su máximo rendimiento cuando se utiliza la misma configuración, un tamaño de lote de 16 y un tiempo de espera máximo de 100 ms. Ambas configuraciones se refieren al modo NVP Max-N. El uso de otros modos NVP obtiene diferentes valores referencia, alterando escasamente la tendencia de las líneas de datos para las dos métricas que se han tenido en cuenta, la latencia y la productividad.

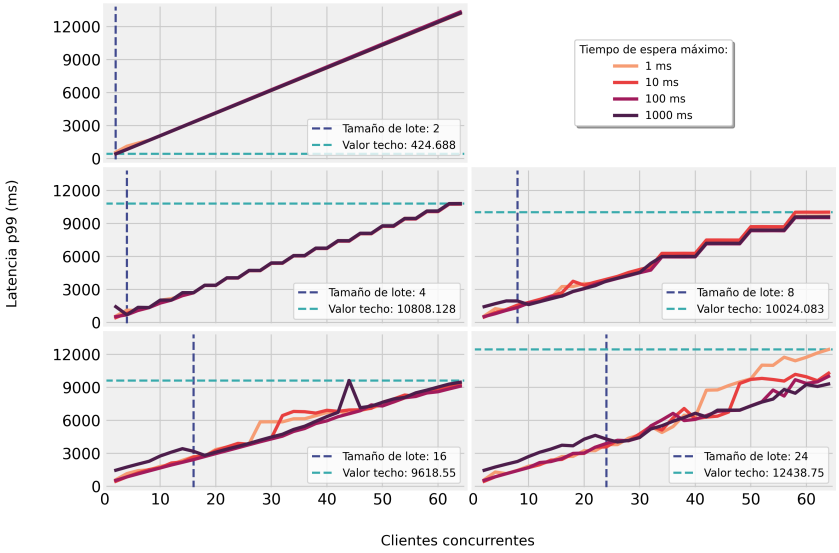
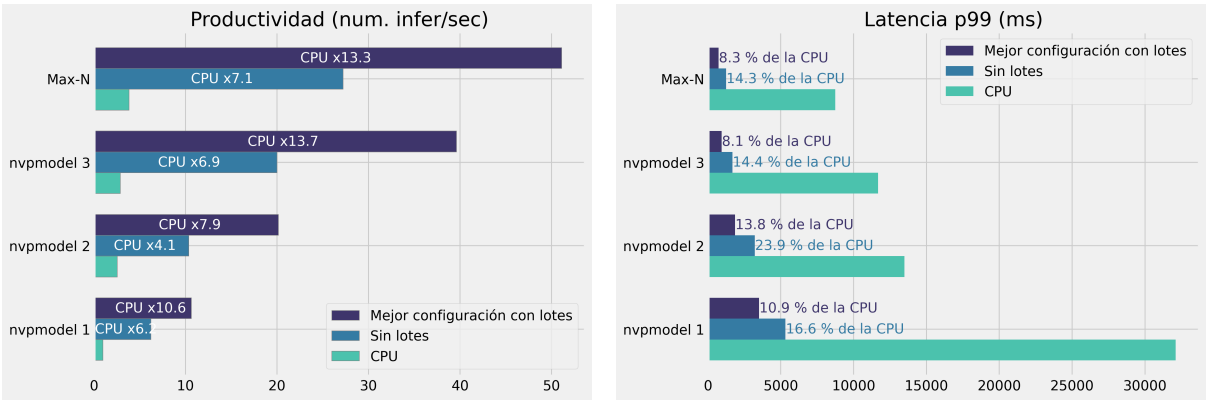


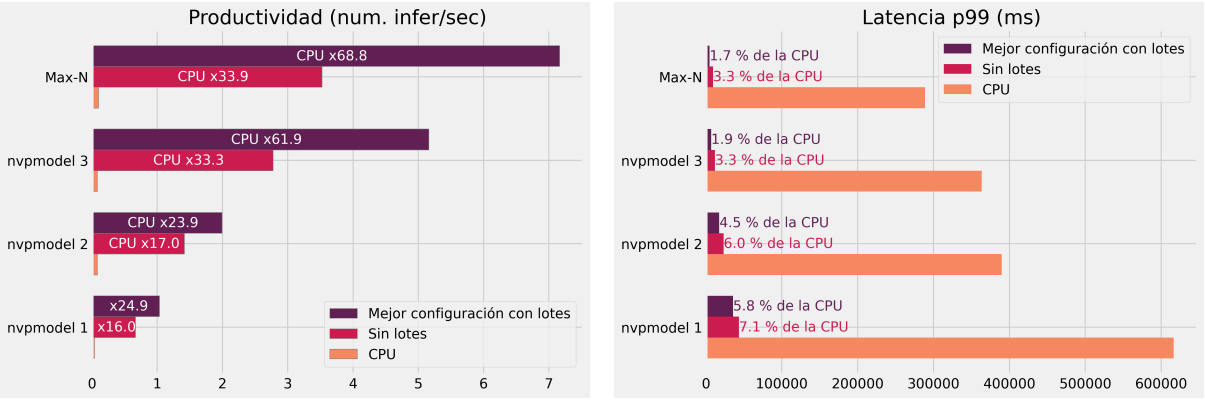
Figura 4.7: Jetson Orin latencia de *BEiT-Large* en modo NVP Max-N.



(a) Productividad pico para distintos modos NVP. (b) Latencia media para distintos modos NVP.

Figura 4.8: Mejora de rendimiento para Jetson Orin con la red *DenseNet-169*.

La mejoría de productividad y latencia desglosadas por cada modo NVP se pueden consultar en las Figuras 4.8 y 4.9. El rendimiento relativo de la red *BEiT-Large* ha sido optimizado en mayor medida que el de la *DenseNet-169*. Además, un análisis pormenorizado de los resultados permite concluir que la tasa de mejora relativa para ambas métricas es igual o incluso mejor en el modo NVP 3 que en el Max-N. Así, aunque el valor absoluto sea menor, el modo NVP 3 puede ser viable en escenarios de requisitos más laxos y en los que se quiera reducir el consumo energético. El Anexo E ofrece un estudio detallado de todos los modos NVP analizados.



(a) Productividad pico para distintos modos NVP. (b) Latencia media para distintos modos NVP.

Figura 4.9: Mejora de rendimiento para Jetson ORIN con la red *BEiT-Large*.

4.2.3. Plataforma Jetson Xavier

Los experimentos realizados en la plataforma Jetson XAVIER se definen a partir de los parámetros definidos en la Tabla 4.8. Como las especificaciones de este hardware son menos potentes que los de la Jetson ORIN, se ha probado la red *DenseNet-169* con valores de tamaño de lote máximo más pequeños y un único tamaño muy alto para comprobar si el servidor lo aguanta. El *BEiT-Large*, por su parte, ha sido sometido a la misma carga de trabajo.

Tabla 4.8: Experimentos con *DenseNet-169* y *BEiT-Large* en XAVIER.

Red neuronal	Parámetros	Valores
DenseNet-169	Concurrencia	Rango [2, 64] y saltos de 2.
	Tamaño máximo de lote	[2, 8, 16, 128]
	Tiempo máximo de espera en la cola (µs)	[1000, 10000, 100000, 1000000]
BEiT-Large	Concurrencia	Rango [2, 64] y saltos de 2.
	Tamaño máximo de lote	[2, 4, 8, 16, 24]
	Tiempo máximo de espera en la cola (µs)	[1000, 10000, 100000, 1000000]

La productividad medida frente al número de clientes concurrentemente enviando peticiones al servidor para la red neuronal *DenseNet-169* se muestra en la Figura 4.10, donde las tendencias de las diferentes líneas se asemejan a los resultados mostrados para Jetson ORIN.

En cambio, las tendencias de las líneas de datos para la red *BEiT-Large*, presentados en la Figura 4.11 son ligeramente diferentes a los anteriores resultados. En parte, debido a que los valores son muy próximos a la unidad y cualquier pequeña alteración que registre el cliente sintético al estabilizar los valores es más perceptible. Para el tamaño de lote 24 únicamente hay

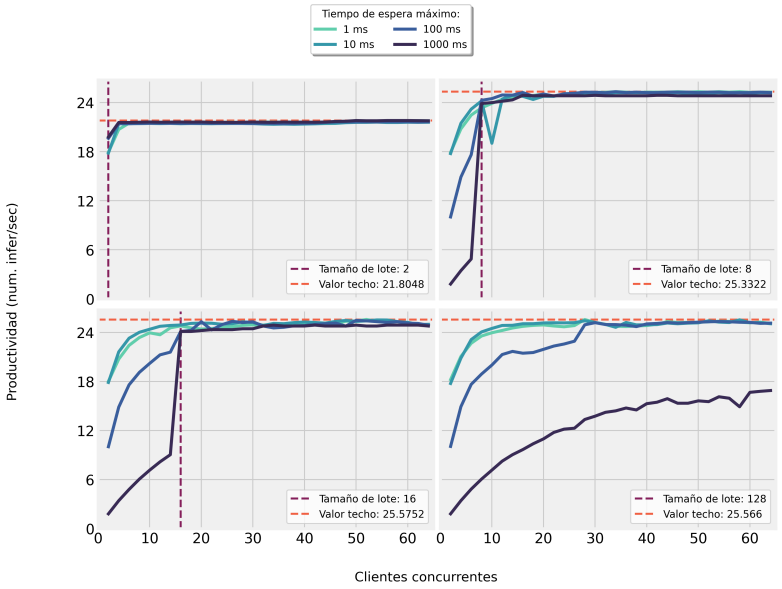


Figura 4.10: Jetson XAVIER productividad de *DenseNet-169* en modo NVP Max-N.

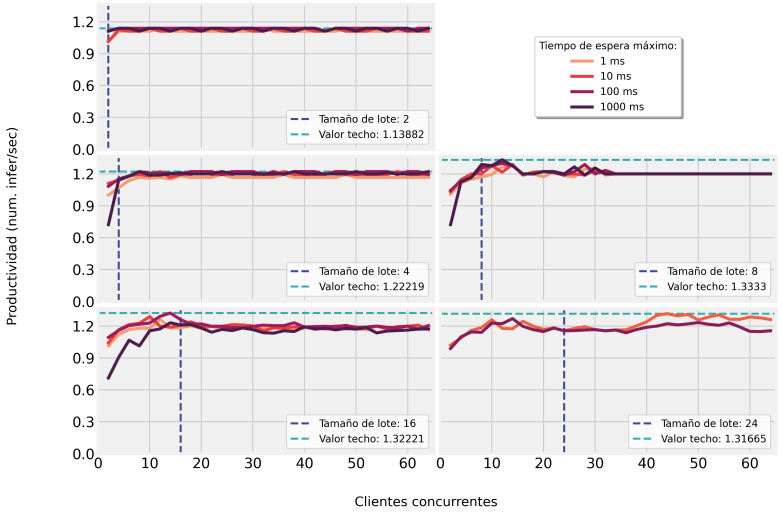


Figura 4.11: Jetson XAVIER productividad de *BEiT-Large* en modo NVP Max-N.

dos líneas dibujadas, ya que las otras dos configuraciones provocaban que el servidor dejase de funcionar.

En cuanto a la latencia en la plataforma Jetson XAVIER, las Figuras 4.12 y 4.13 referentes a las redes *DenseNet-169* y *BEiT-Large* respectivamente muestran comportamientos ya descritos con anterioridad. Las tendencias de los datos son lineales y el rendimiento de las métricas se ve

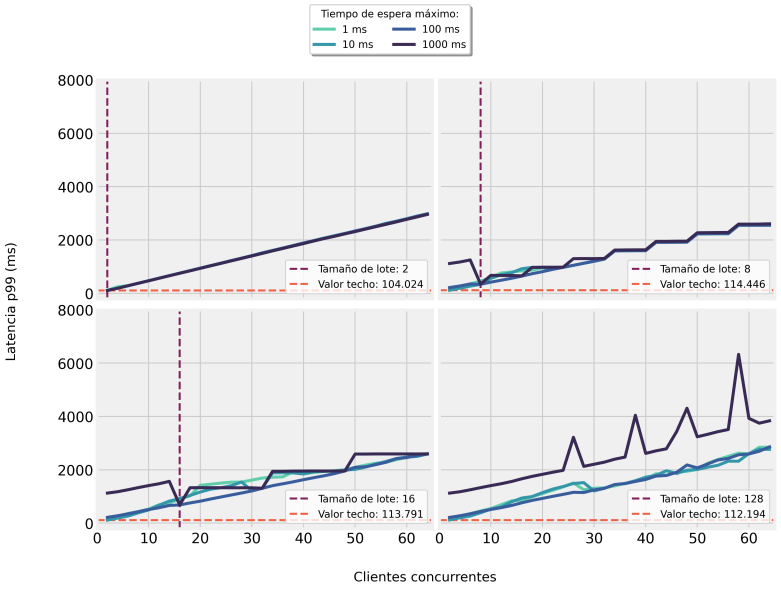


Figura 4.12: Jetson Xavier latencia de *DenseNet-169* en modo NVP Max-N.

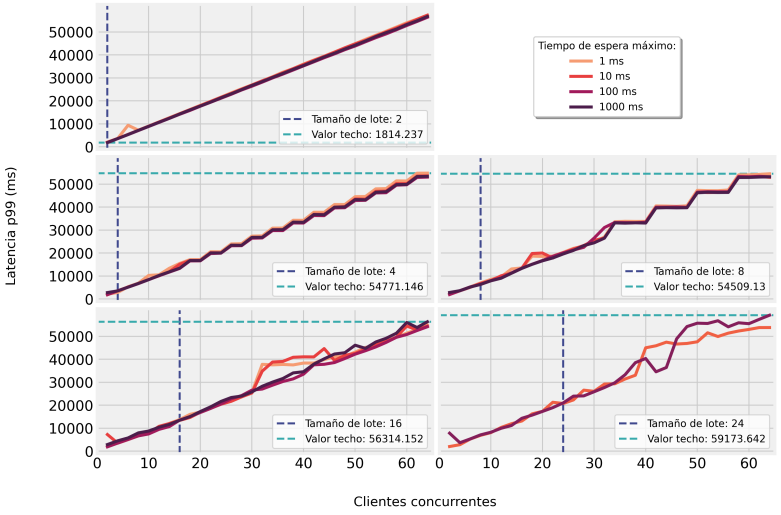
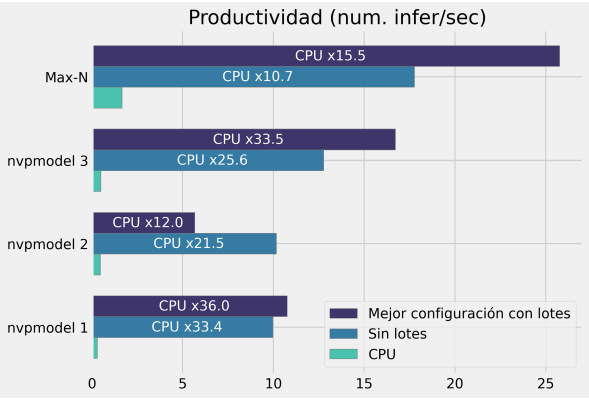


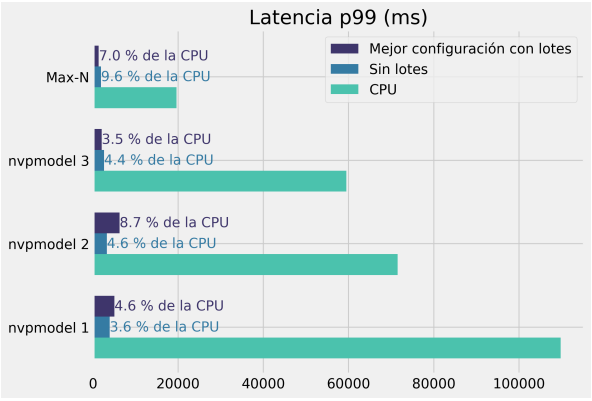
Figura 4.13: Jetson Xavier latencia de *BEiT-Large* en modo NVP Max-N.

deteriorando antes de alcanzar valores de concurrencia iguales al tamaño máximo de lote, igual que en la Jetson Orin.

El uso del programa clasificador de resultados ha permitido concluir que, tanto para la *DenseNet-169* como para el *BEiT-Large*, el mejor modo NVP ha sido el de Max-N, configurado a un tamaño de lote máximo de 16 y un tiempo de espera máximo de 100 ms. Otros modos NVP

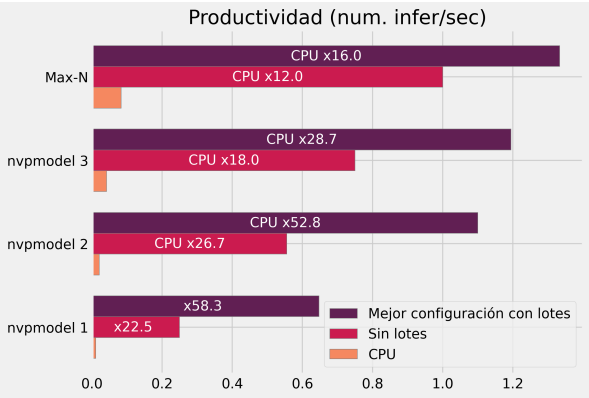


(a) Productividad pico para distintos modos NVP.

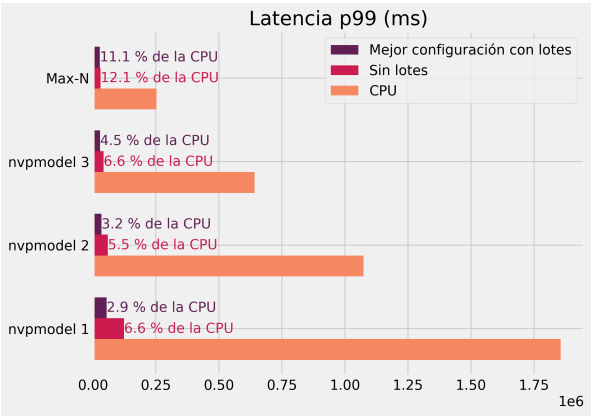


(b) Latencia media para distintos modos NVP.

Figura 4.14: Mejora de rendimiento para Jetson Xavier con la red *DenseNet-169*.



(a) Productividad pico para distintos modos NVP.



(b) Latencia media para distintos modos NVP.

Figura 4.15: Mejora de rendimiento para Jetson Xavier con la red *BEiT-Large*.

obtienen diferentes valores referencia, alterando escasamente la tendencia de las líneas de datos, al menos para las dos métricas que se han tenido en cuenta, la latencia y la productividad.

La mejoría de productividad y latencia desglosadas por cada modo NVP se pueden consultar en las Figuras 4.14 y 4.15. A diferencia de la plataforma anterior, se ha optimizado en mayor medida el rendimiento general de la red *DenseNet-169* que la de la *BEiT-Large*, y se ha producido una mejora relativa mucho más significativa del modo NVP 3 al Max-N. Es por ello que los datos reflejan un aumento de productividad de 33.5 veces en el modo NVP 3 contra 15.5 veces en Max-N respecto de la CPU y una latencia de tan sólo el 3.5 % para el primero y 7.0 % el segundo. El Anexo F ofrece un estudio detallado de todos los modos NVP analizados.

4.2.4. Plataforma Jetson Nano

Teniendo en cuenta que las especificaciones de la plataforma de bajo consumo Jetson Nano son las menos potentes de las tres utilizadas en este trabajo, se ha acotado en gran medida el rango de parámetros con los que experimentar. Los valores de los parámetros que definen los experimentos realizados aparecen en la Tabla 4.8

Tabla 4.9: Experimentos con *DenseNet-169* cuantizada en Nano.

Red neuronal	Parámetros	Valores
DenseNet-169 cuantizada	Concurrencia	Rango [2, 40] y saltos de 2.
	Tamaño máximo de lote	[2, 4, 8]
	Tiempo máximo de espera en la cola (µs)	[1000, 10000, 100000, 1000000]

La productividad de la red *DenseNet-169* cuantizada que se observa en la Figura 4.16 tiene un comportamiento similar al mostrado en las otras plataformas hasta alcanzar la concurrencia que coincide con el tamaño de lote máximo, si bien al aumentar el número de clientes en lugar de mantener este máximo el valor desciende. Esto ocurre porque tanto servidor como clientes son desplegados en la Jetson Nano, y a medida que aumenta la concurrencia más recursos consumen estos últimos, quedando disponibles menos para el servidor.

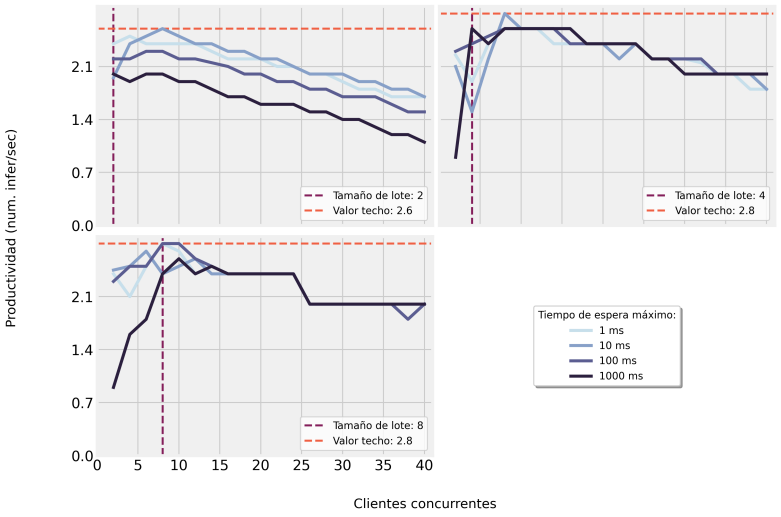


Figura 4.16: Jetson Nano productividad de *DenseNet-169* cuantizada en modo NVP Max-N.

Por su parte, la latencia es lineal e interrumpida por dientes de sierra, afín a los resultados que se han visto en las otras dos plataformas. Como se puede comprobar en la Figura 4.17, los tamaños de lote máximos medidos son pequeños y no permiten verificar al detalle si los valores anteriores a la concurrencia igual al lote máximo son peores. En el tercer gráfico se trasluce que esto es así, y las configuraciones tienen que esperar a que el tiempo de espera se alcance.

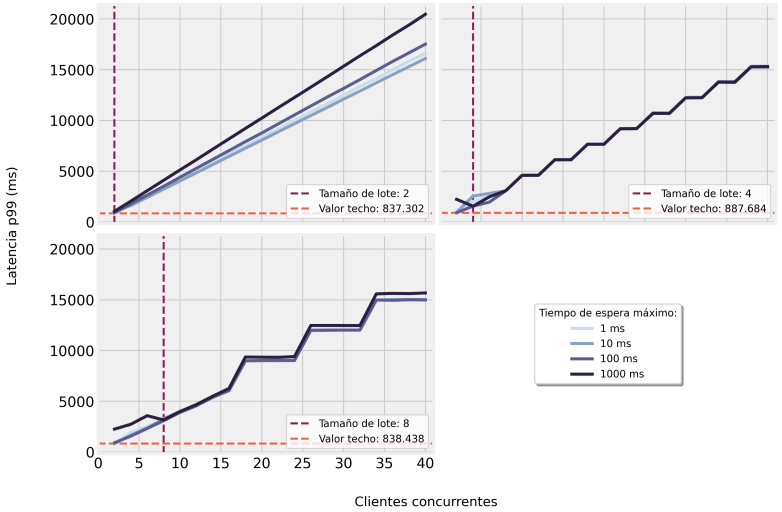
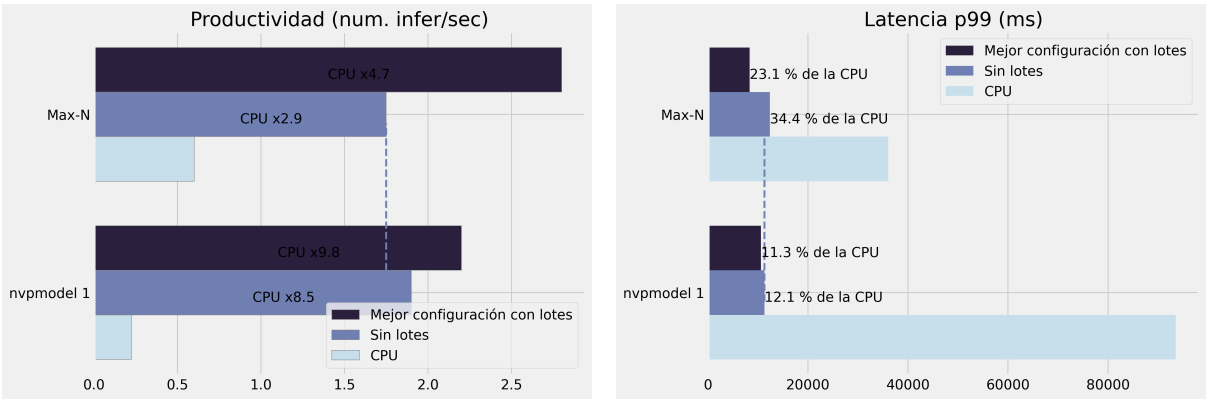


Figura 4.17: Jetson Nano latencia de *DenseNet-169* cuantizada en modo NVP Max-N.

El clasificador de resultados ha determinado que la configuración óptima para la red *DenseNet-169* cuantizada es el tamaño de lote 4 y tiempo de espera en cola de 100 ms. Una vez más, la configuración con mejor rendimiento pertenece al modo NVP de Max-N.

El modo NVP 1 sin agrupar peticiones de inferencia en el servidor consigue no sólo mejor optimización relativa que en el modo Max-N, sino también mejor rendimiento absoluto. Es decir, si se posee una red que no soporta crear lotes dinámicamente y se quiere desplegar en la Jetson Nano sobre Triton es preferible utilizar el modo NVP 1 por rendimiento, además de otras ventajas como el ahorro energético. Estos datos resumidos están reflejados en la Figura 4.18, y en el Anexo G se desglosan todas las gráficas del modo NVP 1, para ampliar la información.



(a) Productividad pico para distintos modos NVP. (b) Latencia media para distintos modos NVP.

Figura 4.18: Mejora de rendimiento para Jetson Nano con la red *DenseNet-169* cuantizada.

4.2.5. Análisis de resultados

Las redes *DenseNet-169* y *BEiT-Large* manejadas tienen el mismo objetivo, detectar patologías a partir de una imagen de un tórax, pero su composición es totalmente diferente. El *BEiT-Large* es una red con muchos más parámetros, por lo que es esperable que el tiempo de inferencia sea mayor que el de la *DenseNet-169* resultando en menor productividad y mayor latencia. Aun así, las tendencias de los resultados en las cuatro plataformas y las dos redes son muy similares. Se expone a continuación un breve resumen de las conclusiones más significativas:

1. El hecho de que el servidor agrupe peticiones de inferencia para crear lotes y así poder inferirlos en grupo resulta en una mayor productividad y una menor latencia.

2. Para cada entorno de ejecución, definido este por el tipo de red neuronal, plataforma hardware y modelo NVP, se observa un rendimiento máximo alcanzable. Por ejemplo, para la *DenseNet-169* en Orin con el modelo NVP Max-N este límite es alrededor de las 50 inferencias/segundo máximo y 78 milisegundos mínimo.

3. No es una única configuración del servidor Triton la que consigue acercarse a los valores óptimos, hay más de una combinación de parámetros diferente. En estos casos, lo que distingue una opción de las demás es la tendencia de los datos, por ejemplo, con cuántos clientes concurrentes alcanza los valores óptimos y si este rendimiento se mantiene para muchas concurrencias consecutivas.

4. Tamaños máximos de lote pequeños resultan en latencias ligeramente menores, con el coste de una menor productividad. Tamaños muy grandes no reportan mejores resultados y por el contrario, sí pueden generar demoras al no rellenar del todo el lote y verse retrasada la ejecución hasta que venza el tiempo máximo de espera en la cola.

5. Aumentar el tiempo máximo de espera en la cola no mejora ninguna de las métricas, los experimentos muestran constantemente resultados superiores para los tiempos más bajos. En cambio, valores altos pueden ser contraproducentes y generar retrasos indeseados, incurriendo en resultados inferiores.

4.2.6. Optimización del entorno del HGUCS

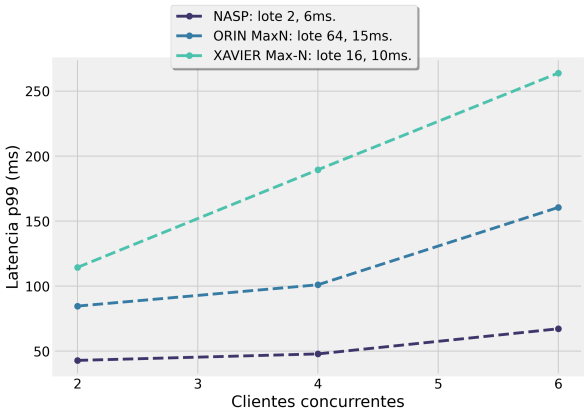
Partiendo de los datos recogidos con el cliente sintético, se ha utilizado el software clasificador desarrollado para analizar los resultados. El objetivo del análisis es ajustar la configuración del servidor desplegado en el HGUCS para las plataformas hardware probadas. En los casos de las Jetsons, se han considerado todos los modos NVP y se ha seleccionado aquel que mejor rendimiento ha reportado.

El hospital está dotado de cinco aparatos de RX en el servicio de urgencias desde las que capturan y envían estudios de RXT al PACS. Observando el flujo de imágenes se puede razonar que entre estos cinco aparatos capturan y envían alrededor de 30 RXT al día, una media de 1,25 imágenes a la hora. Asimismo, se ha contemplado que una misma modalidad rara vez envía más de un estudio en menos tiempo de lo que el servidor tarda en procesar la primera.

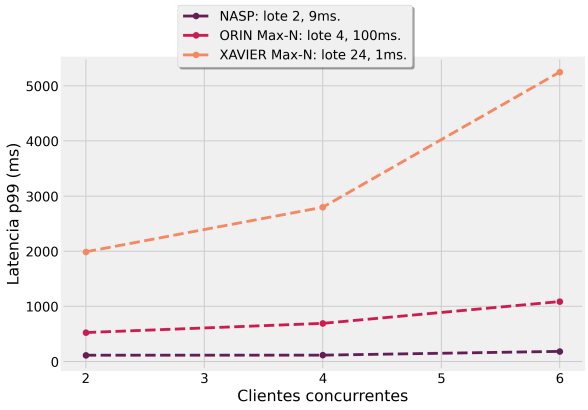
El servidor debe satisfacer al sistema no sólo cuando el número de peticiones sigue la media, sino también en las situaciones más desfavorables. El peor caso se produce cuando el servidor recibe una petición de inferencia de cada uno de los aparatos al mismo tiempo. Este caso corresponde a una concurrencia de cinco, por lo que el número de clientes máximo a tener cuenta sería cinco. Otro de los factores a determinar es la métrica por la que han de clasificarse las distintas configuraciones.

Como se mencionó anteriormente, si bien la productividad es especialmente importante para escenarios sin conexión, los servicios en línea en tiempo real priorizan la latencia. Los aparatos de rayos del hospital envían las radiografías al servidor implementado y esperan recibir la respuesta en el menor tiempo posible. En la Figura 4.19 se muestran solamente las mejores configuraciones para cada red neuronal clasificadas en base a su latencia, concretamente en base a la latencia p99.

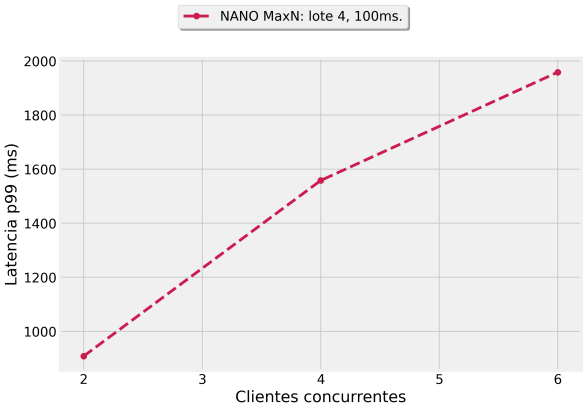
Una latencia inferior a 100 ms suele catalogarse como buena, y por debajo de 50 ms como muy buena. Esto quiere decir que los datos recogidos mediante experimentos para la red *DenseNet-169* son buenos, independientemente de la plataforma hardware. La red *DenseNet-169* cuantizada, en cambio, no cumple los requisitos mínimos en la Jetson Nano y debería ser sometida a otras técnicas para mejorar su latencia, tales como, la poda u otro tipo de cuantización. Tomando estos valores como referencia, también puede concluirse que para el HGUCS, la red *BEiT-Large* sólo sería viable en el nodo del clúster nasp y necesitaría métodos adicionales que acelerasen el tiempo de inferencia para poder desplegarse en condiciones razonables en las Jetsons.



(a) Red *DenseNet-169*.



(b) Red *BEiT-Large*.



(c) Red *DenseNet-169* cuantizada.

Figura 4.19: Mejor configuración por hardware simulando carga de trabajo del HGUCS.

Capítulo 5

Conclusiones

5.1. Contribuciones y resultados del trabajo

El proyecto aborda dos áreas de conocimiento complejas, como es el manejo de las redes neuronales y la HPC, y su objetivo principal pasa por conseguir un producto que debe funcionar en el mundo real, con las dificultades que ello conlleva. En este sentido, el resultado ha sido satisfactorio, ya que el servidor de inferencia desarrollado se encuentra en funcionamiento en el HGUCS, y su DB de resultados tiene ya más de mil diagnósticos de RXT PA y AP de pacientes de este hospital. Por tanto, se ha cumplido el objetivo principal y también los objetivos secundarios marcados como O1, O2, y O3 en este documento.

En respuesta al último objetivo perseguido, referido al análisis de la latencia y el rendimiento de los modelos AI, se ha logrado desarrollar un sistema que permite definir una batería de pruebas que puede ser ejecutada automáticamente sin necesidad de intervención humana. Además, se ha logrado analizar los resultados y obtener la mejor configuración para cada uno de los escenarios de prueba.

A nivel personal, el proyecto me ha permitido conocer un nuevo mundo como es el de la AI, y profundizar más en el HPC, concretamente en el del hardware de altas prestaciones y bajo consumo, en constante evolución. También he podido experimentar y aprender protocolos de comunicación y estándares informáticos que llevan muchos años siendo útiles en el mundo de la medicina, como el protocolo DICOM.

Los conceptos aprendidos sobre la computación paralela han sido fundamentales en lo que a la optimización del servidor de inferencia concierne. Estos conocimientos me han permitido realizar el análisis del rendimiento de las distintas redes neuronales a partir de mediciones

recogidas en diversos experimentos de carga. Gracias a lo aprendido acerca de la visualización de datos científicos, también he sido capaz de transformar los datos brutos recolectados en figuras comprensibles. Asimismo, he podido aplicar los conocimientos adquiridos en el área de la programación de sistemas a la hora de diseñar el middleware para garantizar la portabilidad y adaptabilidad.

5.2. Trabajos futuros

Como parte de futuros proyectos y ampliación del actual trabajo, existen diferentes mejoras a abordar de las que destaco las dos siguientes:

1. La decisión de trabajar con Python para el middleware ha garantizado la portabilidad del conjunto del sistema. Es un lenguaje sencillo que ha permitido un rápido desarrollo de la herramienta, pero su ejecución es lenta y no permite paralelismo. Una mejora importante en cuanto a rendimiento podría ser utilizar C++ en lugar de Python, aprovechando la API de C disponible de manera nativa en el servidor Triton.

2. Los experimentos de carga realizados se han aplicado exclusivamente al servidor de inferencia objeto de este proyecto. Sería interesante extender el estudio a la arquitectura completa, siendo ésta un área clara de mejora. Con este fin, existe software de código libre que puede ser utilizado como apoyo, los principales son:

   - **Prometheus:** software de código abierto para la supervisión y la generación de alertas de eventos. Recopila datos de servicios y servidores mediante el envío de solicitudes HTTP en puntos finales de métricas. Las métricas, como el uso de GPU y de memoria, pueden ser enviadas por Triton y almacenadas en Prometheus.
   - **Grafana:** software de código abierto para visualizar datos numéricos de serie temporal, como el rendimiento del sistema. Grafana tiene la ventaja de ser totalmente compatible con Prometheus.
   - **Locust:** herramienta de prueba de carga de código abierto, especialmente utilizado para medir el rendimiento de los sistemas de inferencia en línea para DNNs. Admite la ejecución de pruebas de carga que se distribuyen en varias máquinas y se puede usar para simular millones de usuarios simultáneos. Se ha utilizado en el proyecto, en la fase de desarrollo, para simular el envío de imágenes DICOM.

Anexos

Anexo A

Manejo de DICOM en Python

El Listado A.1 muestra un ejemplo de visualización de una imagen DICOM en Python.

```
import pydicom
import matplotlib.pyplot as plt
import pathlib
def main():
    # path DICOM image
    image_path = pathlib.Path(pathlib.Path(__file__).parent.resolve().as_posix
    (),"torax/files/CRdicom")
    ds = pydicom.dcmread(image_path)
    # plot DICOM image
    plt.figure()
    plt.imshow(ds.pixel_array, cmap=plt.cm.bone)
    plt.title("DICOM torax - TFM RELIANCE")
    plt.show()

if __name__ == '__main__':
    main()
```

Listado A.1: Visualización de imagen DICOM en Python.

El Listado A.1 lee un fichero DICOM completo, extrae la imagen que contiene y muestra la Figura A.1.

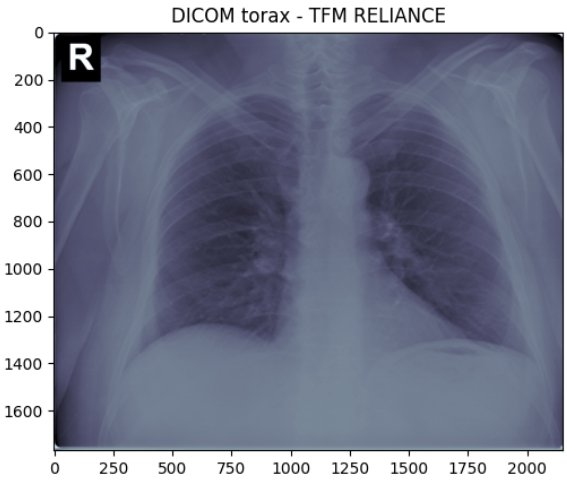


Figura A.1: Visualización de tórax utilizando pydicom

Anexo B

Procedimientos almacenados MariaDB

El Listado B.1 muestra el *script* de creación de los procedimientos almacenados utilizados en el proyecto. Concretamente, crea un procedimiento para insertar una entrada de datos en la tabla *ToraxPredictions* y devuelve el *Primary Key* que identifica únicamente la entrada añadida. El segundo procedimiento que se establece recibe el *Primary Key* del anterior procedimiento, el nombre de la etiqueta de salida y el resultado de inferencia e inserta una entrada de datos en la tabla *ToraxPredictionsLabels*.

```
USE `reliancedb`;
DROP procedure IF EXISTS `insert_torax_prediction`;
USE `reliancedb`;
DROP procedure IF EXISTS `reliancedb`.`insert_torax_prediction`;
;
DELIMITER $$
USE `reliancedb`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `insert_torax_prediction`(
  IN dicomId varchar(255),
    IN received_time datetime,
    IN processed_time datetime,
    IN acquisition_time datetime,
    OUT torax_predictions_id INT
)
SQL SECURITY INVOKER
BEGIN
  -- Crear entrada en Dicoms
  INSERT INTO dicoms (DicomID) VALUES (dicomId) ON DUPLICATE KEY UPDATE DicomID
     = dicomId;
    -- Insertar entrada en ToraxPredictions
    INSERT INTO toraxpredictions (AcquisitionTime, ReceivedTime, ProcessedTime,
     DicomID, ModelID) VALUES (acquisition_time, received_time, processed_time,
    dicomId, 1);
    -- No hay problema de que otro hilo inserte otra entrada porque
    LAST_INSERT_ID() es especifico de la conexiÃ§n.
    SELECT LAST_INSERT_ID() INTO torax_predictions_id;
END$$
DELIMITER ;
;
USE `reliancedb`;
DROP procedure IF EXISTS `insert_torax_prediction_label`;
USE `reliancedb`;
DROP procedure IF EXISTS `reliancedb`.`insert_torax_prediction_label`;
;
DELIMITER $$
USE `reliancedb`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `insert_torax_prediction_label`(
  IN torax_predictions_id INT,
    IN label_name VARCHAR(255),
    IN result DOUBLE
)
SQL SECURITY INVOKER
BEGIN
  DECLARE label_id INT;
  SELECT LabelsID FROM labels WHERE Disease = label_name INTO label_id;
  INSERT INTO toraxpredictionslabels (ToraxPredictionsID, LabelsID, result)
    VALUES(torax_predictions_id, label_id, result);
END$$
DELIMITER ;
;
```

Listado B.1: Procedimientos almacenados en MariaDB.

Anexo C

Certificados SSL

En el Listado C.1 se puede consultar la creación de los certificados SSL que se han utilizado para securizar las comunicaciones entre el middleware y el servidor Triton. El *script* genera una llave privada y una pública autofirmadas, es decir, sin ser validado por una autoridad de certificación (*Certificate Authority* o CA). A continuación, genera una llave privada y una solicitud de firma de certificado (*Certificate Signing Request* o CSR) para el servidor Triton, y firma el CSR con la llave privada de la CA autofirmada para formar un CSR firmado. Se utiliza el mismo procedimiento para crear la llave privada, el CSR y el CSR firmado para el cliente. Por último, se genera una cadena de certificados (certificate chain) que utilizará el cliente para verificar que el remitente y las CAs son de confianza.

```
CA_PATH="./ca"
SERVER_PATH="./server"
CLIENT_PATH="./client"

rm ${CA_PATH}/*.pem
rm ${SERVER_PATH}/*.pem
rm ${CLIENT_PATH}/*.pem

# 1. Generate CA's private key and self-signed certificate
openssl req -x509 -newkey rsa:4096 -days 365 -nodes -keyout ${CA_PATH}/ca-key.
    pem -out ${CA_PATH}/ca-cert.pem -subj "/CN=ssl-ca-cert"

echo "CA's self-signed certificate"
openssl x509 -in ${CA_PATH}/ca-cert.pem -noout -text

# 2. Generate web server's private key and certificate signing request (CSR)
openssl req -newkey rsa:4096 -nodes -keyout ${SERVER_PATH}/server-key.pem -out
    ${SERVER_PATH}/server-req.pem -subj "/CN=localhost"

# 3. Use CA's private key to sign web server's CSR and get back the signed
    certificate
openssl x509 -req -in ${SERVER_PATH}/server-req.pem -days 365 -CA ${CA_PATH}/ca
    -cert.pem -CAkey ${CA_PATH}/ca-key.pem -CAcreateserial -out ${SERVER_PATH}/
    server-cert.pem

echo "Server's signed certificate"
openssl x509 -in ${SERVER_PATH}/server-cert.pem -noout -text

# 4. Generate client's private key and certificate signing request (CSR)
openssl req -newkey rsa:4096 -nodes -keyout ${CLIENT_PATH}/client-key.pem -out
    ${CLIENT_PATH}/client-req.pem -subj "/CN=ssl-ca-client-cert"

# 5. Use CA's private key to sign client's CSR and get back the signed
    certificate
openssl x509 -req -in ${CLIENT_PATH}/client-req.pem -days 365 -CA ${CA_PATH}/ca
    -cert.pem -CAkey ${CA_PATH}/ca-key.pem -CAcreateserial -out ${CLIENT_PATH}/
    client-cert.pem

echo "Client's signed certificate"
openssl x509 -in ${CLIENT_PATH}/client-cert.pem -noout -text

# 6. Create client's certificate chain
openssl x509 -req -days 365 -in ${CLIENT_PATH}/client-req.pem -CA ${CA_PATH}/ca
    -cert.pem -CAkey ${CA_PATH}/ca-key.pem -set_serial 01 -out ${CLIENT_PATH}/
    client-chain.pem

#7. Verify client's certificate chain with CA's public key
openssl verify -CAfile ${CA_PATH}/ca-cert.pem ${CLIENT_PATH}/client-chain.pem
```

Listado C.1: Creación de los certificados SSL.

Anexo D

Concurrencia de modelos en NVIDIA Triton

La opción de configuración para permitir la ejecución concurrente de redes en Triton se llama “instance-group” y permite especificar cuántas ejecuciones paralelas de cada red deben ser permitidas. Es decir, cuántas instancias de cada red pueden ejecutarse a la vez. El valor por defecto de “instance-group” es uno, de modo que, Triton da a cada red una única instancia por cada GPU disponible en el sistema.

Escenario 1: Se presenta un escenario con una única GPU y dos redes diferentes desplegados, ambos con una única instancia habilitada. Pueden darse los siguientes casos:

- Llegan dos peticiones simultáneas, una para cada red: Triton programará las peticiones en las GPUs disponibles y comenzará a trabajar en ambos cálculos en paralelo. Este caso se muestra en la Figura D.1.
- Cuando llega una petición de inferencia para cada red, Triton trabaja concurrentemente. En el caso ilustrado en la Figura D.2, llega una tercera petición para una red que ya está utilizando su instancia para procesar su primera petición. Como hay más peticiones que instancias de red, Triton serializará su ejecución programándolas una a una en la GPU y la última petición tendrá que esperar a que la instancia quede libre.

Escenario 2: en la Figura D.3 se presenta un escenario con una única GPU y dos redes diferentes. Las redes son una DenseNet-169 configurada para permitir una instancia y una BEiT-Large con valor dos para la configuración de la “instance-group”.

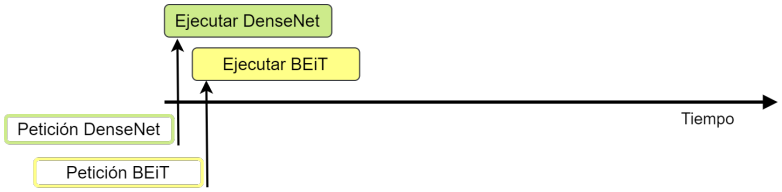


Figura D.1: Mismo número de instancias de red y peticiones.

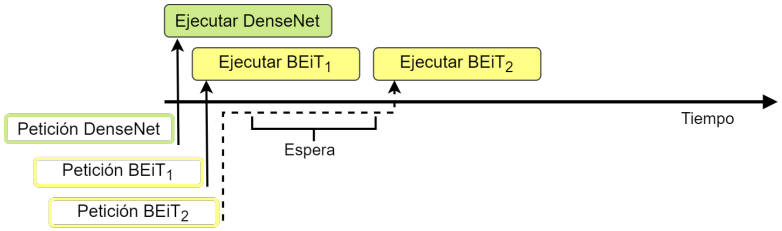


Figura D.2: Mayor número de peticiones que instancias de red.

Cuando llega una única petición para la red DenseNet-169, Triton la empieza a procesar. En este momento, tres solicitudes de inferencia para la red BEiT-Large son recibidas casi en el mismo instante. El número de instancias permitidas para este modelo es de dos, por lo que Triton procesa paralelamente dos de las peticiones para la BEiT-Large y la de la DenseNet-169. La tercera solicitud es encolada y deberá esperar a que quede libre una de las instancias de su red objetivo para ser procesada.

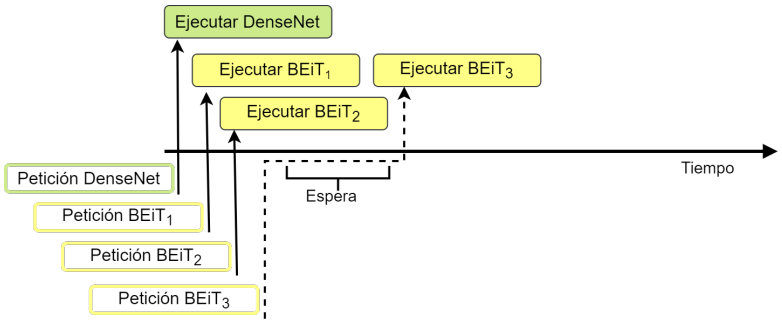


Figura D.3: Mayor número de peticiones en redes multi-instancia.

Anexo E

Resultados detallados en Jetson Orin

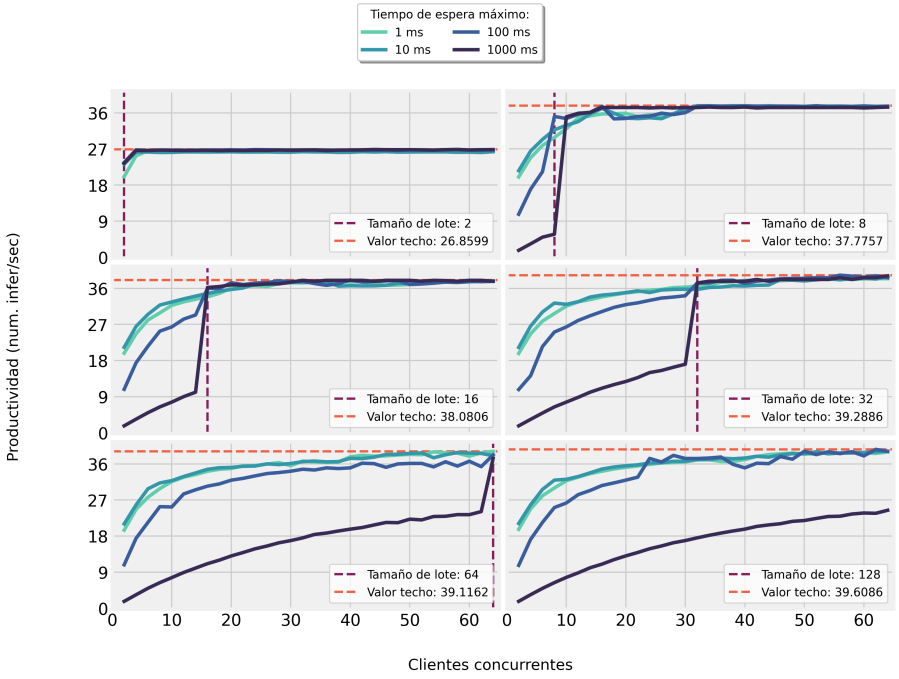


Figura E.1: Productividad de DenseNet-169 en modo NVP 3.

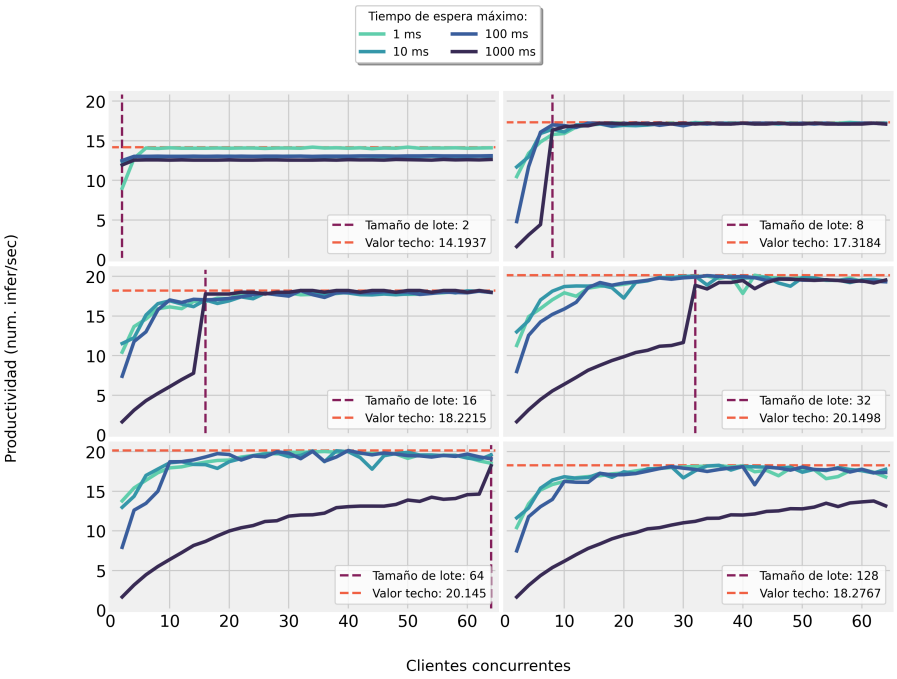


Figura E.2: Productividad de DenseNet-169 en modo NVP 2.

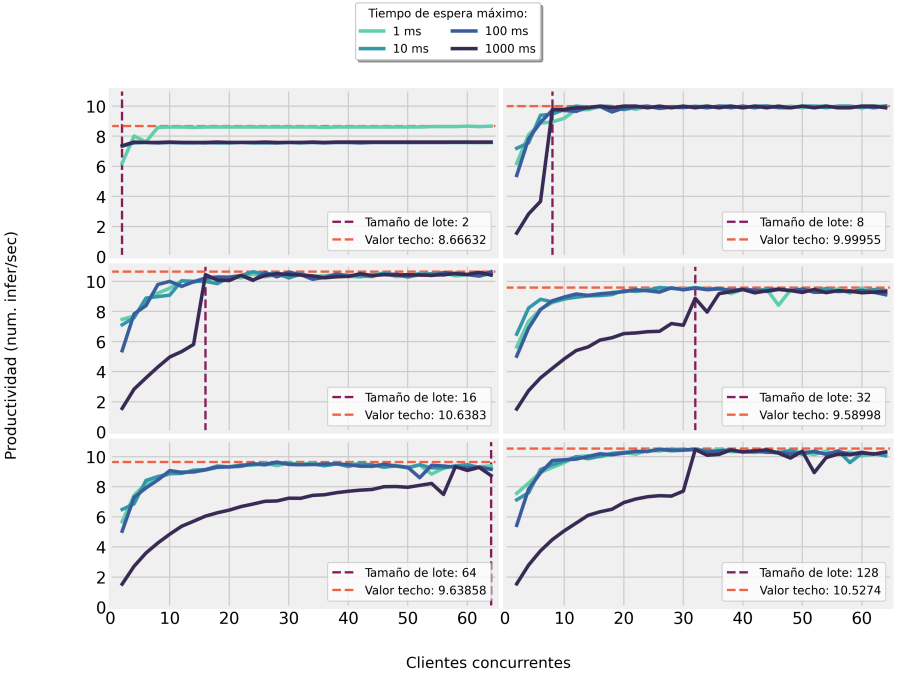


Figura E.3: Productividad de DenseNet-169 en modo NVP 1.

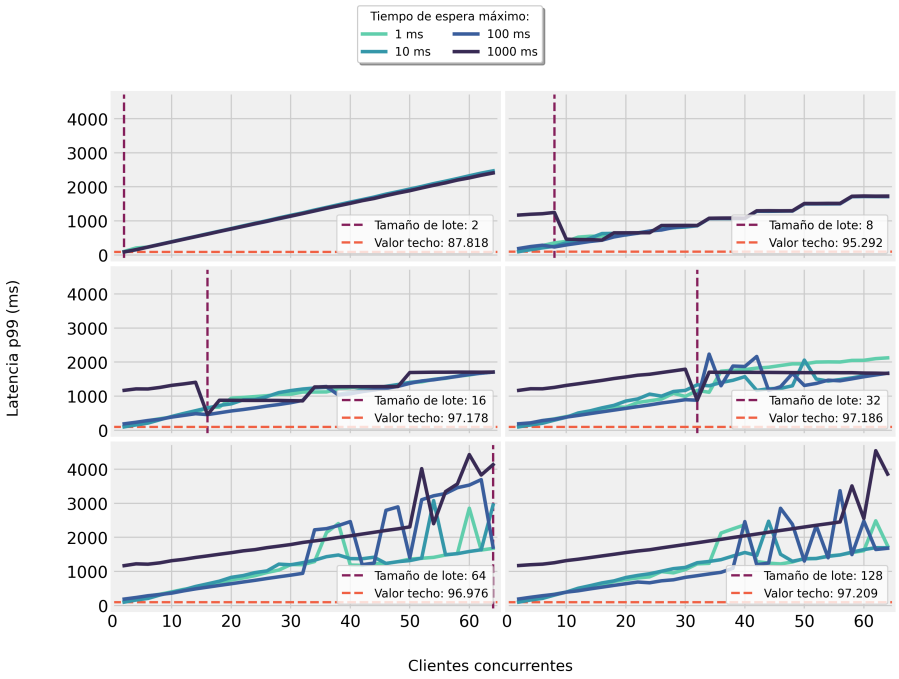


Figura E.4: Latencia de DenseNet-169 en modo NVP 3.

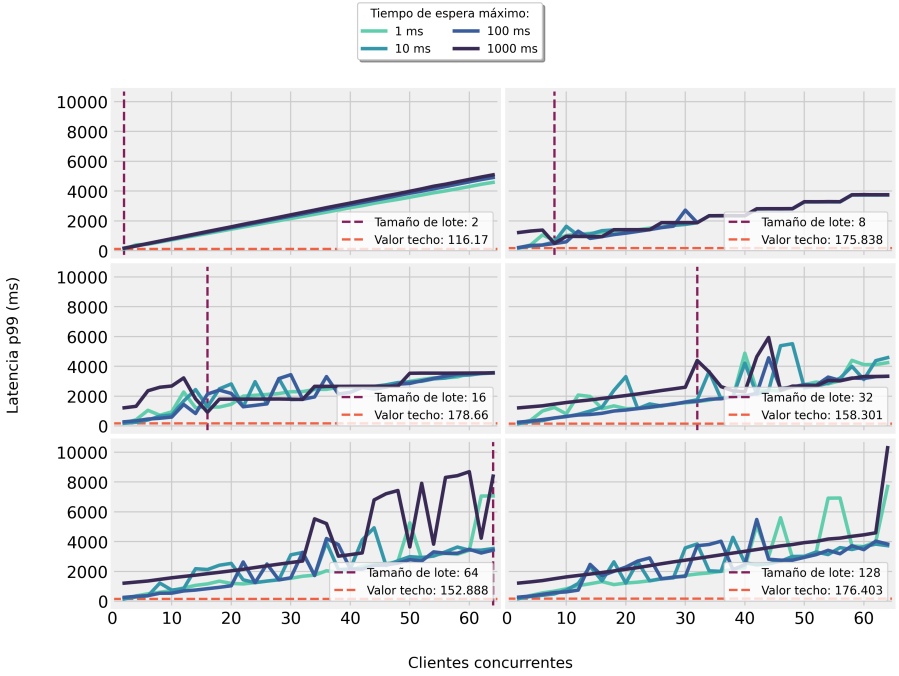


Figura E.5: Latencia de DenseNet-169 en modo NVP 2.

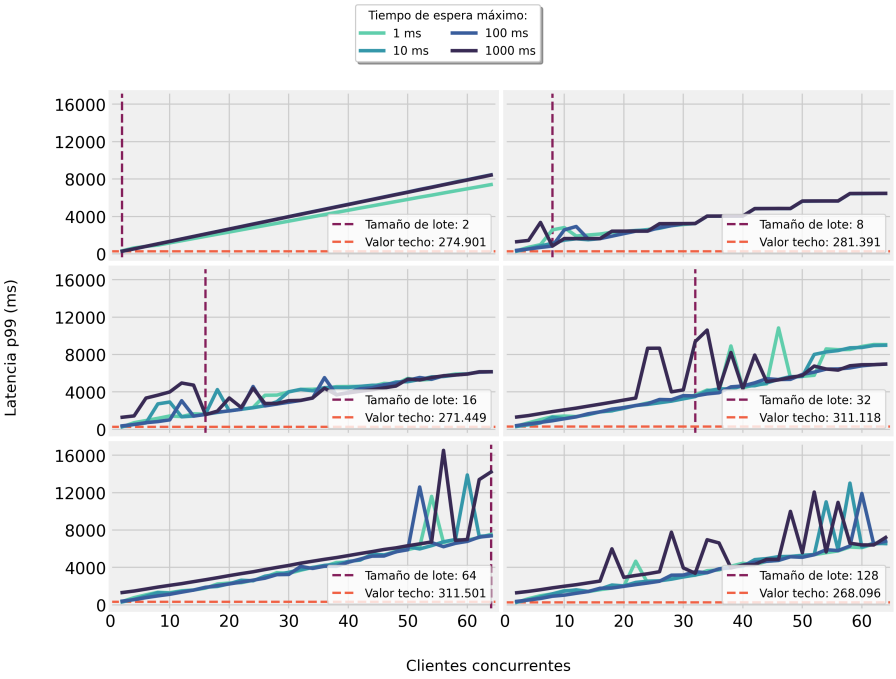


Figura E.6: Latencia de DenseNet-169 en modo NVP 1.

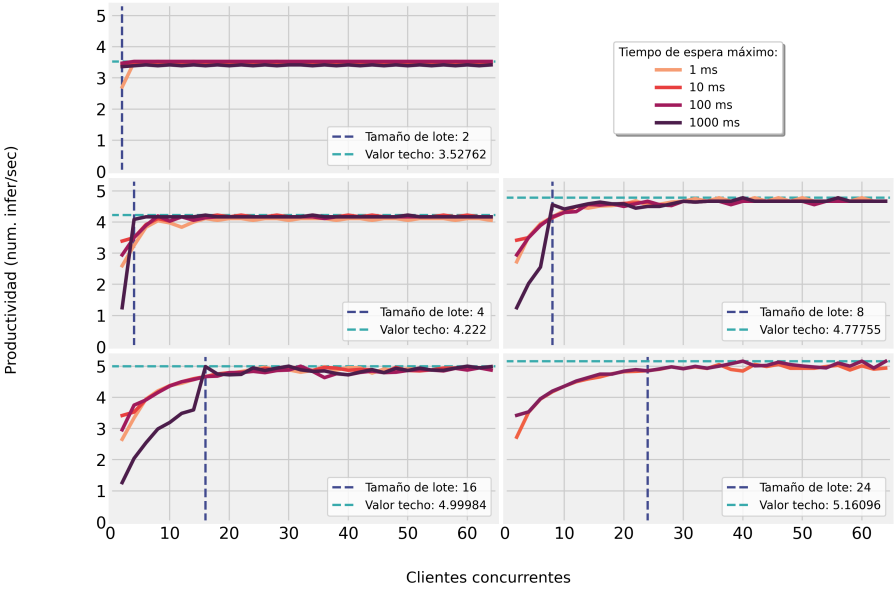


Figura E.7: Productividad de BEiT-Large en modo NVP 3.

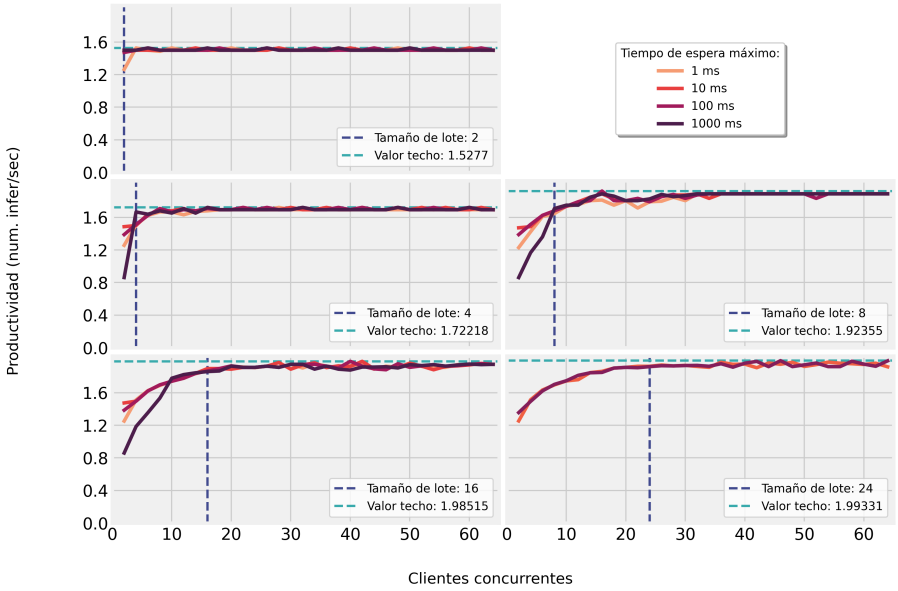


Figura E.8: Productividad de BEiT-Large en modo NVP 2.

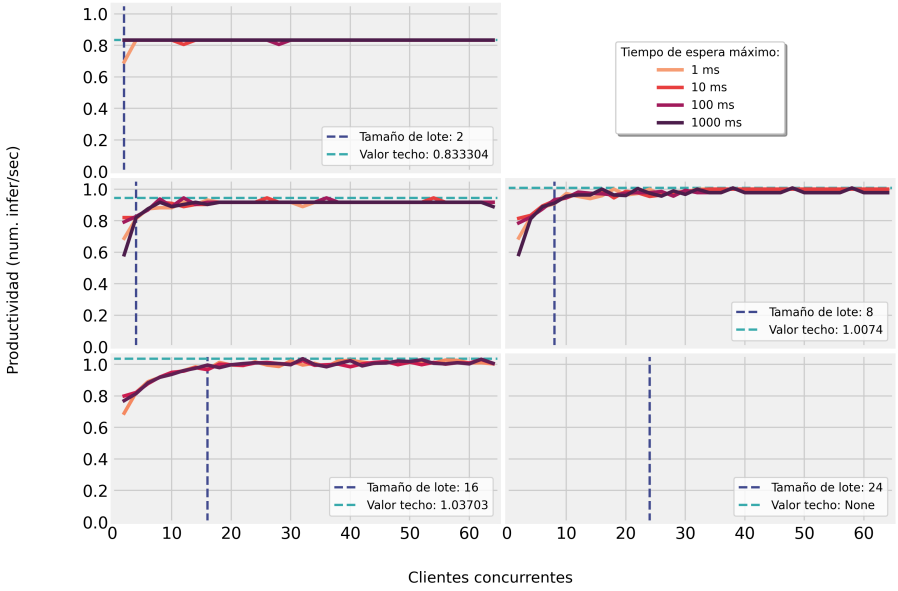


Figura E.9: Productividad de BEiT-Large en modo NVP 1.

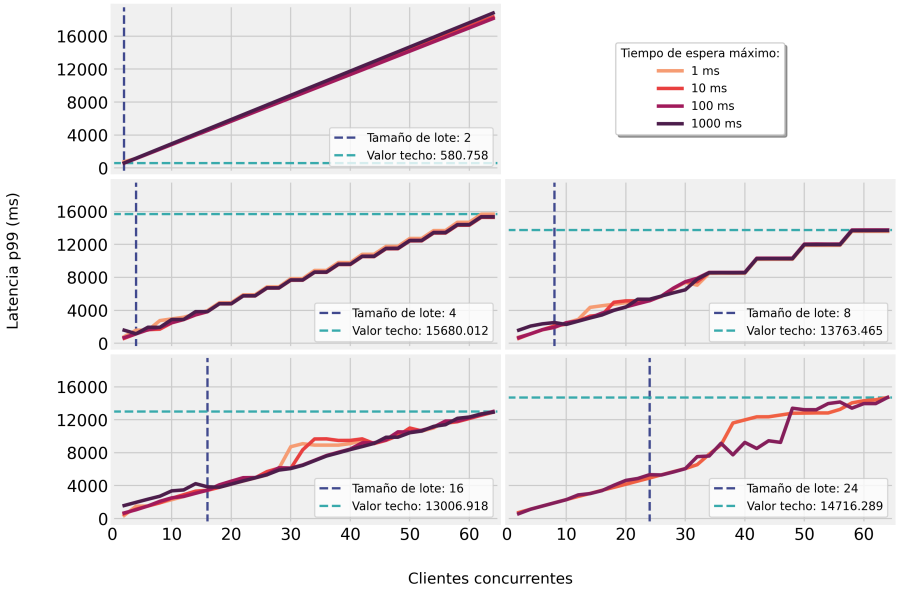


Figura E.10: Latencia de BEiT-Large en modo NVP 3.

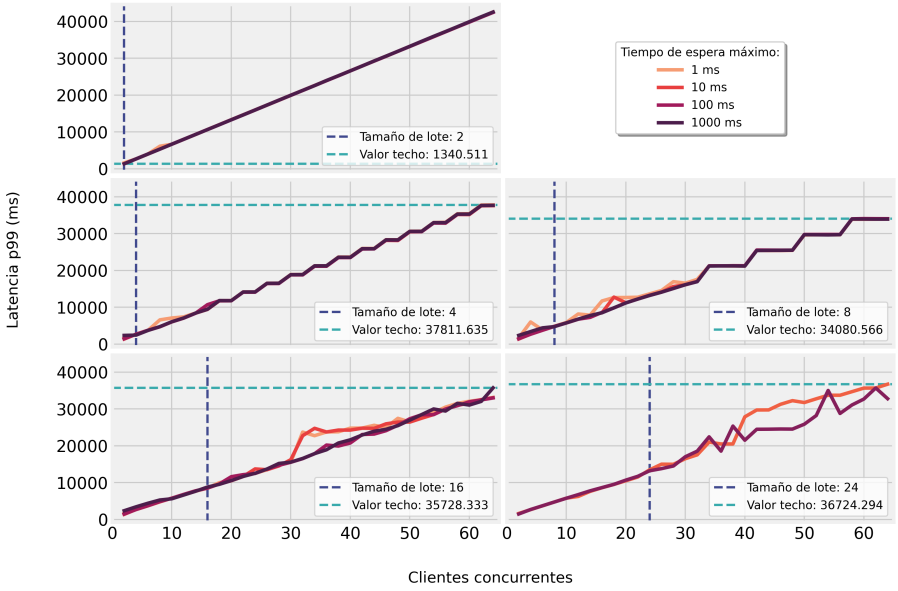


Figura E.11: Latencia de BEiT-Large en modo NVP 2.

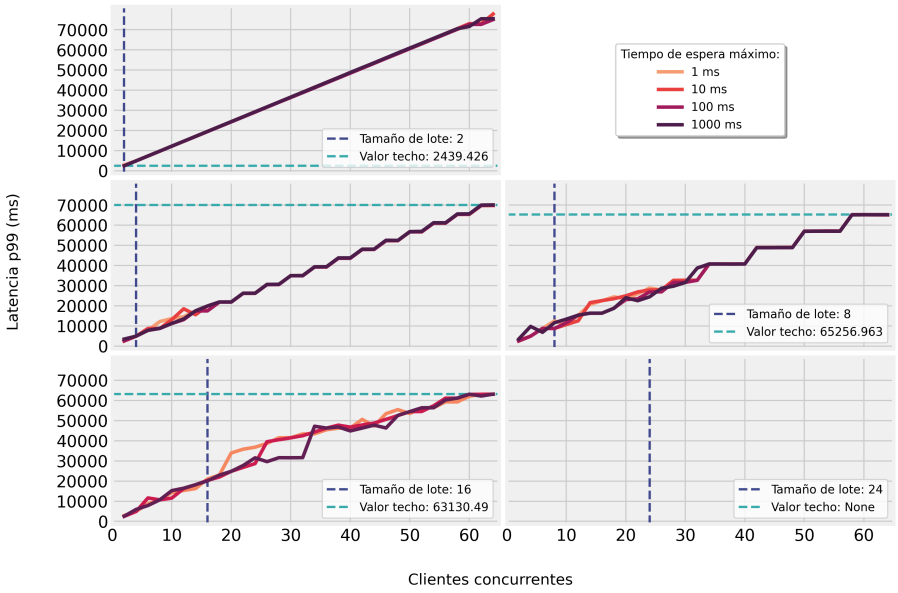


Figura E.12: Latencia de BEiT-Large en modo NVP 1.

Anexo F

Resultados detallados en Jetson Xavier

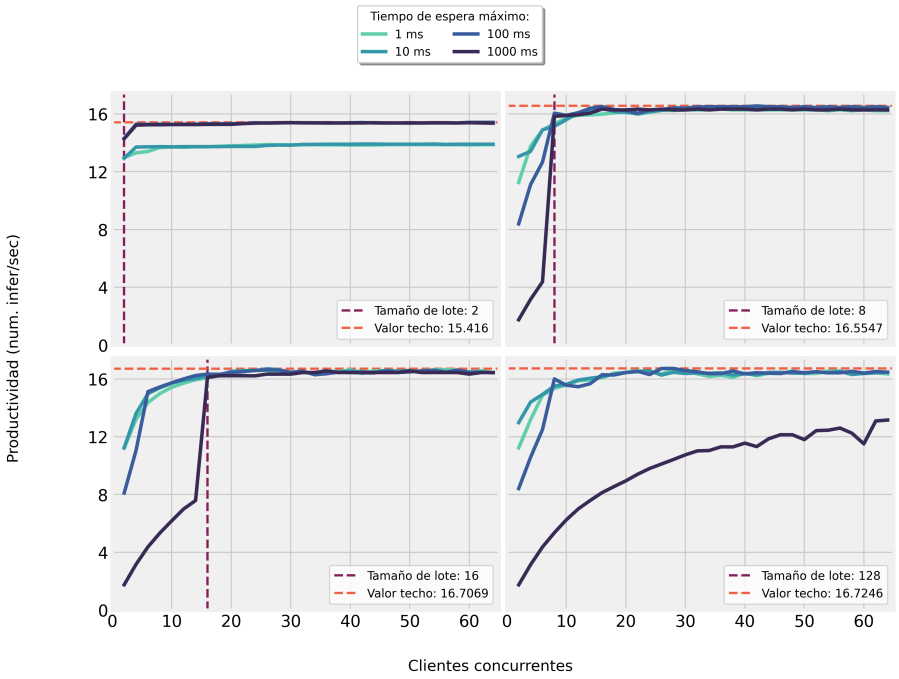


Figura F.1: Productividad de DenseNet-169 en modo NVP 3.

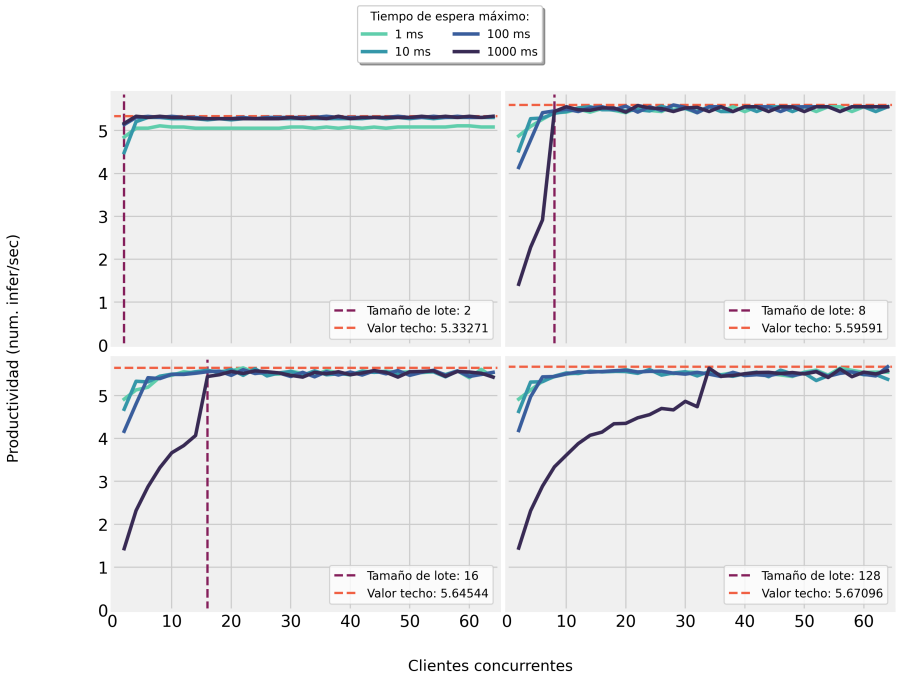


Figura F.2: Productividad de DenseNet-169 en modo NVP 2.

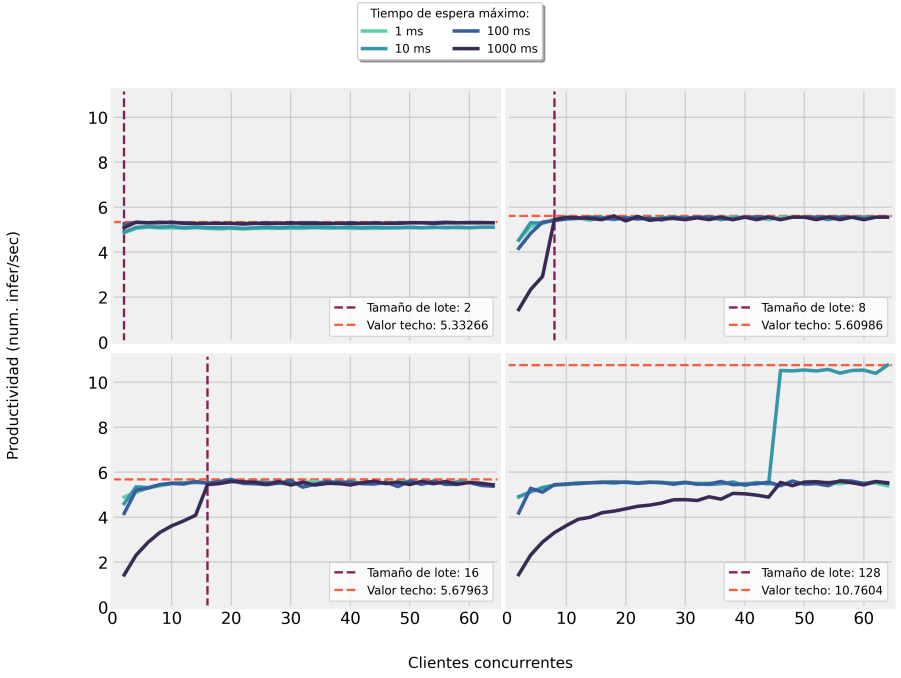


Figura F.3: Productividad de DenseNet-169 en modo NVP 1.

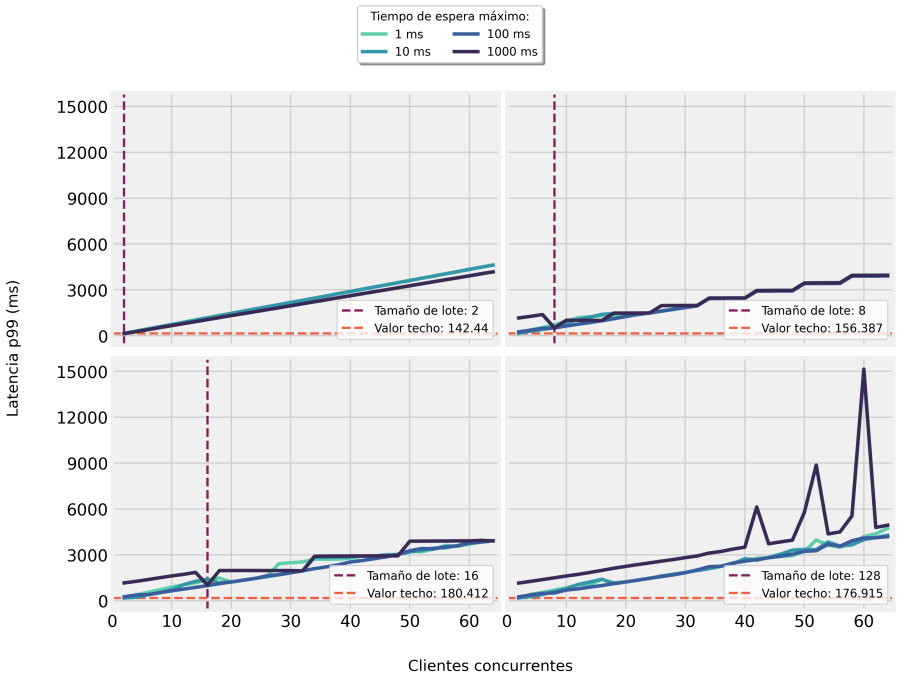


Figura F.4: Latencia de DenseNet-169 en modo NVP 3.

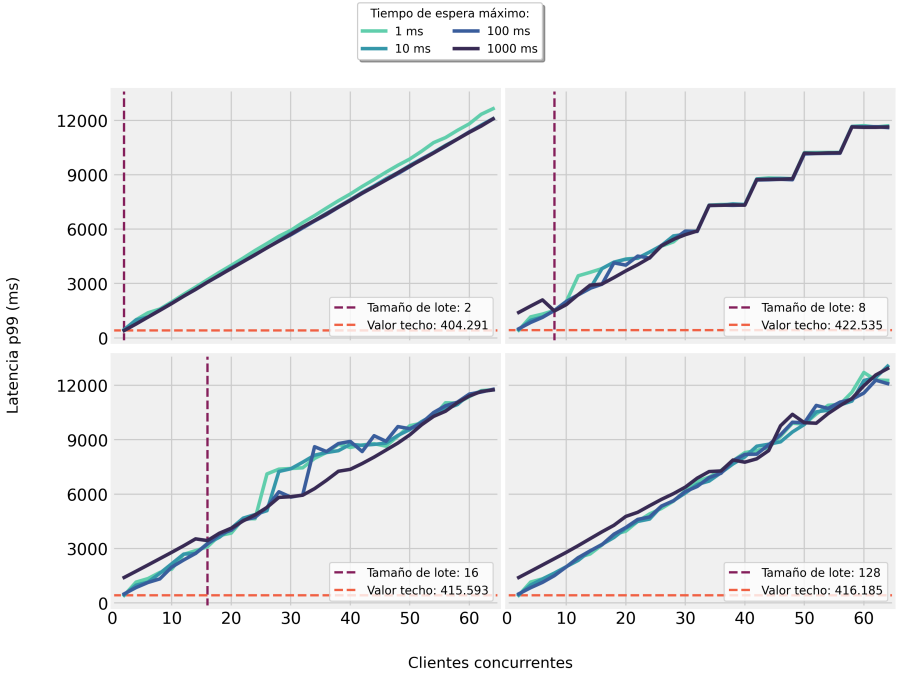


Figura F.5: Latencia de DenseNet-169 en modo NVP 2.

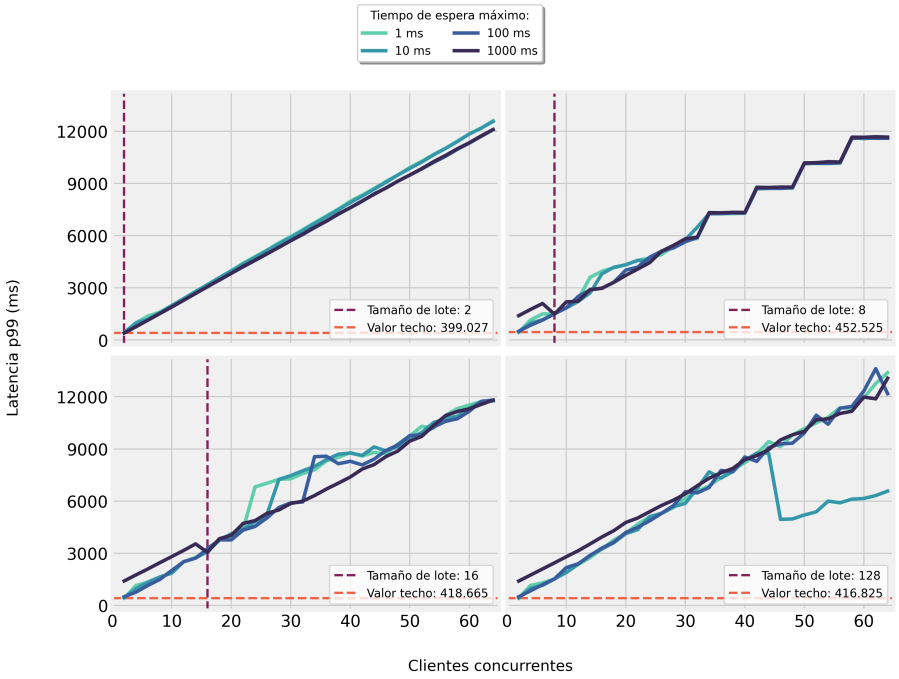


Figura F.6: Latencia de DenseNet-169 en modo NVP 1.

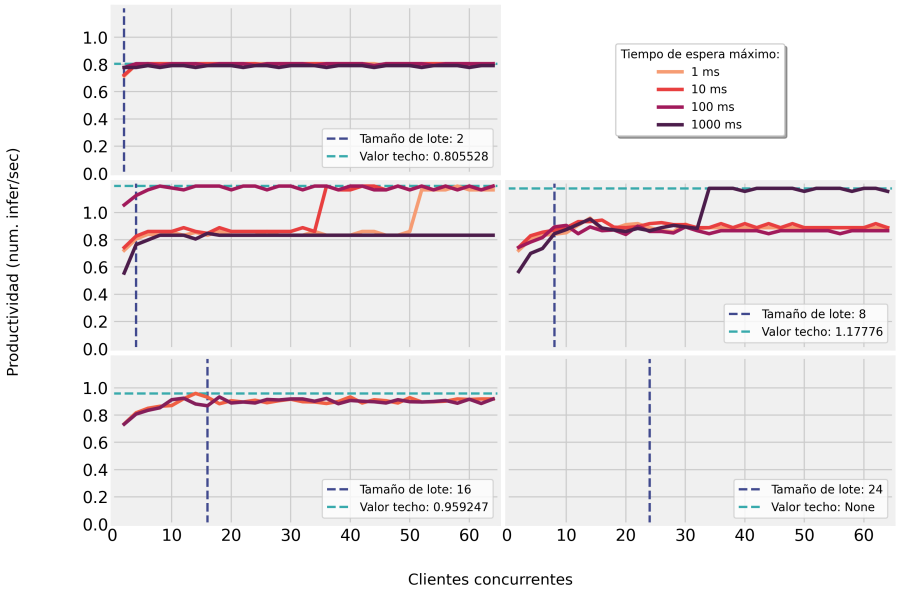


Figura F.7: Productividad de BEiT-Large en modo NVP 3.

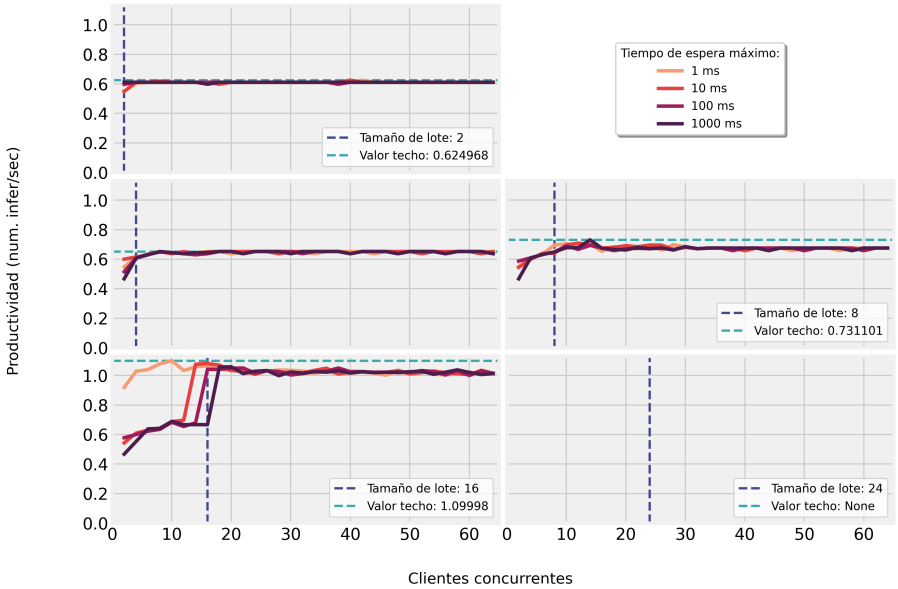


Figura F.8: Productividad de BEiT-Large en modo NVP 2.

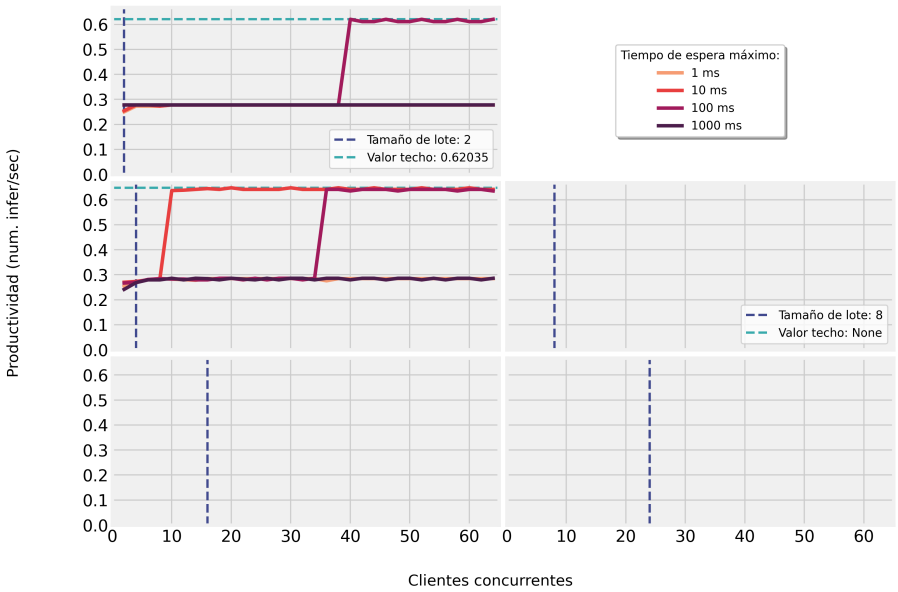


Figura F.9: Productividad de BEiT-Large en modo NVP 1.

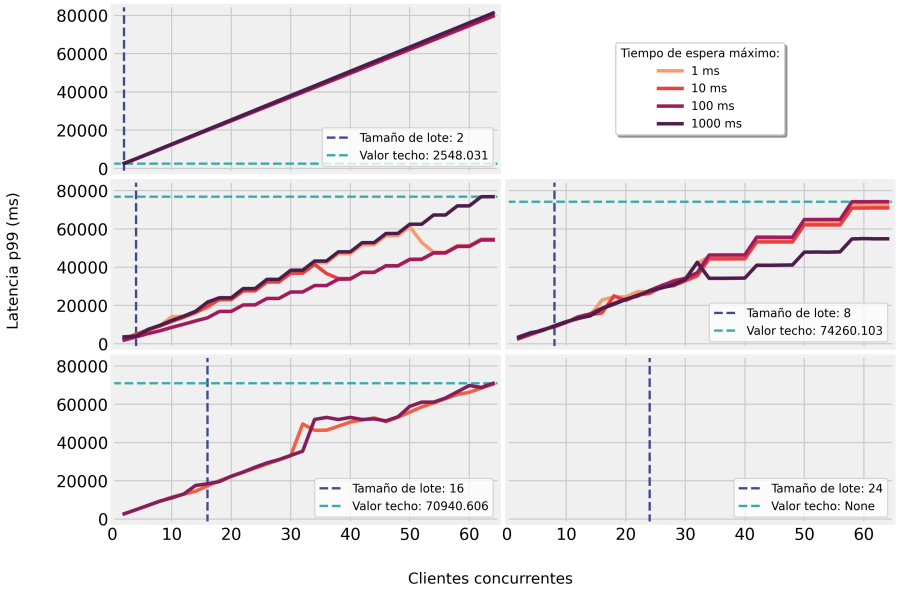


Figura F.10: Latencia de BEiT-Large en modo NVP 3.

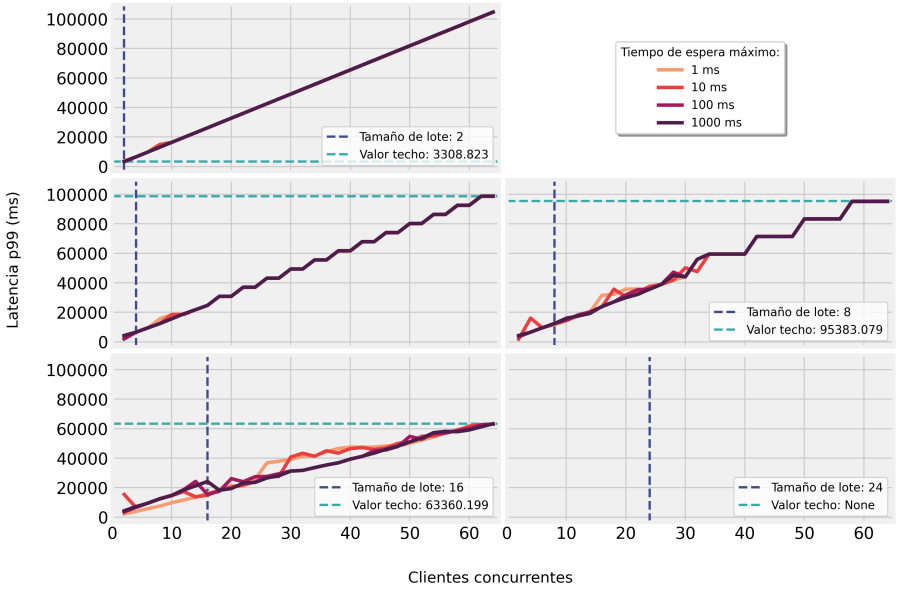


Figura F.11: Latencia de BEiT-Large en modo NVP 2.

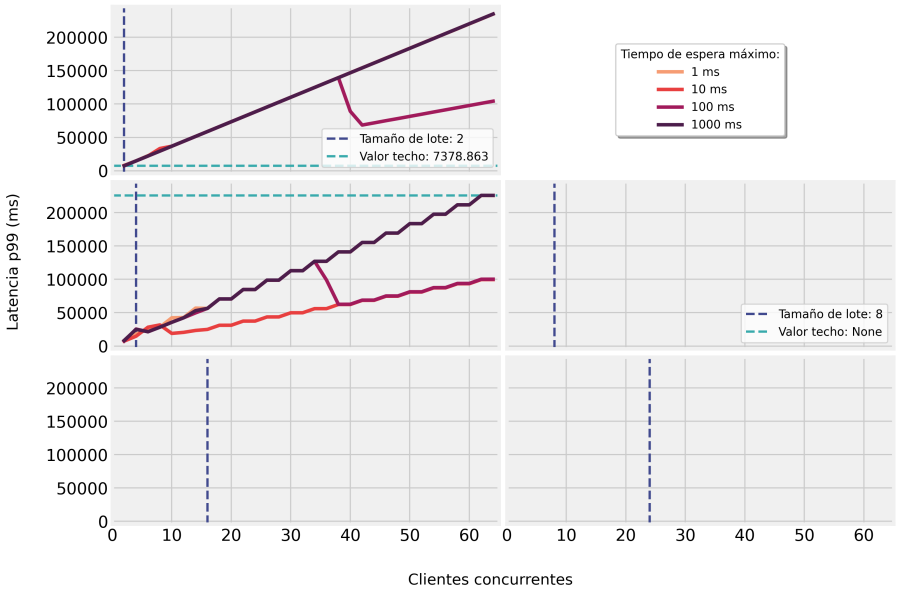


Figura F.12: Latencia de BEiT-Large en modo NVP 1.

Anexo G

Resultados detallados en Jetson Nano

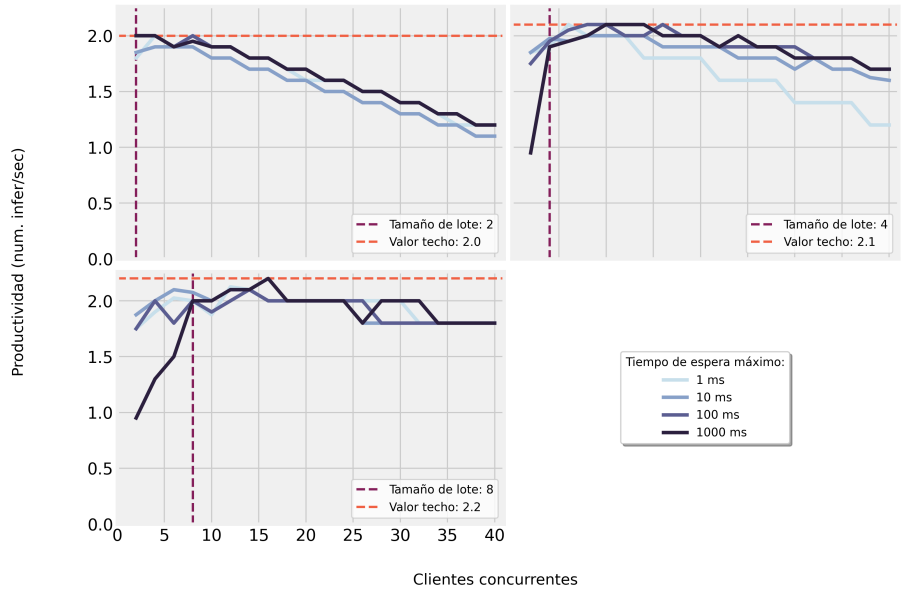


Figura G.1: Productividad de DenseNet-169 cuantizada en modo NVP 1.

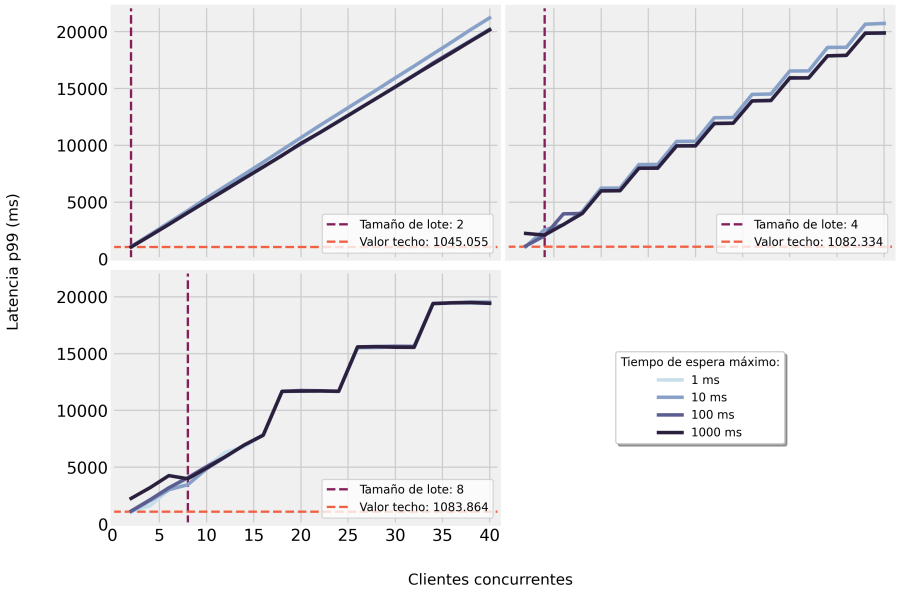


Figura G.2: Latencia de DenseNet-169 cuantizada en modo NVP 1.

Bibliografía

[1] Niladri Syam and Arun Sharma. Waiting for a sales renaissance in the fourth industrial revolution: Machine learning and artificial intelligence in sales research and practice. *Industrial marketing management*, 69:135–146, 2018.

[2] John D Owens, Mike Houston, David Luebke, Simon Green, John E Stone, and James C Phillips. Gpu computing. *Proceedings of the IEEE*, 96(5):879–899, 2008.

[3] Sparsh Mittal and Jeffrey S Vetter. A survey of cpu-gpu heterogeneous computing techniques. *ACM Computing Surveys (CSUR)*, 47(4):1–35, 2015.

[4] Robert R Schaller. Moore's law: past, present and future. *IEEE spectrum*, 34(6):52–59, 1997.

[5] Charles E Leiserson, Neil C Thompson, Joel S Emer, Bradley C Kuszmaul, Butler W Lampson, Daniel Sanchez, and Tao B Schardl. There's plenty of room at the top: What will drive computer performance after moore's law? *Science*, 368(6495):eaam9744, 2020.

[6] Jack Choquette, Wishwesh Gandhi, Olivier Giroux, Nick Stam, and Ronny Krashinsky. Nvidia a100 tensor core gpu: Performance and innovation. *IEEE Micro*, 41(2):29–35, 2021.

[7] Anne C Elster and Tor A Haugdahl. Nvidia hopper gpu and grace cpu highlights. *Computing in Science & Engineering*, 24(2):95–100, 2022.

[8] Stefano Markidis, Steven Wei Der Chien, Erwin Laure, Ivy Bo Peng, and Jeffrey S Vetter. Nvidia tensor core programmability, performance & precision. In *2018 IEEE international parallel and distributed processing symposium workshops (IPDPSW)*, pages 522–531. IEEE, 2018.

[9] Kefa Rabah. Convergence of ai, iot, big data and blockchain: a review. *The lake institute Journal*, 1(1):1–18, 2018.

[10] Lila Rajabion, Abdusalam Abdulla Shaltooki, Masoud Taghikhah, Amirhossein Ghasemi, and Arshad Badfar. Healthcare big data processing mechanisms: The role of cloud computing. *International Journal of Information Management*, 49:271–289, 2019.

[11] Suhua Lei, Huan Zhang, Ke Wang, and Zhendong Su. How training data affect the accuracy and robustness of neural networks for image classification, 2019.

[12] Peter Mell, Tim Grance, et al. The nist definition of cloud computing. *NIST special publication*, 2011.

[13] Shancang Li, Li Da Xu, and Shanshan Zhao. The internet of things: a survey. *Information systems frontiers*, 17(2):243–259, 2015.

[14] Léon Bottou. Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*, pages 421–436. Springer, 2012.

[15] Ahmet Ali Süzen, Burhan Duman, and Betül Şen. Benchmark analysis of jetson tx2, jetson nano and raspberry pi using deep-cnn. In *2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, pages 1–5. IEEE, 2020.

[16] Zhi Zhou, Xu Chen, En Li, Liekang Zeng, Ke Luo, and Junshan Zhang. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proceedings of the IEEE*, 107(8):1738–1762, 2019.

[17] Johan Barthélemy, Nicolas Verstaevel, Hugh Forehead, and Pascal Perez. Edge-computing video analytics for real-time traffic monitoring in a smart city. *Sensors*, 19(9):2048, 2019.

[18] Shankar Chandrasekaran. Hacer que la gente hable: Microsoft mejora la calidad de la ia y la eficiencia del traductor con nvidia triton. NVIDIA, blog, Mar. 22, 2022 [Online].

[19] Batta Mahesh. Machine learning algorithms-a review. *International Journal of Science and Research (IJSR).[Internet]*, 9:381–386, 2020.

[20] Oludare Isaac Abiodun, Aman Jantan, Abiodun Esther Omolara, Kemi Victoria Dada, Nachaat AbdElatif Mohamed, and Humaira Arshad. State-of-the-art in artificial neural network applications: A survey. *Heliyon*, 4(11):e00938, 2018.

[21] Yu-chen Wu and Jun-wen Feng. Development and application of artificial neural network. *Wireless Personal Communications*, 102(2):1645–1656, 2018.

[22] Andrinandrasana David Rasamoelina, Fouzia Adjailia, and Peter Sinčák. A review of activation function for artificial neural network. In *2020 IEEE 18th World Symposium on Applied Machine Intelligence and Informatics (SAMI)*, pages 281–286. IEEE, 2020.

[23] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[24] David H Hubel and Torsten N Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*, 148(3):574, 1959.

[25] Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do, and Kaori Togashi. Convolutional neural networks: an overview and application in radiology. *Insights into imaging*, 9(4):611–629, 2018.

[26] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 international conference on engineering and technology (ICET)*, pages 1–6. Ieee, 2017.

[27] Naila Murray and Florent Perronnin. Generalized max pooling. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2473–2480, 2014.

[28] Y-Lan Boureau, Jean Ponce, and Yann LeCun. A theoretical analysis of feature pooling in visual recognition. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 111–118, 2010.

[29] Matthew D Zeiler and Rob Fergus. Stochastic pooling for regularization of deep convolutional neural networks. *arXiv preprint arXiv:1301.3557*, 2013.

[30] Abien Fred Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018.

[31] Bolin Gao and Lacra Pavel. On the properties of the softmax function with application in game theory and reinforcement learning. *arXiv preprint arXiv:1704.00805*, 2017.

[32] William H Sandholm. *Population games and evolutionary dynamics*. MIT press, 2010.

[33] David E Rumelhart, Richard Durbin, Richard Golden, and Yves Chauvin. Backpropagation: The basic theory. *Backpropagation: Theory, architectures and applications*, pages 1–34, 1995.

[34] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[35] Zhaoyang Niu, Guoqiang Zhong, and Hui Yu. A review on the attention mechanism of deep learning. *Neurocomputing*, 452:48–62, 2021.

[36] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45, 2020.

[37] Yutong Xie, Jianpeng Zhang, Chunhua Shen, and Yong Xia. Cotr: Efficiently bridging cnn and transformer for 3d medical image segmentation. In *International conference on medical image computing and computer-assisted intervention*, pages 171–180. Springer, 2021.

[38] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[39] Michal Filipiuk and Vasu Singh. Comparing vision transformers and convolutional nets for safety critical systems. In *SafeAI@ AAAI*, 2022.

[40] Hangbo Bao, Li Dong, and Furu Wei. Beit: Bert pre-training of image transformers. *arXiv preprint arXiv:2106.08254*, 2021.

[41] Iclr | 2022. `https://iclr.cc/Conferences/2022`. Accessed: 2022-09-25.

[42] Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 535–541, 2006.

[43] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*, 2017.

[44] Matrix compression operator. `https://blog.tensorflow.org/2020/02/matrix-compression-operator-tensorflow.html`. Accessed: 2022-11-16.

[45] Tara N Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6655–6659. IEEE, 2013.

[46] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129(6):1789–1819, 2021.

[47] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning? *Proceedings of machine learning and systems*, 2:129–146, 2020.

[48] Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.

[49] Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing rewinding and fine-tuning in neural network pruning. *arXiv preprint arXiv:2003.02389*, 2020.

[50] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[51] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference. *arXiv preprint arXiv:2103.13630*, 2021.

[52] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[53] Paul Merolla, Rathinakumar Appuswamy, John Arthur, Steve K Esser, and Dharmendra Modha. Deep neural networks are robust to weight binarization and other non-linear distortions. *arXiv preprint arXiv:1606.01981*, 2016.

[54] Rishabh Goyal, Joaquin Vanschoren, Victor Van Acht, and Stephan Nijssen. Fixed-point quantization of convolutional neural networks for quantized inference on embedded platforms. *arXiv preprint arXiv:2102.02147*, 2021.

[55] Torchserve. `https://pytorch.org/serve/`. Accessed: 2022-09-23.

[56] Tensorflow serving. `https://www.tensorflow.org/tfx/guide/serving`. Accessed: 2022-09-23.

[57] Nvidia triton server. `https://developer.nvidia.com/nvidia-triton-inference-server`. Accessed: 2022-09-23.

[58] Zhigang Zhou, Hongli Zhang, Xiaojiang Du, Panpan Li, and Xiangzhan Yu. Prometheus: Privacy-aware data retrieval on hybrid cloud. In *2013 Proceedings IEEE INFOCOM*, pages 2643–2651. IEEE, 2013.

[59] Tyson Condie, Neil Conway, Peter Alvaro, Joseph M Hellerstein, Khaled Elmeleegy, and Russell Sears. Mapreduce online. In *Nsdi*, volume 10, page 20, 2010.

[60] Multi model server. `https://github.com/awslabs/multi-model-server`. Accessed: 2022-09-23.

[61] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, et al. Mlperf inference benchmark. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 446–459. IEEE, 2020.

[62] Atam P Dhawan. *Medical image analysis*. John Wiley & Sons, 2011.

[63] Robert H Choplin, JM Boehme 2nd, and CD Maynard. Picture archiving and communication systems: an overview. *Radiographics*, 12(1):127–129, 1992.

[64] M Elon Gale and Daniel R Gale. Dicom modality worklist: an essential component in a pacs environment. *Journal of digital imaging*, 13(3):101–108, 2000.

[65] Putu Wuri Handayani, Achmad Nizar Hidayanto, Ave Adriana Pinem, Ika Chandra Hapsari, Puspa Indahati Sandhyaduhita, and Indra Budi. Acceptance model of a hospital information system. *International journal of medical informatics*, 99:11–28, 2017.

[66] WD Bidgood Jr and Steven C Horii. Introduction to the acr-nema dicom standard. *Radiographics*, 12(2):345–355, 1992.

[67] Mario Mustra, Kresimir Delac, and Mislav Grgic. Overview of the dicom standard. In *2008 50th International Symposium ELMAR*, volume 1, pages 39–44. IEEE, 2008.

[68] Dicom conformance statements. `https://global.agfahealthcare.com/dicomconformance/`. Accessed: 2022-09-28.

[69] Interoperability: Dicom. `https://www.gehealthcare.com/products/interoperability/dicom`. Accessed: 2022-09-28.

[70] Dicom: The standard foundation for imaging & image management. `https://www.siemens-healthineers.com/es/services/it-standards/dicom`. Accessed: 2022-09-28.

[71] About dicom: Overview. `https://www.dicomstandard.org/about-home`. Accessed: 2022-09-28.

[72] JORGE HERNANDO RIVERA PIEDRAHITA, WALTER SERNA SERNA, and JUAN PABLO TRUJILLO LEMUS. Descripción del estándar dicom para un acceso confiable a la información de las imágenes médicas. *Scientia et technica*, 16(45):289–294, 2010.

[73] Ohif viewer. `https://ohif.org/`. Accessed: 2022-09-28.

[74] Micro dicom. https://www.microdicom.com/. Accessed: 2022-09-28.

[75] Osirix dicom viewer. `https://www.osirix-viewer.com/`. Accessed: 2022-09-28.

[76] Postdicom. `https://www.postdicom.com/es`. Accessed: 2022-09-28.

[77] Sante dicom viewer. `https://www.santesoft.com/win/sante-dicom-viewer-pro/sante-dicom-viewer-pro.html`. Accessed: 2022-09-28.

[78] D Muthukumaran, K Umapathy, and S Omkumar. Health cloud—health care as a service. In *Next Generation of Internet of Things*, pages 489–497. Springer, 2021.

[79] George C Kagadis, Christos Kloukinas, Kevin Moore, Jim Philbin, Panagiotis Papadimitroulas, Christos Alexakos, Paul G Nagy, Dimitris Visvikis, and William R Hendee. Cloud computing in medical imaging. *Medical physics*, 40(7):070901, 2013.

[80] Yasunari Shiokawa, Noriko Mori, Takaya Sakusabe, Takeshi Imai, Hiroshi Watanabe, and Michio Kimura. Medical image sharing in japan. *Journal of Digital Imaging*, 35(4):772–784, 2022.

[81] Thomas Sterling, Maciej Brodowicz, and Matthew Anderson. *High performance computing: modern systems and practices*. Morgan Kaufmann, 2017.

[82] Kevin Dowd and Charles Severance. High performance computing. 2010.

[83] David A Patterson. Reduced instruction set computers. *Communications of the ACM*, 28(1):8–21, 1985.

[84] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017.

[85] Sparsh Mittal. A survey on optimized implementation of deep learning models on the nvidia jetson platform. *Journal of Systems Architecture*, 97:428–442, 2019.

[86] Model analyzer. `https://github.com/triton-inference-server/model_analyzer`. Accessed: 04-09-2022.

[87] Performance analyzer. `https://github.com/triton-inference-server/server/blob/main/docs/user_guide/perf_analyzer.md`. Accessed: 24-11-2022.

[88] Benchmarking triton (tensorrt) inference server for transformer models. `https://blog.salesforceairesearch.com/benchmarking-tensorrt-inference-server/`. Accessed: 04-09-2022.