Instituto Tecnológico de Costa Rica
Escuela de Ingeniería Mecatrónica

Diseño de un modelo de inferencia energéticamente eficiente de un Vision Transformer implementado en FPGA para su aplicación en sistemas embebidos.

Informe de proyecto de graduación para optar por el título de Ingeniero en Mecatrónica con el grado académico de Licenciatura

Gabriel Eduardo Blanco Mora

2018149147

Cartago, febrero de 2025

Declaro que el presente Proyecto de Graduación ha sido realizado enteramente por mi persona, utilizando y aplicando literatura referente al tema e introduciendo conocimientos propios.

En los casos en que he utilizado bibliografía, he procedido a indicar las fuentes mediante las respectivas citas bibliográficas.

En consecuencia, asumo la responsabilidad total por el trabajo de graduación realizado y por el contenido del correspondiente informe final.

Cartago, Costa Rica
Febrero, 2025

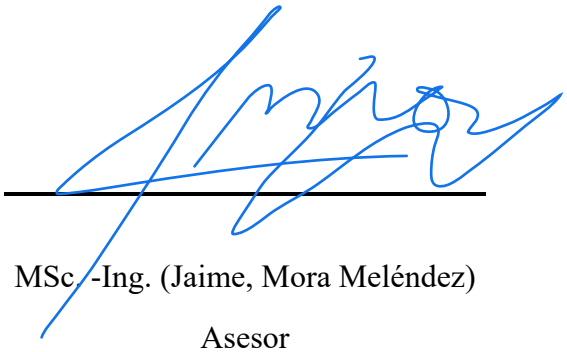
Gabriel Eduardo Blanco Mora
Céd: 117880024

INSTITUTO TECNOLÓGICO DE COSTA RICA
PROGRAMA DE LICENCIATURA EN INGENIERÍA MECATRÓNICA
PROYECTO FINAL DE GRADUACIÓN
ACTA DE APROBACIÓN

El profesor asesor del presente trabajo final de graduación, indica que el documento presentado por el estudiante cumple con las normas establecidas por el programa de Licenciatura en Ingeniería Mecatrónica del Instituto Tecnológico de Costa Rica para ser defendido ante el jurado evaluador, como requisito final para aprobar el curso Proyecto Final de Graduación y optar así por el título de Ingeniero(a) en Mecatrónica, con el grado académico de Licenciatura.

Estudiante: Gabriel Euardo Blanco Mora

Proyecto: Diseño de un modelo de inferencia energéticamente eficiente de un Vision Transformer implementado en FPGA para su aplicación en sistemas embebidos



MSc. -Ing. (Jaime, Mora Meléndez)

Asesor

Cartago, febrero de 2025

INSTITUTO TECNOLÓGICO DE COSTA RICA
PROGRAMA DE LICENCIATURA EN INGENIERÍA MECATRÓNICA
PROYECTO FINAL DE GRADUACIÓN
ACTA DE APROBACIÓN

Proyecto final de graduación defendido ante el presente jurado evaluador como requisito para optar por el título de Ingeniero(a) en Mecatrónica con el grado académico de Licenciatura, según lo establecido por el programa de Licenciatura en Ingeniería Mecatrónica, del Instituto Tecnológico de Costa Rica.

Estudiante: Gabriel Eduardo Blanco Mora

Proyecto: Diseño de un modelo de inferencia energéticamente eficiente de un Vision Transformer implementado en FPGA para su aplicación en sistemas embebidos

Miembros del jurado evaluador

Ing. Juan Luis Crespo Marino
Jurado

Ing. Ronald Loaiza Baldares
Jurado

Los miembros de este jurado dan fe de que el presente proyecto final de graduación ha sido aprobado y cumple con las normas establecidas por el programa de Licenciatura en Ingeniería Mecatrónica.

Cartago, febrero de 2025

Resumen

En el presente informe se expone el diseño e implementación de un modelo de inferencia energéticamente eficiente basado en el algoritmo de clasificación de imágenes conocido como “Vision Transformer”. Este modelo fue implementado en una FPGA y está orientado a aplicaciones de sistemas embebidos que demandan alta eficiencia energética.

El proceso de diseño inició con la identificación de las necesidades del cliente, lo cual permitió establecer los objetivos del sistema diseñado. A partir de estas necesidades se tomaron las decisiones pertinentes durante el desarrollo del modelo con el fin de garantizar los requerimientos en términos de precisión, eficiencia energética y tiempo de inferencia.

La optimización energética del modelo de inteligencia artificial se logró mediante técnicas de computación aproximada, específicamente a través de la cuantización directa de pesos y activaciones. En este caso, se utilizó la cuantización tipo FP16, implementada por medio del software Vitis 2023.2, lo que permitió realizar inferencias con mayor eficiencia energética y una pérdida de la precisión nula.

El correcto funcionamiento del diseño fue validado tanto mediante simulaciones realizadas en Vitis como mediante la toma de datos de manera directa al implementar el modelo optimizado en una FPGA. Los resultados obtenidos confirmaron que el sistema diseñado cumple con los requerimientos.

Palabras clave: FPGA, Cuantización, High Level Synthesis, eficiencia energética, precisión, Floating point 16.

Abstract

This report presents the design and implementation of an energy-efficient inference model based on the image classification algorithm known as “Vision Transformer.” This model was implemented on an FPGA and is aimed at embedded systems applications that require high energy efficiency.

The design process began with identifying the client's specific needs, which allowed the system's objectives to be defined. Based on these needs, the necessary decisions were made during the development of the model to ensure compliance with requirements in terms of accuracy, energy efficiency, and inference time.

The energy optimization of the artificial intelligence model was achieved through approximate computing techniques, specifically by directly quantizing the model's weights and activations. FP16 quantization was used, and it was implemented making use of Vitis 2023.2 software. Inference with higher energy efficiency and negligible loss of precision were achieved.

The design's correct functionality was validated through simulations carried out in Vitis and direct data collection by implementing the optimized model on an FPGA. The results confirmed that the designed system meets the client's requirements.

Keywords: FPGA, Quantization, High Level Synthesis, energy efficiency, precision, Floating Point 16

Dedicatoria

Este proyecto está dedicado a mis padres Carlos Eduardo Blanco y Valeria Mora, mis hermanas Maripaz Blanco y Daniela Blanco, mi abuelita Mari y mi novia Catalina Muñoz, sin su apoyo y persistencia este proyecto de graduación no hubiese sido posible.

Agradecimiento

Agradezco profundamente a Dios por la oportunidad de estudiar y todas las bendiciones que me ha dado. También quiero brindar mi más sincero agradecimiento a mis mentores durante este proyecto Luis León Vega y Jaime Mora Meléndez, así como a mi compañero del laboratorio ECAS Lab. Sin su apoyo incondicional y constante la confección de este proyecto no hubiese sido posible.

Lista de Contenido

Lista de figuras

Lista de Tablas

Lista de Abreviaciones

1. ECAS lab ..... Efficient Computing Across the Stack Lab
2. FPGA ..... Field Programmable Gate Array
3. CPU..... Central Processing Unit
4. TPU..... Tensor Processing Unit
5. GPU..... Graphics Processing Unit
6. DNN..... Deep Neural Networks
7. FAL..... Flexible Accelerator Library
8. CNN..... Convolutional Neural Networks
9. ADAS..... Advanced Driver Assistance systems
10. NLP..... Natural Language Processing
11. ViT..... Vision Transformer
12. MLP ..... Multilayer Perceptron
13. EW ..... Element-Wise
14. IA ..... Inteligencia Artificial
15. GeLU..... Gaussian Error Linear Unit
16. HDL ..... Harware Description Language
17. RTL..... Register Transfer Logic
18. HLS..... High Level Synthesis
19. IP..... Intelectual Property
20. IOT..... Internet of things
21. HW..... Hardware
22. SW..... Software
23. LUTs ..... Look-Up Tables
24. DSPs..... Digital signal Processors
25. FFs..... Flip-Flops
26. ASICs..... Application Specific Integrated Circuits
27. PTQ..... Post-training Quantization
28. QAT ..... Quantization-Aware Training
29. FP ..... Floating Point
30. FPS..... Frames per second
31. NaN..... Not a number
32. EMAC..... Exact multiply and Accumulate
33. BRAM..... Block Random access memory
34. SD ..... Sistema diseñado

1. Introducción

1.1Entorno del proyecto

El proyecto se llevó a cabo en conjunto con el ECAS Lab (Efficient Computing Across the Stack Lab) perteneciente a la escuela de ingeniería electrónica del Instituto Tecnológico de Costa Rica. Este laboratorio se dedica a estudiar la optimización energética de modelos computacionales de inteligencia artificial desde una perspectiva holística, es decir, analiza métodos para obtener una mayor conservación de energía desde varias aristas. Se analiza como perfeccionar tanto el Software como el Hardware, y también como utilizar el Edge y el "Cloud Computing" en armonía para obtener el mejor rendimiento, junto al menor gasto de recursos posible [1] [2].

El laboratorio y sus investigadores se han enfocado en el desarrollo de distintos tipos de aceleradores de Hardware para operaciones que se utilizan frecuentemente en algoritmos de inteligencia artificial. Para la implementación de estos aceleradores de Hardware, las FPGAs se presentan como una alternativa más energéticamente amigable a otros tipos de Hardware, como las CPUs, TPUs (Tensor processing Unit) o las GPUs, las cuales poseen una demanda exorbitante en la industria hoy en día, y además no son muy eficientes en términos energéticos.

Por otra parte, el proyecto se realizó bajo la supervisión directa del MHPC Luis G. León Vega. En la actualidad, el máster León se encuentra en la Universidad de Trieste en Italia trabajando en su tesis doctoral [3]. León trabaja activamente con el ECAS Lab y su tesis de maestría consistió en el desarrollo de una serie de aceleradores de Hardware genéricos para redes neuronales profundas (DNN). Además, generó un "framework" denominado FAL ("Flexible Accelerator Library") el cual se enfoca en facilitar la implementación de nuevos aceleradores de Hardware.

Uno de los algoritmos que se han implementado utilizando estos aceleradores de Hardware en FPGA son los "Vision Transformers". Este algoritmo de inteligencia artificial sigue prácticamente al pie de la letra al famoso modelo de transformador que utilizan los "Large Language Models", con la diferencia de que la entrada del modelo será una imagen en lugar de una serie de palabras. Los llamados "Vision Transformers" han demostrado ser una alternativa altamente competitiva, y en muchos casos superior, a las clásicas redes neuronales convolucionales (CNN). Como se afirma en [4], esta arquitectura obtiene resultados de precisión muy similares a las CNN del estado del arte, pero requiere de un tiempo de entrenamiento mucho menor.

Aunque hasta el momento la investigación del máster Luis León ha obtenido resultados satisfactorios en términos de eficiencia energética y precisión, existen varias técnicas de

computación aproximada que podrían llegar a reducir aún más los recursos energéticos necesarios para la inferencia de estos algoritmos. [5]

1.2 Descripción del problema

La mayoría de los modelos de inteligencia artificial que en la actualidad están revolucionando a la sociedad utilizan computación en la nube debido a los altos requerimientos computacionales y energéticos que poseen, no solo para el proceso de entrenamiento, sino también para el de inferencia [6]. En ciertas aplicaciones implementadas en sistemas embebidos, como por ejemplo los sistemas avanzados de asistencia al conductor (ADAS), no se tiene esta posibilidad.

Los sistemas ADAS son de naturaleza crítica y, en la mayoría de las ocasiones, no pueden emplear la nube para procesar los datos debido a la posibilidad de encontrarse en zonas con conectividad limitada o a la necesidad de operar en tiempo real [1]. Por lo tanto, es necesario recurrir a la computación en el borde (Edge Computing). Además, en este contexto, es fundamental optimizar el consumo de energía durante el procesamiento de datos mediante inteligencia artificial para prolongar la duración de la batería y aumentar el alcance entre cargas.

Al reducir los recursos computacionales necesarios para procesar los datos en el borde, se incrementa el rango de los vehículos y se permite que los algoritmos se ejecuten en computadoras de menor costo. Esto contribuye tanto a la democratización como al acceso generalizado de estas tecnologías. Además, se reduce la huella de carbono asociada al funcionamiento de estos sistemas. Sin embargo, al disminuir los recursos energéticos, es común que se pierda precisión en el modelo, lo cual es poco deseable en este contexto donde la precisión es crítica, ya que de ello depende la seguridad e integridad humana.

Actualmente, los circuitos integrados que suelen utilizarse para implementar las redes neuronales son GPUs (Graphic processing Units). Aunque, debido a su capacidad de paralelización, las GPUs son en principio candidatas ideales para implementar algoritmos de inteligencia artificial, estas consumen altos niveles de energía. Además, en la actualidad, la oferta de este producto no es suficiente para la enorme demanda que poseen a nivel mundial. Por estas razones, este tipo de Hardware no es ideal para las aplicaciones relacionadas a sistemas embebidos que no tienen acceso a computación en la nube [3]. Un dispositivo que surge como candidato para solucionar esta problemática energética es el Field Programmable Gate Array (FPGA),

principalmente las de gama baja. Las FPGAs ofrecen un mayor rendimiento por unidad de energía consumida debido a su arquitectura configurable y la versatilidad en su implementación.

La tesis doctoral del MHPCH Luis G. León Vega se enfoca en el desarrollo de aceleradores de Hardware para operaciones matemáticas de uso común en DNN. En el marco de esta investigación, se presenta la problemática de que el proceso de inferencia de los algoritmos basados en atención aún no posee niveles de eficiencia energética ideales, lo cual es fundamental para las aplicaciones de ADAS. Este Proyecto toma como referencia el trabajo que ya ha sido realizado por el Máster Luis León Vega y optimiza el consumo energético a un grado aún mayor utilizando técnicas de computación aproximada.

1.3 Síntesis del problema

Las alternativas actuales para la aceleración de redes neuronales basadas en sistemas de atención del ECAS Lab implementadas en FPGA no obtienen niveles ideales de eficiencia energética, lo cual conlleva a dificultades de implementación en dispositivos con recursos energéticos restringidos.

1.4 Objetivos

1.4.1 Objetivo General

Desarrollar un acelerador energéticamente eficiente del proceso de inferencia de un Vision Transformer para su uso en sistemas embebidos con restricciones energéticas.

1.4.2 Objetivos Específicos

- Analizar el estado actual de los aceleradores de Hardware diseñados en el ECAS Lab con el fin de identificar oportunidades de mejora en términos energéticos.
- Diseñar una optimización del modelo Vision Transformer utilizando técnicas de computación aproximada para disminuir el gasto energético, manteniendo la precisión en las operaciones de inferencia
- Implementar en FPGA la optimización del modelo Vision Transformer diseñada.
- Validar que el modelo implementado mantiene la precisión de clasificación de imágenes al realizar la inferencia de un set de datos relacionado a sistemas de conducción autónoma.

1.5 Estructura del documento

En el presente documento se describe detalladamente el proceso de diseño de un modelo de inferencia energéticamente eficiente de un Vision Transformer implementado en FPGA. El documento se divide en 7 secciones principales: Introducción, Marco Teórico, Marco Metodológico, Diagnóstico, Diseño de la solución, Resultados y Análisis y Conclusiones y Recomendaciones. A continuación se describe brevemente en que consiste cada uno de estos capítulos.

La introducción corresponde al primer capítulo y en ella se describen generalidades del proyecto tales como su contexto y entorno. Se describe de manera breve la organización con la cual se realiza el proyecto y cuáles son sus objetivos principales. También se presenta el problema de investigación y se determinan los objetivos que se buscan alcanzar para presentar una solución ante esta problemática.

El Marco Teórico define los conceptos ingenieriles que son necesarios tener claros para comprender el desarrollo del proyecto y las decisiones de diseño que se toman. Se tratan temas del funcionamiento interno y la arquitectura de los transformadores de visión, FPGAs y High Level Synthesis, y computación aproximada. Además se describe de manera detallada los distintos tipos de cuantización y los resultados del estado del arte con estos tipos de cuantización.

El Marco metodológico describe y aplica la metodología de diseño de productos conocida como Ulrich-Epinger. Se mencionan también las distintas etapas que conforman esta metodología, que corresponden a: Planeación, Identificación de las necesidades del cliente, establecimiento de las especificaciones del producto, selección de conceptos candidatos y selección del concepto ganador.

El cuarto capítulo se denomina Diagnóstico y detalla el estado actual de los aceleradores de HW diseñados por el ECAS Lab y como han sido implementados hasta el momento en el modelo de Inteligencia Artificial ViT. En este capítulo también se muestran el rendimiento actual del modelo con los aceleradores en términos de precisión de clasificación, cantidad de recursos computacionales, potencia disipada y tiempo de inferencia.

Posteriormente en el capítulo de Diseño de solución se detalla como se diseña e implementa la cuantización en el modelo con el fin de mejorar su eficiencia energética al mismo tiempo que se mantiene la precisión. Se exponen todas las decisiones de diseño tomadas y el razonamiento

ingenieril con el cual se llegó a estas decisiones. Además se expone un estudio de histogramas con el cual se tomaron decisiones de diseño.

El capítulo de resultados y análisis muestra el rendimiento del modelo ViT con la cuantización diseñada implementada. Primero se describen las pruebas de validación diseñadas y luego se muestran los resultados de cantidad de recursos computacionales utilizados, potencia disipada, latencia y precisión del modelo. Finalmente se evalúa si estos resultados cumplen con los requerimientos iniciales del cliente y se analiza las razones por la cuales estos requerimientos fueron alcanzados o no. El capítulo final de conclusiones y recomendaciones se resumen los resultados del proyecto y se dan recomendaciones generales para futuros trabajos relacionados.

2. Marco Teórico

2.1 Transformadores de Visión

En el contexto de la inteligencia artificial, los transformadores consisten en un tipo de arquitectura de red neuronal profunda que introduce el mecanismo de atención. Este algoritmo fue introducido en el año 2017 por un equipo de investigadores de Google en el "paper" "Attention is all you need" y fue diseñado originalmente con el objetivo de destacarse en el área de procesamiento de lenguaje natural (NLP). El mecanismo de atención revolucionó el mundo de la inteligencia artificial ya que permite al modelo enfocarse en distintas partes de una secuencia de entrada de manera flexible. Esto se debe a que el modelo no se basa en un procesamiento secuencial, sino en uno en paralelo que busca relaciones e identifica dependencias complejas entre distintas secciones de la información que debe procesar [6].

En el año 2021 se introduce un "paper" denominado "An Image is worth 16x16 words". Esta investigación presenta un algoritmo que se basa fundamentalmente en la arquitectura del transformador, pero introduce unos cuantos cambios para poder enfrentarse a la tarea de clasificación de imágenes sin perder la capacidad de entendimiento de contexto que poseen los transformadores. Los Transformadores de Visión (ViT por su siglas en inglés presentan ciertas ventajas con respecto a otros algoritmos de visión de computadora, como por ejemplo las redes neuronales convolucionales. La principal de ellas es que, en tareas de clasificación de imágenes, el ViT puede alcanzar precisiones y resultados muy similares o ligeramente superiores que los otros algoritmos de vanguardia, pero requiere significativamente menos recursos computacionales en el proceso del preentrenamiento. Los ViT alcanzan estas precisiones especialmente cuando se entrenan con grandes conjuntos de datos [4]. A continuación, se explica a detalle la arquitectura de los ViT.

Los Vision Transformers (ViT) constan de cuatro secciones o módulos principales: "Patch Embedding", "Attention", "Multilayer Peceptron (MLP)" y "Unembedding". En el Patch Embedding, la imagen se divide en pequeños parches (patches), que se codifican en vectores de alta dimensionalidad, y se añade un "class token" que se utilizará para la clasificación final. En la fase de Attention, el modelo intercambia contexto entre los patches, permitiendo una comprensión global de la imagen. Luego, en la MLP, estos vectores se refinan a través de capas densas para extraer características más complejas. Finalmente, en la fase de Unembedding, se analiza el class token, que se multiplica por una matriz de pesos para producir un vector de probabilidades, que

determina la clase a la que pertenece la imagen. En la Figura 1 se muestra la representación de la arquitectura del ViT que fue presentada en el “paper” original [4]. A continuación, se explica a fondo cada uno de estos módulos del ViT, así como de las operaciones matemáticas que los componen.

2.1.1 Patch Embedding:

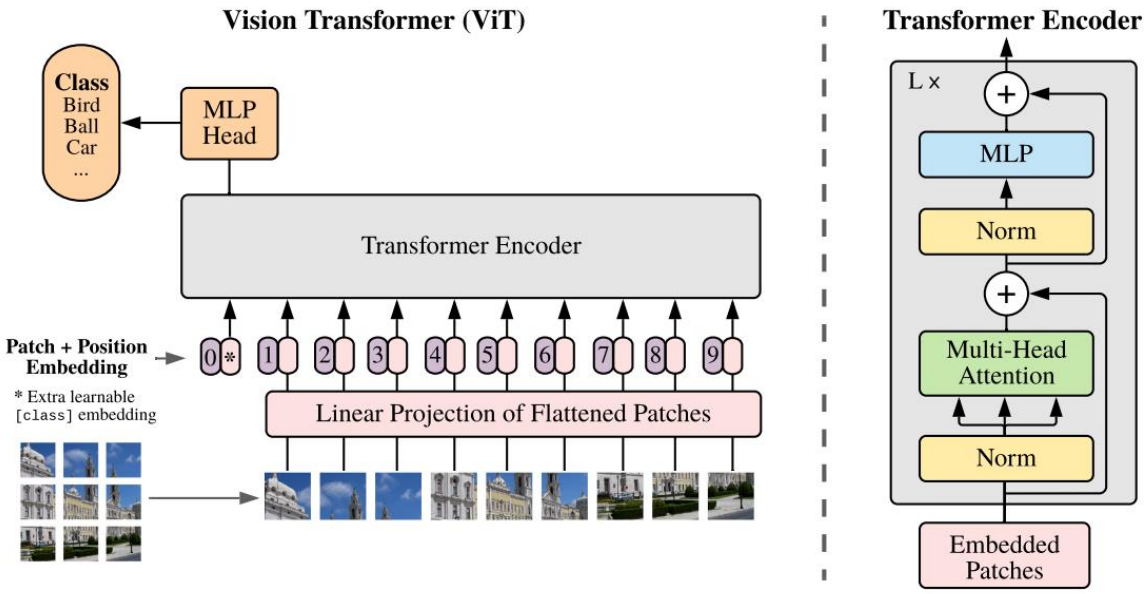


Figura 1. Arquitectura de un Vision Transformer

En la Figura 1 se ilustra la arquitectura y las operaciones matemáticas del módulo "Patch Embedding". En esta, así como en las siguientes figuras, cada rectángulo simboliza una operación matemática del modelo, las flechas continuas indican el tensor de entrada a medida que avanza por todas las operaciones, y las líneas discontinuas corresponden a los tensores de parámetros entrenables, como los pesos y bias del modelo. En este primer módulo del ViT, la imagen se divide en patches de un tamaño determinado, que son simplemente secciones cuadradas de la imagen original. Al ingresar al modelo, la imagen se representa como un tensor con forma [HxWxC], donde H y W se refieren a las dimensiones de la imagen, y C corresponde al número de canales, que comúnmente es 3 en una representación RGB.

Esta imagen primero pasa por una operación de convolución que cumple con dos funciones principales. En primer lugar, al configurar un tamaño de “Kernel” y un “Stride” iguales a las dimensiones deseadas de los patches, y no aplicar “Padding”, la convolución tiene la capacidad de separar la imagen en patches. En segundo lugar, los valores del Kernel son entrenados para que, simultáneamente con la separación en patches, el modelo genere los “embeddings” correspondientes a cada patch. Un embedding corresponde a una representación numérica de cada patch que capta sus características en un espacio de altas dimensionalidad.

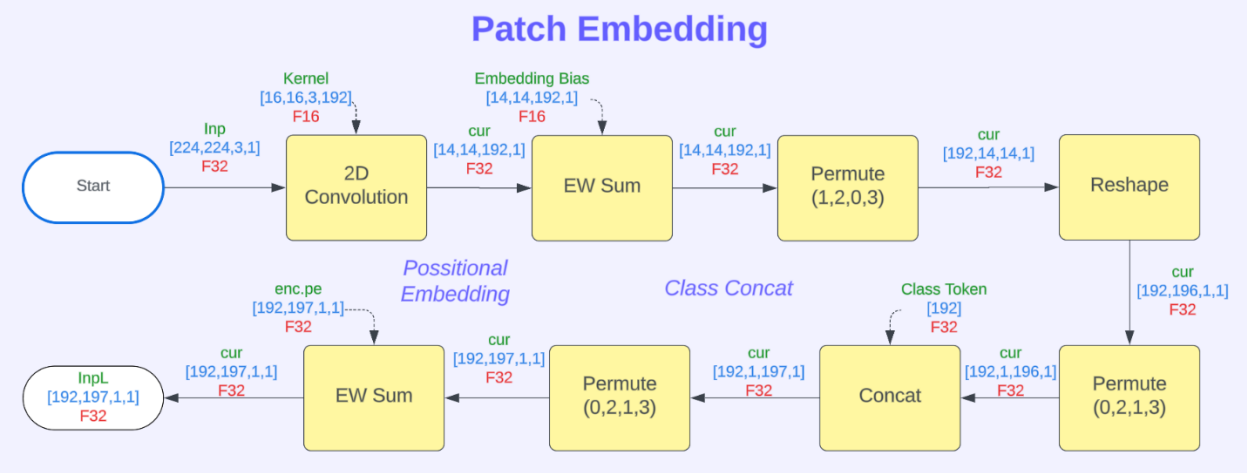


Figura 2. Arquitectura del primer módulo del ViT: Patch Embedding

A continuación, se suma un término de Bias mediante una operación EW a cada patch, lo cual ajusta sus activaciones permitiendo al modelo capturar mejor las características de cada uno. Es importante mencionar que, a lo largo del proceso, se realizan operaciones como por ejemplo “permutes” o “reshapes” que no serán explicadas en este marco teórico, ya que estas solo se encargan de ajustar la forma de los tensores para que tengan el formato indicado para las operaciones matemáticas.

Luego, el class Token se concatena al tensor de entrada. Este Token es fundamental en los Vision Transformer porque representa una visión global o un tipo de “resumen” de todas las características todos patches. Durante el proceso de atención, el class token interactúa con todos los demás patches y genera una visión del contexto o el significado de la imagen entera. Como se muestra en la Figura 2 , la inicialización de sus valores se realiza durante el proceso de entrenamiento.

Finalmente, por medio de una operación de suma “element-wise” (EW), el tensor de positional embedding se une al tensor de la imagen. A diferencia de los transformadores aplicados

en Natural Language Processing (NLP), los valores del tensor positional embeding son completamente generados durante el proceso de entrenamiento, no se usa ningún tipo de codificación senoidal, como sí se hace en NLP. La función primordial del positional embedding es proporcionar información sobre la estructura espacial de cada patch al ViT, es decir, permite que el ViT entienda en qué posición de la imagen original se encuentra cada patch.

Como se observa en la Figura 2, la salida del módulo de Patch Embedding se copia en un tensor etiquetado como InpLL. Este tensor se reutiliza al final del módulo de atención, donde se suma a la salida de la capa de atención. Conceptualmente, este tensor representa una codificación numérica de cada parche de la imagen original, es decir, una lista de valores que encapsula tanto las características visuales como la ubicación espacial de cada sección de la imagen. Esta representación se ha formado a través de su interacción con los parámetros entrenables de la red, proporcionando una base para el procesamiento posterior en el modelo.

2.1.2 Módulo de atención:

El módulo de atención es fundamental en los transformadores, ya que es quien permite que la entrada sea analizada de manera íntegra y completa, tomando en cuenta las relaciones y interacciones entre cada token y el resto, lo cual permite tener una idea contextualizada y global de los datos que se están analizando. En otras palabras, el mecanismo de autoatención permite que cada token de la entrada considere y brinde contexto a todos los demás tokens logrando captar relaciones complejas entre ellos de manera eficiente. En los enfoques y algoritmos de IA anteriores esto no era posible, ya que las relaciones entre los elementos de entrada estaban limitadas a sus vecinos inmediatos. El lograr entender un contexto global es lo que ha permitido que este algoritmo de IA sea tan poderoso y sea capaz de enfrentarse a tareas que hace unos años era inimaginables para modelos de IA.

Para entender esta función del mecanismo de autoatención, se puede analizar el siguiente ejemplo enfocado en NLP. Considérese, la palabra “banco”. Esta palabra puede tener varios significados, y el significado que corresponde es comúnmente dado por el contexto. En la oración “Me senté en el banco” su significado se refiere a un asiento, pero en la oración “Deposité dinero en el banco” su significado es el de una institución financiera. En ambos casos, la definición de la palabra está dado por los tokens cercanos, sin embargo, si consideramos un texto más largo que finaliza con la oración “Me dirigí al banco”, el significado de la palabra será uno si al inicio del texto dice “Me encontraba caminando en el parque” y otro si dice “Necesitaba pedir un préstamo”. Este ejemplo subraya la importancia de entender la relación entre dos tokens que se encuentran lejanos entre sí, algo que el mecanismo de autoatención en los transformadores puede hacer muy efectivamente.

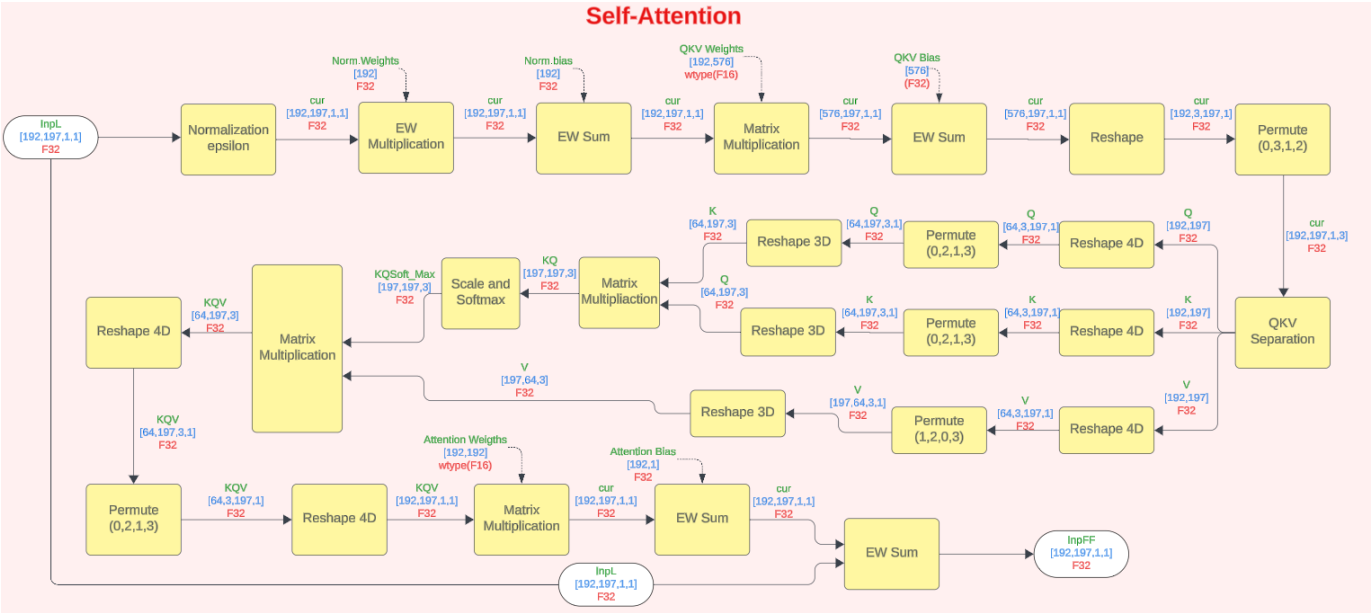


Figura 3. Arquitectura del Segundo módulo del ViT: Self-Attention

Se puede también plantear un ejemplo similar para el caso de los ViT. Se tienen dos imágenes que deben ser clasificadas de maneras distintas, pero que ambas tienen un patch muy similar o prácticamente igual. Por ejemplo, un patch que contiene un círculo rojo, puede corresponder al centro de la bandera de Japón o a un semáforo dependiendo del contexto. Aunque en ambos casos tenemos un círculo rojo, el significado de cada uno de estos es completamente distinto. Por esta razón, la capacidad del ViT para analizar relaciones globales entre los distintos patches es tan poderosa.

La Figura 3 muestra la representación gráfica del módulo de atención. Primero, la entrada de este módulo, que corresponde a la salida del módulo de Patch Embedding pasa por un proceso de normalización. Este proceso ajusta los datos del tensor de la imagen para que tengan un promedio de 0 y una varianza unitaria. Esta normalización se repite varias veces a lo largo de toda la arquitectura del ViT, y es fundamental para brindar estabilidad al modelo porque reduce la probabilidad de que los gradientes se vuelvan demasiado grandes o pequeños. Sin ella, los valores de los tensores podrían crecer o disminuir exponencialmente. En esta primera normalización del módulo de atención, se logra además que las características en la entrada tengan una escala consistente, al igual que las similitudes entre los vectores "Key" y "Query". A grandes rasgos, el proceso de normalización acondiciona la entrada para que el modelo de atención pueda procesarla de una manera más eficiente y precisa.

Posteriormente, la entrada pasa por una etapa que se asemeja mucho a una capa neuronal densa, en la cual se realiza una multiplicación EW entre esta entrada y un tensor de pesos entrenados, seguida de una suma EW con un bias entrenado. Básicamente, esta sección termina de acondicionar el tensor de entrada mediante una transformación lineal que termina de ajustar las activaciones antes de que pasen a las operaciones de atención. A continuación, el tensor de entrada se multiplica con tres matrices de pesos entrenados W_Q, W_K y W_V, y se suman tres bias b_Q, b_K y b_V, lo cual genera las matrices Key, Query y Vector K, Q y V, tal como se muestra en las ecuaciones 1, 2 y 3

$$Q = X * W_Q + b_c \qquad (1)$$

$$K = X * W_K + b_K \qquad (2)$$

$$V = X * W_V + b_V \qquad (3)$$

Las matrices Query y Key se relacionan entre sí para que el modelo pueda la relación entre dos patches y cuál es la "intensidad" de esta relación, es decir, que tanto atiende un patch específico a otro. El tipo de relación entre patches se codifica en estas dos matrices. Al multiplicar la Matriz de entrada X por la matriz de pesos W_Q, el resultado se puede interpretar como una "pregunta" que busca entender como varía el significado de cada token en relación con los otros. Por otra parte, la

matriz K que se obtiene al multiplicar la entrada X por la matriz de pesos W_K, actúa como la "respuesta" o "llave" que responde a esa pregunta. Entender exactamente qué preguntas y respuestas codifican Q y K puede ser muy complejo para los humanos, sin embargo, como muchos otros modelos de Inteligencia artificial, esto realmente no es de nuestro interés. Solamente interesa que el modelo aprenda a través del entrenamiento y minimice la función de costo. Es algo similar a lo que sucede con las redes neuronales convolucionales, donde no siempre es fácil interpretar los mapas de activación, pero sabemos que estas activaciones son cruciales para dotar de "inteligencia" al modelo.

Para relacionar esta "pregunta" Q con su "llave" K se realiza una operación de producto punto entre las dos matrices. Cuanto mayor sea el resultado de esta multiplicación, más fuerte es la atención o la intensidad de la relación entre un token y otro, es decir, más relevante es cada palabra para actualizar el significado de las demás. Desde un punto de vista matemático, esto indica que los vectores Q y K se encuentran más alineados en el espacio multidimensional. Este resultado es luego normalizado por medio de la función "Softmax". Softmax es una función que convierte un conjunto de valores en un conjunto de probabilidades, donde cada valor se escala exponencialmente y luego se normaliza para que la suma total sea 1. Las ecuaciones 4 y 5 muestran los dos procesos matemáticos explicados en este párrafo, donde d_k representa la dimensión de los vectores Q y K y funciona como un factor de escalado para evitar que los valores resultantes sean demasiado grandes. I representa el vector de entrada a la función softmax y n es el número total de valores del vector.

$$KQ_{ij} = \frac{Q_i * K_j^T}{\sqrt{d_k}} \qquad (4)$$

$$softmax(I_i) = \frac{\exp(I_i)}{\sum_{j=1}^{n} \exp(I_j)} \qquad (5)$$

Una vez se ha computado la cantidad de atención que deben prestarse los tokens entre sí, el vector "Value", que se obtiene al multiplicar cada token por la matriz W_V generada en el proceso de entrenamiento. es el encargado de almacenar la información que se transmite de un token a otro.

Es decir, Value contiene las características o datos que el modelo utilizará para actualizar la representación de un token después de haber evaluado la importancia relativa de los demás tokens.

Desde un punto de vista en el espacio multidimensional, cada embedding representa un vector que será modificado para reflejar un token que incorpora el contexto de los demás tokens. La intensidad del cambio en la representación de este token es determinada por el producto punto KQ, mientras que la dirección de la modificación está influenciada por V. Para obtener el vector que se sumará al embedding original y así modificar su significado para incluir el contexto, se realiza una multiplicación entre el vector V y el resultado de aplicar "Softmax" a KQ. La ecuación 6 representa matemáticamente el proceso completo de atención. Esta ecuación se extrae directamente del paper original [6], y de hecho, las ecuaciones de la 1 a la 2 se derivan de 6

$$Attention(Q, K, V) = softmax\left(\frac{Q * K^T}{\sqrt{d_k}}\right) * V \qquad (6)$$

El resultado del bloque de atención se suma al embedding original del patch, que en la Figura 3 está etiquetado como InpLL. De este modo, cada patch se modifica y se genera un nuevo vector que sigue codificando su significado original, pero que ahora también incorpora características del contexto global de la imagen. Antes de continuar con la explicación del próximo módulo del ViT, es importante mencionar que el mecanismo de atención no se limita a una sola operación, sino que se implementan múltiples cabezas de atención que se ejecutan en paralelo. Cada una de estas cabezas analiza distintas características de los patches de la imagen, lo cual permite que se capturen relaciones y contextos variados al mismo tiempo. Este enfoca mejora significativamente la precisión del ViT y su capacidad para captar al mismo tiempo el contexto general de la imagen y sus detalles específicos.

2.1.2 Módulo de Perceptrón multicapa:

La red neuronal tipo perceptrón multicapa (MLP) es un estándar en la academia e industria de la inteligencia artificial y es implementada en la arquitectura de transformador a la salida del módulo de atención. Una MLP se compone de una capa de entrada, una de salida, y una o más capas ocultas. Estas capas se componen de varias neuronas tipo perceptrón, las cuales computan una suma ponderada de todas sus entradas, aplican esta suma ponderada por una función de

activación, y la salida de esta función corresponde a la salida de la neurona [7]. Las funciones de activación corresponden a “una función matemática que determina la salida de una neurona en una red neuronal, permitiendo la introducción de no linealidades en el modelo" [8].

Las redes neuronales MLP son densamente conectadas, lo cual significa que todas las neuronas de una capa están conectadas por medio de sus entradas con todas las neuronas de la capa anterior, y, del mismo modo, su salida se conecta con todas las neuronas de la siguiente [7]. Los parámetros entrenables de las MLP corresponden a los pesos del modelo, los cuales se utilizan para computar la suma ponderada de todas las entradas de una neurona, y los bias, los cuales se suman al resultado de esta suma ponderada. En la Figura 4 se muestra gráficamente las conexiones de una red neuronal MLP y en la ecuación 7 se representa matemáticamente este tipo de red neuronal densa. En esta ecuación, f representa la función de activación, $w_{ij}^{(l)}$ representa la matriz de pesos que conecta la capa l con la capa $l-1$, $a_i^{(l-1)}$ son las salidas o activaciones de la capa $l-1$ y b_j^l es el bias de la neurona j en la capa l.

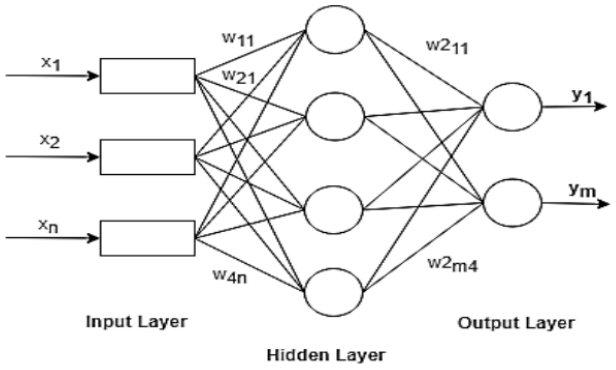


Figura 4. Red neuronal MLP [9]

$$a_j^{(l)} = f\left(\sum_{i=1}^{n} w_{ij}^{(l)}\, a_i^{(l-1)} + b_j^{(l)}\right) \qquad (7)$$

En el caso del ViT, la capa MLP tiene el objetivo de refinar la información que acaba de ser procesada por la capa de atención, la cual se etiqueta en la Figura 5 como InpFF. Una vez el módulo de atención ha captado el contexto de los patches, la MLP es capaz de captar características más complejas y abstractas. Esto se logra ya que la MLP introduce no linealidades al modelo por medio de la función de activación GELU. La función GELU es muy similar a la clásica ReLU, con la diferencia de que suaviza la transición entre valores positivos y negativos, lo que mejora la convergencia y el rendimiento en redes neuronales profundas [10]. La Figura 6 muestra la representación gráfica de la función GELU.

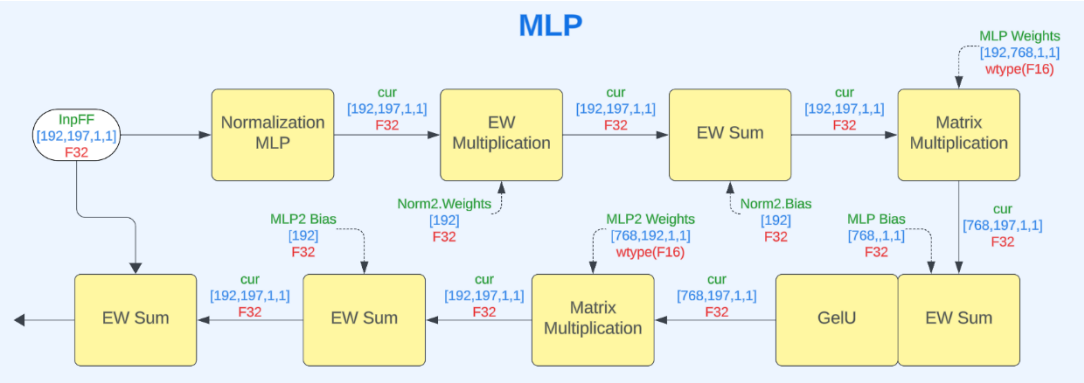


Figura 5. Tercer módulo del ViT: MLP

En el caso del ViT, la MLP consiste en una capa de normalización, y luego una red neuronal con dos capas ocultas, que se conectan entre sí por medio de la función de activación. En resumen, la capa MLP del ViT tiene una función muy similar a cualquier otra MLP que se utiliza en sistemas de visión: captar características espaciales de la imagen. La diferencia radica en que estas MLP utilizan como entrada un tensor que ya ha sido procesado por las capas de atención, lo que significa que contiene la información del patch individual pero también de su contexto.

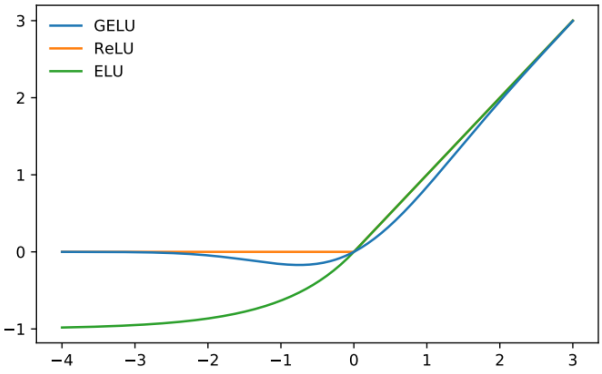


Figura 6. Funciones GeLU, ReLU y ELU.

2.1.4 Patch Unembedding

Finalmente, la salida de la MLP pasa al módulo de “Unembedding”. Este módulo primero realiza la operación “get rows”, la cual extrae del tensor de entrada la columna que representa el class token. Después de que la información ha sido procesada por varios ciclos de módulos de atención seguidos de MLP, el class token ya posee una idea global de todo el contexto de la imagen; codifica que simboliza cada uno de los patches así como la relación entre ellos.

A continuación, este class token es normalizado con el fin de obtener una distribución que tenga media de 0 y una desviación estándar unitaria. Al igual que las normalizaciones anteriores del ViT esto contribuye a la convergencia. Finalmente, el class token es escalado linealmente mediante dos operaciones EW, una multiplicación y una suma que se definen por parámetros entrenables de pesos y bias.

Luego, las siguientes operaciones se encargan de transformar la representación interna del token hacia un espacio en donde cada una de las dimensiones del vector corresponde a una clase específica de clasificación. La multiplicación matricial proyecta el token en el espacio de clases, y la adición del bias ajusta la predicción para cada clase. De esta manera, se obtiene un vector en el cual los valores más elevados representan probabilidades más altas de que la imagen original corresponda a la clase que representa este elemento del vector. Finalmente, se aplica la función Softmax, lo que genera un vector de probabilidades del 0 a 1 de estos valores.

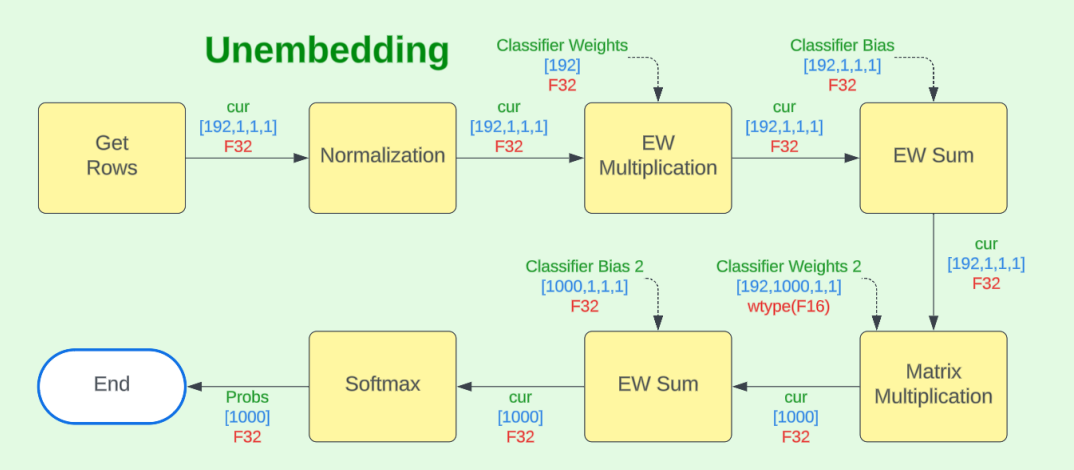


Figura 7. Módulo Final del ViT: Unembedding

2.2 FPGA y High Level Synthesis

2.2.1 FPGA

Las FPGA son dispositivos que consisten en arreglos de bloques de lógica reconfigurables conectados entre sí por medio de cables reprogramables. Estos bloques poseen grandes cantidades de Look-Up Tables (LUTs) y Flip-Flops, lo cual permite que la FPGA implemente cualquier tipo de función lógica o circuito secuencial que se desee. Las FPGAs modernas además incluyen bloques especializados como por ejemplo: multiplicadores, matrices de memoria RAM e incluso pequeños microprocesadores integrados [11].

Las FPGAs son altamente flexibles y reprogramables, debido a que permiten la reconfiguración de Hardware por medio de programación [1]. Por esta razón, las FPGAs son alternativas muy ventajosas para implementar unidades de procesamiento que efectúen tareas específicas y repetitivas. Al diseñar circuitos que no son de propósito general, sino que ejecutan operaciones específicas, se pueden obtener mejoras notables en términos de eficiencia energética y de tiempo de cómputo.

2.2.2 High Level Synthesis

El "High Level Synthesis" es una herramienta que permite la programación directa de módulos de Hardware por medio de lenguajes de alto nivel, como por ejemplo C o C++. Anteriormente, la programación de Hardware se debía realizar directamente por medio de lenguajes de descripción de hardware (HDL) como por ejemplo Verilog o VHDL. Estos HDLs al compilarse generan un archivo de lógica de transferencia de registros (RTL), los cuales describen como los datos se mueven entre registros y cuáles son las operaciones lógicas o aritméticas por medio de las cuales se transforman estos datos [1]. HLS permite a los programadores que tienen poco o nulo conocimiento en diseño de Hardware implementar redes neuronales en FPGAs a un nivel de abstracción mucho más elevado [12].

La reutilización de bloques en FPGAs ha sido posible por medio del uso de propiedades intelectuales (IP). Estos bloques de IP que normalmente están descritos en RTL tienen una gran desventaja, ya que deben ser optimizados manualmente si se desean implementar en un nuevo nodo, es decir, cada vez que estos bloques se ejecutan en una FPGA distinta, se deben de realizar muchos cambios para que el RTL sea compatible con el modelo específico de FPGA [13]. Las herramientas de HLS permiten solucionar este problema, ya que a la hora de compilar código de

alto nivel de abstracción, solo se debe especificar la plataforma objetivo y el archivo RTL para esta plataforma especifica es generado de manera automática [14].

2.3 Computación Aproximada y Cuantización

La computación aproximada es un conjunto de técnicas de cómputo que tienen como objetivo principal alcanzar una disminución en la cantidad de recursos energéticos y computacionales necesarios para ejecutar un algoritmo sin perder la precisión de este, o, perdiéndola en un nivel que no perjudique el objetivo fundamental del algoritmo. En lugar de buscar una ejecución completamente determinista, la computación aproximada tolera disminuciones en precisión y así logra un uso energético mucho más eficiente.

Tareas computacionales que han tomado mucha importancia en los últimos años como lo son el procesamiento de imágenes, audio y video, reconocimiento, minería de datos, y especialmente redes neuronales profundas, tienen la característica en común de que, con frecuencia, un resultado perfecto no es necesario, y un resultado aproximado menos es suficiente [15]. Las técnicas de computación aproximada se pueden aplicar en los distintos niveles de abstracción, a nivel de Software, a nivel de datos, a nivel de arquitectura e inclusive a nivel de circuitos. Algunas de las técnicas a nivel de datos son: Estructuras de datos aproximadas, muestreo de datos, compresiones y reducciones dimensionales y cuantización [16]. En la siguiente sección se profundiza sobre esta última técnica y su aplicación en DNN implementadas en FPGA.

La cuantización se refiere al “proceso de reducir la precisión de los datos numéricos en un programa, mapeando los valores a un conjunto más pequeño de valores discretos” [16]. La cuantización se utiliza comúnmente en modelos de aprendizaje automático para reducir los recursos computacionales y de memoria que requiere el modelo, lo cual es muy importante para la implementación de estos modelos en computación en el borde con recursos limitados. La mayoría de los estudios recientes relacionados a cuantización en el contexto de modelos de inteligencia artificial se han enfocado principalmente en el proceso de inferencia en lugar de en el de entrenamiento [16].

La elección del método de cuantización a implementar en modelos de aprendizaje automático conlleva siempre un equilibrio o “trade-off” entre la precisión de los cálculos y el uso de recursos computacionales y energéticos. La cuantización reduce significativamente los recursos computacionales al disminuir la cantidad de bits utilizados para representar los datos y parámetros del modelo. Sim embargo, este proceso conlleva un compromiso notable, ya que la reducción de la precisión en la representación de los operandos y operadores puede impactar la exactitud del modelo, y, por ende, su funcionalidad. Por ejemplo, utilizar valores enteros de baja precisión en 4 bits en lugar de la representación de punto flotante en 32 bits tiene la capacidad de reducir la latencia y el uso de memoria en un factor de hasta 16x [17].

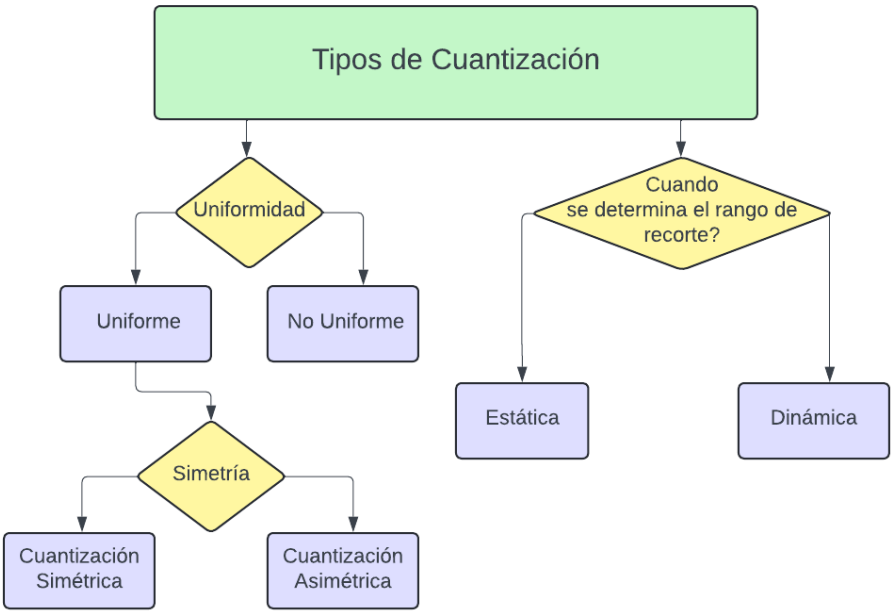


Figura 8. Representación gráfica de los tipos de cuantización

En aplicaciones en las cuales es necesario que los resultados sean altamente precisos, como por ejemplo, en sistemas médicos de detección de enfermedades o en sistemas de seguridad, se prefiere utilizar métodos de cuantización más precisos que por ende utilizan más recursos computacionales, como la cuantización de 16 bits entera (INT16) o en punto flotante reducido (bfloat16) o incluso en Punto flotante de 16 bits (FP16). Por otro lado, en las aplicaciones en las que la precisión no es crítica, y más bien se prefiere un bajo consumo energético, como por ejemplo dispositivos del internet de las cosas (IOT), sensores, o dispositivos embebidos de hogares inteligentes, se opta por cuantizaciones más agresivas como INT8, INT4 o inclusive en casos con requerimientos energéticos críticos cuantización binaria.

A continuación, se presenta la clasificación de categorías de cuantización. Estas categorías son utilizadas en la actualidad en el contexto de aceleradores de HW para modelos de DNN. En la Figura 8 se exponen esta clasificación de manera gráfica, según su división con respecto a simetría, momento de determinación del rango de recorte y uniformidad.

2.3.1 Cuantización Uniforme

La cuantización uniforme toma un set de datos y mapea cada uno de estos a un espacio discreto en el cual cada uno de los valores son equidistantes entre sí. Este tipo de cuantización se implementa en aplicaciones en las cuales se busca simplicidad en los niveles de cuantización y esta simplicidad no afecta en gran medida a la calidad de la señal, como por ejemplo en ciertos sistemas de comunicación [17].

La cuantización uniforme es óptima para distribuciones en las cuales los valores se encuentran distribuidos de manera equitativa a través de todo el rango de datos. En estos casos, los niveles de cuantización equitativamente espaciados resultan ser suficientes para representar los datos sin causar grandes errores de cuantización [18]. En la Figura 9 se compara la cuantización uniforme y la no uniforme de manera gráfica. En este caso específico, la cuantización no uniforme que se ilustra posee mayor precisión cuando los valores a cuantizar se encuentrn más cercanos a cero.

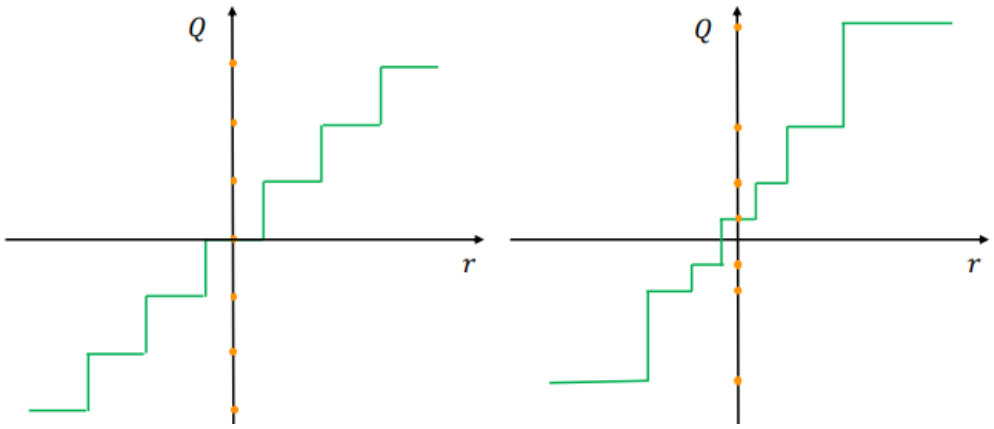


Figura 9. Comparación gráfica entre cuantización uniforme (izquierda) y no uniforme (derecha) [17]

La cuantización uniforme se puede describir matemáticamente como se muestra en las ecuaciones 8 y 9. En la ecuación 8, x representa el valor flotante a cuantizar y Q es la función de

cuantización, α es el valor mínimo del rango de recorte o "clipping range". Por otra parte, S representa el factor de escalado, que de hecho corresponde a la magnitud de los intervalos de cuantización, y Int corresponde a una función de redondeo. Además, en la ecuación 9, β es el valor mayor del rango de recorte y b representa el número de bits utilizados para la cuantización [17]. El punto cero o "zero point" Z se presenta en la ecuación 10, este se refiere al valor cuantizado que corresponde al valor cero real. Este valor toma especial importancia al explicar la cuantización simétrica y asimétrica en la siguiente sección

$$Q(x) = Int\left(\frac{x-\alpha}{S}\right) = Int\left(\frac{x}{S}\right) - Z \qquad (8)$$

$$S = \left(\frac{\beta-\alpha}{2^b-1}\right) \qquad (9)$$

$$Z = Int(\frac{-\alpha}{S}) \qquad (10)$$

El rango de recorte corresponde a el intervalo de valores dentro del que los datos se recortan, evitando que los valores atípicos o extremos que se encuentran fuera de ese intervalo sean representados en el proceso de cuantización. El rango de recorte es de hecho un parámetro del proceso de cuantización, es decir, el diseñador del algoritmo de cuantización elije estos valores como más convenga. En general, un rango de recorte más elevado permite que los valores extremos sean truncados, pero puede reducir la precisión en la representación de los valores intermedios y aumentar el uso de recursos computacionales.

2.3.2 Cuantización Simétrica y Asimétrica

Como se mencionó en el párrafo anterior, el rango de recorte es un parámetro del proceso de cuantización. Al proceso de elegir los valores de β y α se le conoce como calibración. La práctica más común es utilizar el mínimo y máximo del set de datos como rango de recorte. En este caso, se produce una asimetría en el proceso de cuantización, ya que es probable que el valor de α no sea igual al valor de $-\beta$. La cuantización asimétrica normalmente genera un rango de recorte más ajustado, y es comúnmente utilizada cuando los datos presentan asimetría, como por ejemplo en las activaciones a la salida de funciones como ReLU o Softmax [17].

Por otro lado, la cuantización simétrica se puede considerar como un caso "simplificado" de la cuantización asimétrica, en la cual el "zero point" Z corresponde a cero [19]. Este tipo de cuantización es más utilizado cuando los datos se distribuyen de manera simétrica, como por

ejemplo en distribuciones que poseen un comportamiento gaussiano. En este caso, el valor mínimo del rango de recorte α sí es igual al valor de $-\beta$. La decisión más popular en este tipo de cuantización corresponde a elegir la magnitud de β y α (que sería la misma) en función de la magnitud máxima de los valores del set de datos, tal como se muestra en la ecuación 11.

$$-\alpha = \beta = \max(|r_{max}|, |r_{min}|) \qquad (11)$$

La cuantización simétrica posee la ventaja de que, debido a que el valor Z siempre corresponde a cero, se obtiene una reducción del costo computacional de la inferencia en DNN, y además hace que la implementación en Hardware sea mucho más directa y sencilla. Al ser $Z = 0$, en la ecuación 8 se observa que el HW solamente tendría que dividir el valor real entre el factor de escalado y redondear el resultado para obtener el valor cuantizado. De hecho, "el cálculo de pesos por medio de cuantización asimétrica incurre en un gasto extra de energía del 10% al 15%" [20]

En la Figura 10 se muestra la representación gráfica de la cuantización simétrica y asimétrica. En esta se puede apreciar como el valor de cero real es mapeado a cero en la cuantización simétrica y mapeado a Z en la cuantización asimétrica.

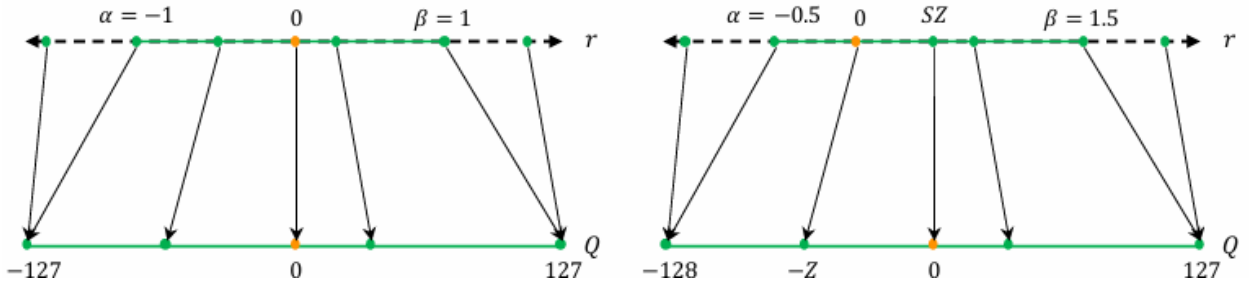


Figura 10. Mapeo de la cuantización simétrica (izquierda) y asimétrica (derecha) [17]

2.3.3 Cuantización Dinámica y Estática

Otra división de los diversos tipos de cuantización se basa en cuando se determinan los parámetros de cuantización, como por ejemplo el rango de recorte, el punto cero o el número de bits. Cuando un modelo de inteligencia artificial es entrenado, sus pesos son determinados y no cambiarán de valor a la hora de utilizar el modelo para realizar inferencia. Por otro lado, las entradas del modelo de IA sobre las cuales se realiza la inferencia cambian constantemente, por lo que todas las activaciones del modelo también. En la cuantización estática, después de entrenar el

modelo, se determinan los parámetros de cuantización, y estos se mantienen estáticos al realizar la inferencia de distintas entradas [17].

Un método popular para realizar cuantización estática consiste en elegir una muestra de las posibles entradas del modelo, y obtener las activaciones de estas muestras. Estas activaciones se computan y, basándose en los valores de todas estas, además de en los valores de los pesos del modelo, se determinan los parámetros de cuantización. Esto significa que el punto cero, el rango de recorte y demás parámetros no varían cada vez que el modelo realiza inferencia a una entrada distinta. Este método resulta en menor exactitud comparado con la cuantización dinámica pero utiliza menos recursos computacionales.

Por otra parte, la cuantización dinámica ajusta de manera cíclica los parámetros de cuantización para cada una de las entradas. De manera automática, el modelo estudia la distribución de los valores de cada una de las entradas y optimiza los parámetros de cuantización para obtener el mayor grado de precisión posible [17]. En este caso, se incurre en una carga computacional adicional o "overhead" porque se requiere computo en tiempo real de las características estadísticas del set de datos, como por ejemplo de su mínimo, máximo, promedio y desviación estándar, y además optimizar los parámetros de cuantización.

Si se busca realizar la implementación del modelo de IA en HW programable como FPGA, esta técnica también introduce varios inconvenientes, como por ejemplo la necesidad de lógica adicional que puede aumentar significativamente la complejidad del diseño y requerir de más recursos lógicos como LUTs, Flip-Flops o DSPs, lo cual a su vez incrementa el consumo energético. También la dificultad de implementación aumenta porque hay que tomar en cuenta que se deben modificar los factores de escalado a medida que los datos atraviesan el pipeline de la FPGA.

2.3.4 Cuantización no Uniforme

Como se mencionó brevemente en la sección 2.3.1, la cuantización no uniforme permite que los niveles de cuantización no estén espaciados de manera uniforme entre sí. El objetivo principal de este tipo de cuantización es utilizar niveles de cuantización más pequeños, lo cual equivale a un mayor nivel de precisión, en los rangos de la distribución que posean una mayor densidad de datos. Esto permite que, para una cantidad de bits específica, en comparación a la cuantización uniforme, la cuantización no uniforme capture de manera más precisa o leal la

distribución de datos original. La cuantización no uniforme es la ideal, por su misma naturaleza, para representar sets de datos que no poseen una distribución uniforme.

Matemáticamente, la cuantización no uniforme se puede describir mediante la ecuación 12. Esta ecuación es muy similar a la ecuación 8, con la importante diferencia de que el factor de escalado en este caso no es una constante, sino otra función que depende del valor a cuantizar "x". El valor más común para $S(x)$ corresponde a una función logarítmica, sin embargo hay muchas funciones distintas que han sido implementadas con éxito para cuantizar un set de datos.

$$Q(x) = Int\left(\frac{x - \alpha}{S(x)}\right) \qquad (12)$$

Como se ha mencionado en los párrafos anteriores, la cuantización no uniforme permite representar de manera más precisa cualquier set de datos que no tenga una distribución completamente uniforme, siempre y cuando la función de factor de escalamiento sea elegida de manera correcta. A pesar de esto, la cuantización no uniforme posee una desventaja muy notable. Este tipo de cuantización es difícil de implementar en HW de computación general, como las GPUs y las CPUs [17]. Esto se debe a que este tipo de HW está diseñado y optimizado para realizar taras que involucren operaciones uniformes y acceso regular a la memoria. La necesidad de realizar cálculos con datos no uniformes y acceso a memoria irregular introduce ineficiencias que son difíciles de tratar en este tipo de HW. La capacidad de paralelización y versatilidad de HW programable como FPGAs o ASICs pueden ser opciones viables para implementar este tipo de cuantización.

2.3.4 PTQ (Post-Training Quantization) vs. QAT (Quantization Aware Training)

En todos los esquemas de cuantización explicados hasta el momento, se ha supuesto que este proceso ocurre después del entrenamiento de la red neuronal, lo cual se conoce como cuantización después del entrenamiento o PTQ por sus siglas en inglés. "Los algoritmos PTQ toman una red pre entrenada en punto flotante de 32 bits y la convierten directamente en una red de punto fijo sin necesidad del proceso de entrenamiento original. Estos métodos pueden o no requerir datos adicionales o necesitar un pequeño conjunto de calibración, que frecuentemente es de fácil acceso" [21].

Este método posee varias ventajas, como por ejemplo que el "overhead" o sobrecarga computacional es muy baja al compararse con QAT [17]. Además, este método es más práctico ya

que se puede aplicar en situaciones en las cuales el set de datos es limitado o no es de fácil acceso. En este caso, tampoco se necesita acceso al proceso de entrenamiento, lo cual permite que aunque el diseñador de la cuantización no tenga mucho conocimiento en el área del Deep learning, aun así pueda optimizar el modelo en términos de eficiencia.

Por otro lado, el QAT implementa el proceso de cuantización durante el entrenamiento de la red neuronal. En general, el algoritmo busca simular las condiciones de baja precisión numérica a las cuales se enfrentará la red neuronal desde el proceso de entrenamiento. Esto se logra al cuantizar los pesos y las activaciones del modelo en cada iteración de la propagación hacia adelante ("forward-pass") del proceso de entrenamiento. Es importante mencionar que aunque el "forward-pass" es cuantizado, la retro propagación del error se ejecuta con el nivel de precisión más elevado, que normalmente corresponde a F32, lo cual evita acumulaciones de error significativas y permite que el modelo ajuste los pesos con precisión [21].

De este modo, se calcula la función de pérdida en condiciones a las que se verá enfrentado el modelo en el proceso de inferencia. Los parámetros y el tipo de cuantización que utiliza el proceso de entrenamiento son las mismas que el modelo utilizará en el proceso de inferencia. Por ejemplo, si el modelo utilizará en la inferencia cuantización de 4 bits asimétrica con un rango de recorte de [1,5], el "forward-pass" del proceso de entrenamiento también será cuantizado haciendo uso de estos parámetros De manera intuitiva, este tipo de cuantización se puede interpretar como que el modelo "aprende" a manejar la reducción de precisión durante el proceso de entrenamiento [21].

En resumen, el QAT reduce de manera efectiva y notable el impacto de la baja precisión en el rendimiento de la red neuronal, lo cual es de suma importancia para aplicaciones en dispositivos con recursos limitados. Además, los modelos generados por QAT son más robustos a la cuantización, lo cual genera un rendimiento más predecible en el HW destino. Estas mejoras en la precisión, rendimiento y predictibilidad se obtienen a costas de un mayor costo de tiempo y recursos computacionales. El entrenamiento de la red neuronal requiere de mucho más tiempo ya que la cuantización debe ser simulada en cada iteración, lo cual puede ser perjudicial especialmente en modelos grandes. En términos generales, el QAT es una mejor opción en casos en los que tanto una alta eficiencia energética como una precisión muy similar al modelo sin cuantizar son fundamentales.

2.4 Estado del Arte y Cuantización en DNNs implementadas en FPGA

En esta sección se presenta un resumen del estado del arte en cuanto a las técnicas de cuantización empleadas para optimizar la eficiencia energética y preservar la precisión en redes neuronales profundas (DNN) implementadas mediante aceleradores de HW en FPGAs. Es importante destacar que este es un campo en constante evolución; desde 2019 hasta la fecha se han desarrollado nuevas técnicas de cuantización que alcanzan resultados cada vez más prometedores. Algunas de estas técnicas se combinan para obtener mejoras adicionales, impulsando futuras investigaciones. Las técnicas de cuantización abordadas en esta sección serán posteriormente comparadas entre sí, con el objetivo de diseñar una solución basada en una o varias de estas metodologías.

2.4.1 Cuantización INT8

La cuantización de 8 bits corresponde a uno de los tipos de cuantización más comunes en optimización de DNN en FPGAs. Además, es una de las técnicas más sencillas de implementar [22] [23] [24]. Corresponde a una cuantización uniforme, tal como la presente en la ecuación 8. En esta cuantización, se transforman los datos de FP32 a enteros de 8 bits espaciados de manera homogénea. Normalmente, las implementaciones de este tipo de cuantización son PTQ y simétricas [22] [23] [24].

Según el estado del arte, el "secreto" para que la precisión de la red disminuya muy poco con respecto a su contraparte no cuantizada, corresponde en realizar un análisis estadístico de cada capa de la DNN, y según esta distribución de los datos, elegir un rango de recorte apropiado [22] [23] [24]. De esta manera, aunque toda la red se cuantiza utilizando 8 bits, el parámetro de cuantización del rango de recorte, y, en consecuencia, el factor de escalamiento, varían para cada tensor.

Un aspecto crucial que considerar en este tipo de cuantización, y en cualquier otra, son las posibles situaciones de desbordamiento que pueden ocurrir durante la inferencia al cuantizar el modelo. Una de las principales ventajas del formato FP32 es su capacidad, al ser una representación de punto flotante con muchos bits, para representar valores extremadamente grandes. Sin embargo, al cuantizar el modelo usando INT8, pueden surgir limitaciones significativas. Por ejemplo, en una cuantización de 8 bits, el valor máximo que se puede representar

es 127. Si en algún momento una operación suma 125 con 124, el resultado superará 127 y no podrá ser representado correctamente, generando un desbordamiento. Esto puede afectar la precisión y el comportamiento del modelo durante su ejecución.

Por esta razón, los diseños de cuantización implementan distintos algoritmos para lidiar con este tipo de situaciones. Por ejemplo, En el caso de [23] se utiliza saturación fija y saturación escalada. La saturación fija corresponde a que siempre que haya un desbordamiento, ese dato desbordado se recorta al límite del rango, lo cual correspondería en el ejemplo del anterior párrafo a representar el resultado de la suma como 127. La saturación escalada corresponde a multiplicar el valor que se encuentra en el rango por el valor más común del histograma, el cual normalmente se encuentra entre 0 y 1, lo cual generalmente causa que el valor vuelva a estar dentro de los rangos permitidos.

Al implementar una arquitectura que solamente utiliza enteros de 8 bits en todas las operaciones de las redes neuronales, se logran obtener mejoras notables en términos de eficiencia energética, velocidad de inferencia y utilización de recursos computacionales. Esto sucede porque se simplifica el flujo de datos en HW y se evitan conversiones adicionales que serían costosas en términos de latencia [23].

En [22] se obtiene una caída de la precisión en la tarea de clasificación de imágenes de solamente un 0.4% y en tareas de detección de prácticamente un 0%, El tiempo de inferencia disminuye notablemente y se alcanzan 70 inferencias por segundo (FPS) en la implementación en FPGA [22]. A modo de comparación, en la actualidad, CPUs de alta gama logran alcanzar entre 2 a 5 FPS [25]. Este estudio también concluye que, para poder mantener la precisión con cuantizaciones de aún una menor cantidad de bits, como INT2 o INT4, una cuantización QAT sería necesaria. Por otra parte, [23] obtiene una pérdida de precisión mínima en tareas de clasificación: un promedio de 1% para distintas DNN. Al mismo tiempo, se obtienen velocidades de 12 FPS. En [24] también se observa un aumento de la eficiencia energética.

2.4.2 Cuantización de punto fijo

La cuantización de punto fijo toma un número racional y lo transforma en una representación binaria en la cual el primer bit representa el bit de signo (0 para positivo y 1 para negativo), los siguientes "N" bits representan la parte entera del número y los últimos "M" bits la parte fraccionaria de este número. La parte entera se calcula como se muestra en la ecuación 13 y la parte fraccionaria como se muestra en la ecuación 14, en las cuales b_i y b_j corresponden a cada bit de la representación en punto fijo de la parte entera y la parte fraccionaria respectivamente [26].

$$Parte\ Entera = -b_{N-1} * 2^{N-1} + \sum_{i=0}^{N-2} b_i * 2^i \qquad (13)$$

$$Parte\ Fraccional = \sum_{j=1}^{M} b_{-j} * 2^{-j} \qquad (14)$$

Además, en este tipo de cuantización, se tiene un control muy directo del rango y la precisión de la cuantización por medio del control del valor de N y M. Para una cantidad de bits totales específica, si N aumenta y M disminuye, se tendrá un mayor rango de representación pero una menor precisión. Si al contrario, N disminuye y M aumenta, se obtiene un menor rango de representación con una mayor precisión. De esta manera, se puede jugar con la granularidad de la cuantización, y aplicar distintas precisiones a diferentes capas de las redes neuronales dependiendo de la distribución en cada una de estas con el fin de obtener representaciones más fieles de los datos originales [27]. En las ecuaciones 15 y 16 se muestra cómo se calcula el rango de representación y la precisión para una cantidad de bits N y M específica.

$$Precisión = 2^M \qquad (15)$$

$$Rango = [2^{N-1}, 2^{N-1} - 1] \qquad (16)$$

En la literatura se observa que, en ciertos casos, es necesario realizar una cuantización dinámica para obtener resultados de buena precisión y bajo gasto energético [28] [29] [30]. En esta cuantización dinámica se modifica, a tiempo real y de manera granular por capa, la cantidad de bits enteros y de bits fraccionarios tomando en cuenta la distribución estadística de los tensores de las activaciones.

"Para decidir el ancho de bits óptimo necesario para representar los pesos y activaciones, existen dos requisitos contradictorios. El primero es minimizar el error de cuantización, y el segundo es eliminar el desbordamiento de datos en los cálculos. Mientras que se requieren más

bits para reducir el error de cuantización, se prefieren anchos de bits menores para minimizar el desbordamiento de datos" [29]. Hay diversas maneras de calcular estas cantidades de bits óptimas, por ejemplo, [30] utiliza un algoritmo de minimización del error L2 y [29] utiliza un algoritmo condicional que considera la mediana de los datos y otras características estadísticas.

Al utilizar punto fijo, las multiplicaciones y divisiones por potencias de dos se ejecutan como operaciones de bit shifting [28]. Las FPGAs están optimizadas para este tipo de operaciones porque son operaciones lógicas muy simples, que requieren muy pocos recursos y se realizan en un solo ciclo de reloj. Por esta razón, en muchas cuantizaciones de punto fijo, se busca aplicar factores de escala en potencias de dos .

La cuantización de punto fijo, al ser implementada junto con bits de protección para evitar el "overflow", puede obtener un incremento desde 2x hasta 6x en tiempo de ejecución. En este caso, el consumo de ancho de banda también se redujo y se obtuvieron precisiones de clasificación prácticamente iguales a las obtenidas con FP32 [29]. En el caso de [28] se obtuvieron resultados similares, una mejora de 4x en la velocidad de inferencia en comparación con una CPU ARM, con una disminución ligera de precisión del 3% al 5% en tareas de detección de objetos. Además, [30] obtuvo una reducción de almacenamiento a una décima parte en comparación a FP32, también con niveles de precisión bastante similares.

2.4.3 Cuantización Bfloat16

La cuantización bfloat16 es una representación de punto flotante que sigue la estructura clásica de usar un bit para el signo, varios bits para el exponente y otros para la mantisa, al igual que las representaciones FP32 y FP16. Este tipo de cuantización fue desarrollado por Google específicamente para acelerar el procesamiento de redes neuronales profundas en su unidad de aceleración de hardware, la TPU (Tensor Processing Unit), optimizando tanto la precisión como la eficiencia en cálculos intensivos en aprendizaje profundo [31]. En la Figura 11 se muestra la estructura de este tipo de cuantización en comparación a la de FP32 y FP16.

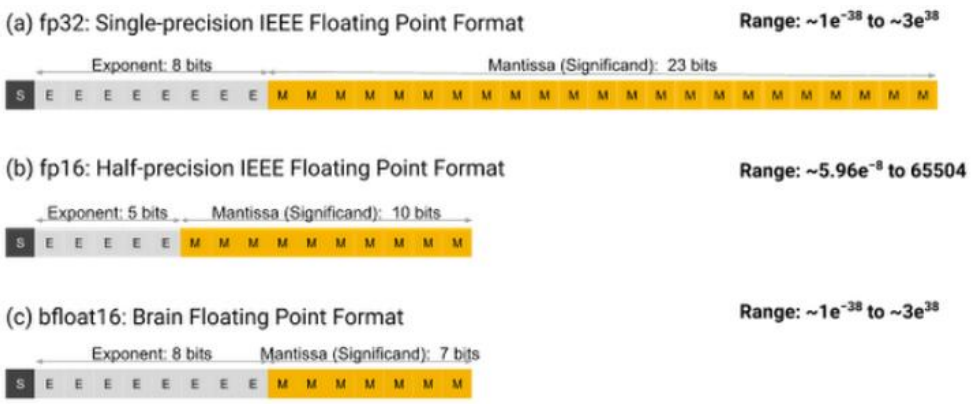


Figura 11. Tres formatos de punto flotante [31]

Como se puede observar en la Figura 11, esta representación mantiene el mismo rango de dinámico que FP32, aunque con precisión reducida y multiplicadores más pequeños [32]. Google descubrió que al tener menos bits en la mantisa, los multiplicadores Bfloat16 son de la mitad del tamaño de un multiplicador FP16 típico y ocho veces más pequeños que un multiplicador FP32. Además, Google también descubrió que las redes neuronales son mucho más sensibles al tamaño del exponente que al de la mantisa, al mantener el exponente igual que en FP32 se manejará de mejor manera las situaciones de subdesbordamiento, desbordamiento y valores NaN. Por lo tanto, Bfloat16 mejora la eficiencia del hardware manteniendo la capacidad de entrenar modelos de aprendizaje profundo precisos [31].

Aunque Bfloat16 fue diseñado originalmente para trabajar en TPUs, existen varias investigaciones que han comprobado la eficiencia y precisión de implementar este tipo de cuantización en FPGAs [33] [34] [35]. Es importante mencionar que en la literatura revisada, normalmente Bfloat16 es implementada como una técnica que realiza la cuantización durante el entrenamiento QAT, y los resultados que se describen en los próximos párrafos se obtuvieron de esta manera.

En el estudio [33] se implementa una red MLP cuantizada en Bfloat16. En este se implementan arreglos sistólicos (arquitecturas de hardware para paralelizar las operaciones matemáticas), ciertas optimizaciones en el producto punto y en el almacenamiento y transporte de datos y “reconocimiento de elementos nulos”. Este reconocimiento identifica y omite valores de cero durante el entrenamiento, evitando operaciones innecesarias y optimizando el rendimiento. De esta manera, se obtiene un reducción de uso de memoria de 50% y de ancho de banda del 35%,

un ahorro de energía de aproximadamente un 50% y una disminución en la precisión de tan solo 2.4% en comparación a FP32.

Por otra parte, la investigación [34] logro obtener por medio de la implementación de este tipo de cuantización en FPGA una disminución de recursos de 1.6x al compararla con FP32 y de 1.33x al compararla con Posit. Además, este estudio afirma que la cuantización bfloat16 obtuvo precisiones muy similares a la precisión original de FP32, pero no tan elevada como la cuantización tipo Posit, la cual se expone en la siguiente sección.

2.4.4 Cuantización Posit

La cuantización Posit es un tipo de cuantización no uniforme que ofrece una precisión adaptable y un rango dinámico optimizado para aplicaciones de cómputo intensivo como las redes neuronales profundas [36]. Para utilizar la cuantización tipo Posit se deben de definir preliminarmente dos parámetros: El número de bits totales y el número de bits de exponente "es", el cual determina el valor de "$useed$" tal como se muestra en la ecuación 17. La ecuación 18 presenta la manera de calcular la cuantización Posit, y la Figura 14 representa ejemplos de esta cuantización para 3, 4 y 5 bits totales con $es = 2$, $useed = 2^{2^{es}} = 16$. La estructura de la representación Posit se presenta en la Figura 12 y consiste en 4 grupos de bits:

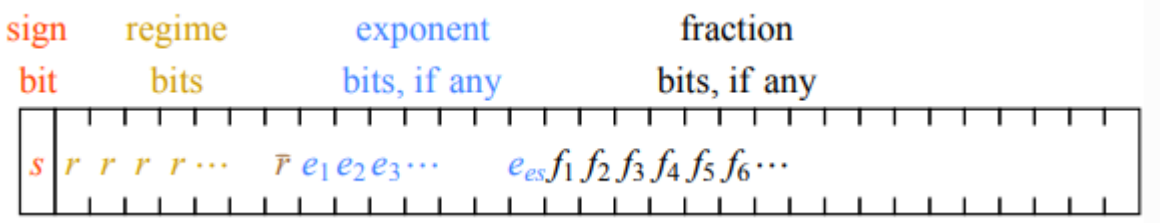


Figura 12. Estructura de bits de cuantización Posit [36]

- Bit de signo: Al igual que muchos otros tipos de cuantización, el primer bit corresponde al bit de signo y un 0 representa a los valores positivos y un 1 a los valores negativos. El bit de signo se muestra en la ecuación 18 como "$sign(p)$". En las figuras estos bits se representan en color rojo
- Bits de régimen: Los bits de régimen corresponden a una secuencia de "1" seguidos de un "0", o a una secuencia de "0" seguidos de un uno. La cantidad de "0" o de "1" consecutivos determina el valor de "k" según se muestra en la Figura 13. En las figuras estos bits se representan de color amarillo y café [36].

- Bits de exponente: También son un factor de escala al igual que los bits de régimen, pero estos funcionan de una manera más "tradicional". El factor de escala de los bits de exponente corresponde simplemente a "2^e", donde "e" corresponde a el valor decimal que representan estos bits en binario. Los bits de exponente no siempre se encuentran presentes, depende de si, después de representar el bit de signo y los bits de régimen, aún quedan bits restantes. En las figuras estos bits se muestran de color azul.
- Bits de fracción o Mantisa: Definen el valor decimal con precisión. Estos bits se comportan de manera igual a como se comportarían en una mantisa de cualquier otro flotante común, por medio de la fracción 1.f. Estos bits, al igual que los bits de exponente, son opcionales y solo estarán presentes si, después de representar el bit de signo, los de régimen y los de exponente, aún quedan bits adicionales.

$$useed = 2^{2^{es}} \tag{17}$$

$$x = \begin{cases} 0, & p = 0 \\ \pm\infty, & p = -2^{n-1} \\ sign(p) * useed^k * 2^e * f, & cualquier\ otro\ p \end{cases} \tag{18}$$

Binary	0000	0001	001x	01xx	10xx	110x	1110	1111
Numerical meaning, k	−4	−3	−2	−1	0	1	2	3

Figura 13. Valor de "k" para distintas combinaciones de bits de régimen

La ventaja principal de la representación tipo Posit es que permite que los números pequeños que se encuentran cercanos a 1 o -1 tengan una precisión muy alta, y al mismo tiempo permite conservar un rango dinámico muy amplio, inclusive igual o mayor a la representación FP32. Todo esto se puede obtener sin un aumento de tamaño y de uso de recursos computacionales y energéticos como al que se tendría que acudir al utilizar FP32 o inclusive FP64.

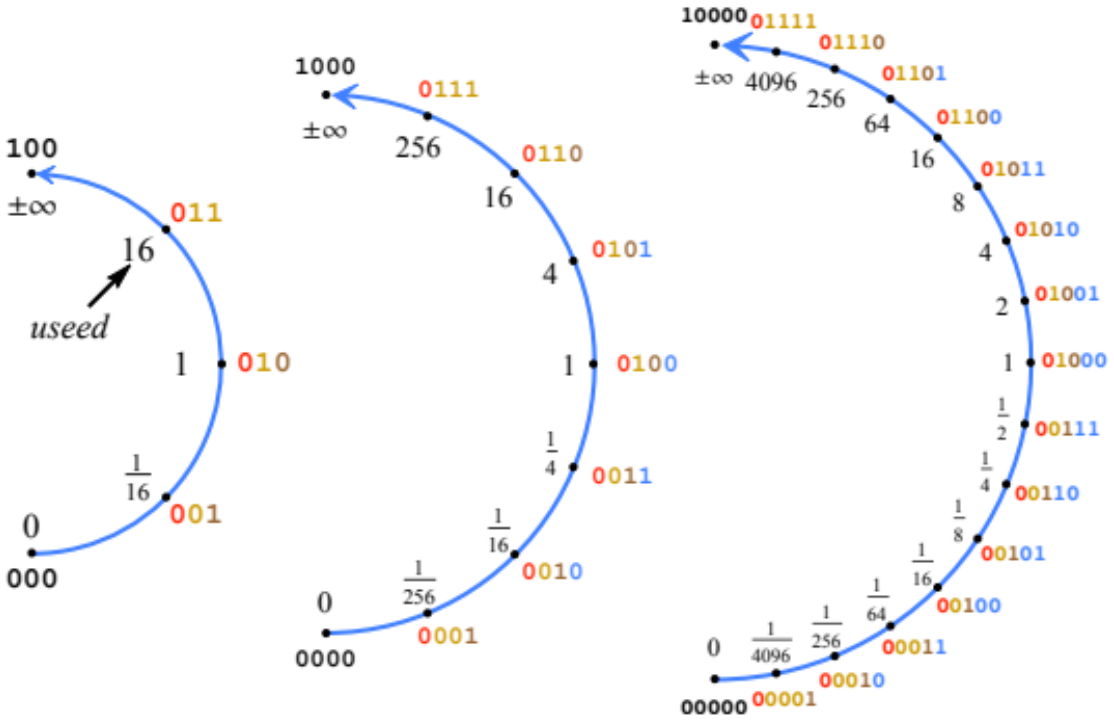


Figura 14. Construcción de posit para 3, 4 y 5 bits con es=2 [36].

Otra característica ventajosa crucial de los Posits al compararlos con FP es la ausencia de valores NaN (Not a number) y valores infinitos, los cuales requieren de HW adicional especializado para su manejo, lo cual agrega carga computacional. Al utilizar la cuantización tipo Posit estos casos excepcionales no se presentan ya que, tal como se puede observar en la Figura 14, la cuantización permite una transición continua hacia los valores extremos, lo cual evita estados inválidos.

Posit ha sido implementado en varios aceleradores de HW en FPGA, obteniendo resultados prometedores. Por ejemplo, en [35] se varía de forma dinámica para cada capa el número de bits de exponente "*es*" según el rango de valores de los datos de entrada, para distribuciones con mayor densidad de valores pequeños se utilizan menos bits de exponente y más bits fraccionarios. En este caso, se obtuvo un 51% de incremento en el uso de recursos energéticos al compararlo con Bfloat16 debido a la complejidad del HW necesario para manejar el régimen y el ajuste dinámico de la mantisa y el exponente, pero un 20% de decremento con respecto a FP32.

Asimismo, en [37] se implementa en la FPGA, junto a la cuantización Posit en 8 bits, un acelerador EMAC (Exact multiply and accumulate) el cual corresponde a una unidad de HW que permite acumular valores sin truncarlos. En este caso, se presentaron perdidas de precisión en el

modelo de 0% a 4%, 30% menos de memoria y latencia y energía similares a la implementación en FP32.

En conclusión, la información de estos dos estudios [35] [37] indica que Posit en 8 bits posee mayor precisión que bfloat 16 y también que representaciones de punto fijo a 8 bits, lo cual lo hace ideal para representaciones de bajos bits que requieren altos niveles de precisión y no requieren tanto ahorro energético, ya que utiliza más energía que bfloat16.

2.4.5 Cuantización Logarítmica

La cuantización logarítmica es un tipo de cuantización en el cual se busca que el conjunto de datos cuantizados corresponda a potencias de dos. Tener un conjunto de datos en el cual todos correspondan a potencias de dos es muy beneficioso ya que, al multiplicar dos de estos datos, la operación a nivel de HW también se puede implementar simplemente como un bit shift. El Bit shift corresponde a simplemente una función de desplazamiento lo cual no requiere de ningún tipo de unidad aritmética compleja como ALUs o multiplicadores. Los bit shift se pueden obtener solamente haciendo uso de pequeños multiplexores o inclusive solamente realizando un enrutamiento de los bits.

La ecuación 19 representa la cuantización logarítmica. El logaritmo base 2 de esta ecuación captura la escala aproximada de los valores originales de una forma eficiente. Posteriormente, este resultado se redondea para poder aproximar el número a una potencia de dos.

$$LogQuant(x) = \begin{cases} Sign(x) * 2^{Round(\log_2|x|)} & x \neq 0 \\ 0, & x = 0 \end{cases} \qquad (19)$$

La literatura muestra que la cuantización logarítmica es ideal cuando se necesita cuantizar una DNN con pocos bits (4 o menos) sin perder grandes cantidades de precisión. La implementación en [35] disminuye significativamente los tamaños de dos modelos de CNN, pasando de 1.9GB a 0.27GB para el modelo de Alexnet y de 4.4GB a 0.97 para el modelo VGG16. En este caso, la cuantización logarítmica a 3 bits disminuyó su precisión en solamente 1.4% para Alexnet y un 0.6% para VGG16 al compararla con la representación original FP32.

2.4.6 Cuantización FP16

Finalmente, la cuantización FP16 es una cuantización de punto flotante de 16 bits. Las cuantizaciones de punto flotante tales como FP32, FP16 o FP64 definen una cantidad de bits para el exponente y una cantidad de bits para la mantisa. El exponente define el rango dinámico del número y la mantisa representa los dígitos significativos, por lo cual define la precisión de los valores.

El número de bits de exponente y número de bits de la mantisa pueden variar según los requerimientos de la representación, sin embargo el estándar IEEE754-2008 [38] define la [38]representación "estándar" de FP16, la cual también se conoce como "half". En esta representación, 1 de los 16 bits se utiliza como bit de signo, 5 bits se utilizan para el exponente y 10 bits para la mantisa. En la ecuación 20 se presenta el funcionamiento de la cuantización FP16. El bias de utiliza para poder representar exponentes positivos y negativos sin necesidad de utilizar un bit de signo exclusivo, y es definido por la cantidad de bits del exponente tal como se muestra en la ecuación 21en la cual "k" representa el numero de bits del exponente En el estándar IEEE754, para k=5, el bias corresponde siempre a 15.

$$-(1)^{signo} * 2^{exponente-bias} * (1 + mantisa) \qquad (20)$$

$$bias = 2^{k-1} - 1 \qquad (21)$$

Como se puede observar, FP16 almacena los datos en un formato similar al que usa FP32, pero al utilizar menos bits, también obtiene una menor precisión y rango dinámico. Al comparar este tipo de cuantización con las explicadas hasta el momento, esta destaca por el balance se utiliza entre rango dinámico y precisión que puede obtener, lo cual la hace ideal para tareas de inferencia de redes neuronales profundas [39], [40], [41], [42]. Aunque este tipo de cuantización no obtiene un ahorro de energía tan elevado como INT8, entre todas las opciones discutidas hasta ahora, F16 es la que cuenta con un nivel mayor de precisión, a excepción de punto fijo si utiliza muchos bits fraccionarios.

En [39] los autores implementan un acelerador de HW en FPGA que utiliza varias técnicas de optimización energética en la red neuronal convolucional VGG16. Las técnicas implementadas corresponden a cuantización de FP16 de todos los pesos y activaciones y modificaciones de arquitectura como un procesador PPMAC (Propagate Partial Multiply-Accumulate) y accesos a memoria tipo Ping Pong. Con estas técnicas, los autores logran obtener una disminución de recursos energéticos de 41.92% con respecto al modelo sin cuantización y sin PPMACs. Además, los autores reportan una mejoría en la precisión top-1 del modelo, pasando de 6.828% a 70.46%

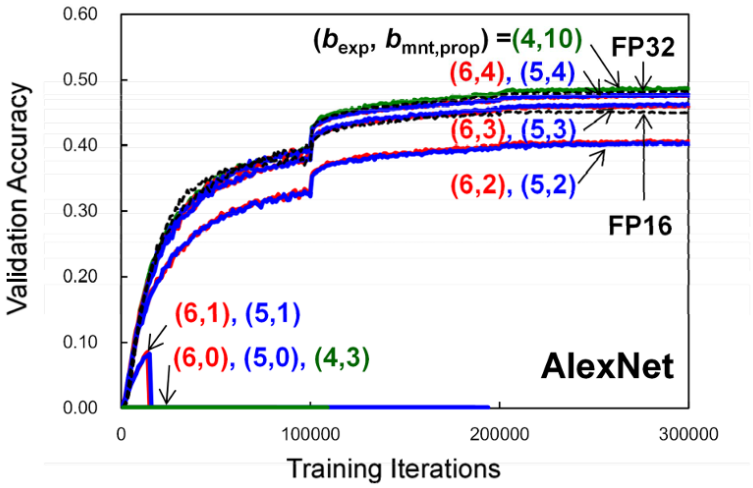


Figura 15. Precisión top 1 de AlexNet para FP32 y FP16 para distintas cantidades de iteraciones [41]

Por otra parte, [41] demuestra, tal como se puede observar en la Figura 15, como la precisión de la red neuronal AlexNet es muy similar al comparar el modelo en FP32 y FP16. Para cantidades no muy altas de iteraciones de entrenamiento, el nivel de precisión es prácticamente el mismo. Para un número más alto de iteraciones más alto, la diferencia en el nivel de precisión entre FP32 y FP16 aumenta pero sigue siendo cercana a 0, de aproximadamente 2 o 3%. [41] también reporta una disminución a aproximadamente un tercio la cantidad de recursos computacionales. Finalmente, en [42] también se muestra una disminución notable, en este caso en un sumador, no una red neuronal profunda completa, en la energía y área de recursos computacionales.

2.5 Normativa en el área de aplicación ADAS

Los sistemas de asistencia de conducción avanzados (ADAS por sus siglas en inglés) corresponden a "tecnologías que automatizan, facilitan y mejoran los sistemas de un vehículo para mejorar y hacer más segura la conducción" [43]. Existen muchos tipos de tecnologías ADAS, como por ejemplo el control crucero adaptativo, reconocimiento de señales de tránsito, detección de peatones, frenos automáticos de emergencia, o inclusive, conducción totalmente autónoma. Algunas de estas tecnologías son diseñadas para mejorar la seguridad del conductor y los pasajeros y otras simplemente para facilitar el proceso de conducción y hacerlo más cómodo [43].

Debido a que es crítico que estos sistemas funcionen de manera confiable en todas las condiciones posibles, porque de ellos depende la seguridad e integridad de los pasajeros, los ADAS están sujetos a estrictas normativas y estándares internacionales. Estas normativas y estándares velan por el correcto funcionamiento de los sistemas en todos los ámbitos, desde el Hardware que ejecuta los algoritmos hasta el Software en sí. Algunas de estas normativas corresponden a la ISO26262 y a la SAEJ3016. Además, la EURO NCAP corresponde a una organización independiente que es reconocida a nivel internacional, que se encarga de evaluar y calificar el nivel de seguridad en los vehículos [44].

ISO26262 corresponde a la principal normativa que aplica para los sistemas ADAS. Esta establece un proceso para identificar riesgos potenciales y los califica en categorías ASIL, donde ASIL D representa el mayor riesgo y ASIL A el nivel de riego más bajo [45]. Además, la normativa establece muchos lineamentos para garantizar la seguridad de estos sistemas, como por ejemplo requisitos a nivel de arquitectura de HW que minimicen la probabilidad de fallas, uso de metodologías específicas para garantizar la fiabilidad del SW diseñado, procesos y métodos específicos de validación y verificación de los sistemas, como pruebas de estrés, y hasta auditorías internas y externas para validar el correcto funcionamiento [46].

Aunque normativas como la ISO26262 no establecen requerimientos específicos cuantitativos con respecto a velocidades de inferencia requeridas, la norma afirma que los sistemas críticos deben responder en intervalos de tiempo que minimicen las posibles consecuencias de fallas ante situaciones peligrosas. Debido a esto, la literatura afirma que, tomando en consideración la velocidad de frenado de un automóvil, y las velocidades que este puede alcanzar bajo condiciones normales, un tiempo de inferencia ideal debe encontrarse entre los 20 a 50ms, y uno aceptable debería de ser al menos menor a 100ms [47].

Con respecto a la precisión de la inferencia en clasificación de imágenes, ISO26262 nuevamente no estipula porcentajes mínimos aceptables, sin embargo sí afirma que los sistemas críticos deben ser validados rigurosamente para minimizar riesgos. Por esta razón, la organización EURO NCAP define que se debe de cumplir con al menos un 70% de precisión en aplicaciones que solamente detectan y avisan al conductor para que este tome decisiones, pero en casos en los cuales el automóvil por sí mismo toma decisiones basándose en los resultados del proceso de inferencia, se buscan precisiones no mayores al 90% [45].

3. Marco Metodológico

La metodología por seguir para el desarrollo del proyecto se basa en la conocida como metodología de desarrollo Ulrich-Epinger [48]. Esta metodología se compone de una serie de etapas, de las cuales, en este proyecto, se seguirán las siguientes: Planeación, Identificación de las necesidades del cliente, establecimiento de las especificaciones del producto, selección de conceptos candidatos y selección del concepto ganador.

3.1 Planeación

En esta primera etapa se estableció un cronograma general para la ejecución del proyecto, en el cual se plantearon actividades específicas para cada uno de los objetivos específicos definidos, y las fechas correspondientes para realizarlas. El establecimiento de este cronograma fue fundamental para poder detectar posibles riesgos y generar planes de mitigación para cada uno de estos.

La metodología Ulrich-Epinger [48] establece que en esta sección es importante realizar un análisis de factibilidad. El análisis de factibilidad se centra en analizar los recursos económicos, temporales, técnicos y tecnológicos que puedan afectar la ejecución del proyecto. En el caso de este proyecto, ya que es de carácter académico e investigativo, el análisis de factibilidad se relaciona profundamente al estudio del estado del arte en el tema. Se analizaron papers en los cuales los autores se enfrentaron a un problema similar al de esta investigación, y se identificaron las tecnologías o técnicas de cuantización se han implementado con éxito. Además, se analizó a que desafíos técnicos o económicos se enfrentaron los investigadores, así como los resultados numéricos de optimización que obtuvieron y las recomendaciones que presentan para futuros estudios similares.

En el caso de este proyecto, se realizó un estudio profundo de 6 técnicas de cuantización distintas implementadas en FPGA para varios tipos de redes neuronales profundas. Se estableció estudiar a fondo al menos tres “papers” distintos para cada técnica de cuantización, los cuales lograron resultados satisfactorios en términos de mejora de tiempo de inferencia, adaptabilidad de la red, precisión y, especialmente, uso de recursos energéticos. Este estudio del estado del arte se presenta en la sección 2.4 de este informe.

Además del estudio académico, en esta sección se analizó cuáles serían los insumos tecnológicos, tanto de SW como de HW que serían necesarios para diseñar e implementar la solución. Como se ha mencionado anteriormente, este proyecto entra en el marco de una investigación más grande que busca diseñar aceleradores de HW en FPGA e incorporarlos en diversas redes neuronales profundas para mejorar su rendimiento. Debido a esto, se realizaron también entrevistas con miembros del grupo de investigación [49] para tener claro que insumos se necesitarían y como se puede obtener acceso a estos. Las principales conclusiones fueron las siguientes:

3.1.1 Insumos de SW necesarios.

La FAL (Flexible Accelerator library) es desarrollada haciendo uso de High Level Synthesis, una herramienta que permite programar HW haciendo uso de lenguajes de alto nivel como C++. La herramienta de diseño de alto nivel que el equipo había utilizado hasta el momento corresponde a Vitis HLS de la empresa Xilinx. El acceso a esta herramienta fue proporcionado por parte del doctor Luis León Vega por medio de conexión remota al servidor denominado “xrp france”. Este servidor además utiliza el sistema operativo de Linux, el cual también es necesario para el desarrollo de este producto ya que el “FAL” fue construida por medio de Linux, por lo que intentar realizar las optimizaciones en algún otro sistema operativo como Windows añadiría complejidad innecesaria al proyecto.

3.1.2 Insumos de HW necesarios.

Con respecto al HW, la ejecución del proyecto necesita principalmente dos dispositivos: Una CPU compatible con Vitis HLS y una FPGA compatible con los aceleradores de HW previamente diseñados. A continuación, se presenta en la Tabla 1 un cuadro comparativo entre las características mínimas de CPU necesarias para ejecutar Vitis HLS y las características de la CPU disponible en el servidor “xrp france”.

Tabla 1. Características necesarias de CPU para ejecutar Vitis HLS y de la CPU del servidor "xrp france" [1].

Requerimientos para ejecutar Vitis HLS	**Características de la CPU "xrp france"**
Procesador Intel o AMD x86-64 de al menos 4 núcleos y 2GHz	Procesador Intel Xeon CPU E3-1270 de 4 núcleos y 3.8GHz
Arquitectura 64 bits	Arquitectura 64 bits
Memoria RAM de 16GB	Memoria RAM de 32GB
Al Menos 100GB de espacio libre de disco para la instalación de Vitis y Archivos temporales	1.5TB de espacio libre de disco
OS: Al menos Windows 10 o Ubuntu 20.04.2 LTS	Ubuntu 20.04.06 LTS

Por otra parte, los aceleradores del "FAL" han sido diseñados para ser compatibles con dos modelos específicos de FPGA: la Xilinx Virtex UltraScale+ XCU250 y la Zynq® UltraScale+™ MPSoC xck26-sfvc784, también conocida como Kria. Entre estas dos alternativas, se eligió inicialmente trabajar con la FPGA XCU250 debido a que, aunque los aceleradores fueron diseñados para se compatibles con ambas FPGA, al momento de iniciar este proyecto, la implementación en sí de los aceleradores no se había iniciado en la "Kria". En contraste, en la XCU250 ya casi todos los aceleradores habían sido implementados y validados. Como se discutirá en posteriores capítulos del informe, ciertos imprevistos se presentaron que obligaron a finalmente implementar los aceleradores y las optimizaciones de cuantización en la "Kria".

El acceso a la XCU250 se obtuvo mediante el programa de Clústeres de Cómputo Acelerado Heterogéneo (HACC) de AMD. Este programa es una "iniciativa especial para apoyar investigaciones novedosas en la aceleración de cómputo adaptativo para la computación de alto rendimiento (HPC). El alcance del programa es amplio e incluye sistemas, arquitectura, herramientas y aplicaciones." [50]. Esta iniciativa permite acceder, de manera remota, la tarjeta aceleradora U250, la cual contiene la FPGA XCU250 además de módulos de memoria DDR4, una interfaz PCIe, memoria flash entre otros.

3.2 Identificación de necesidades del cliente

La etapa de identificación de necesidades es un proceso clave de la metodología Ulrich Epinger en la cual el diseñador logra comprender a fondo las características que el usuario o cliente busca y valora en el producto. Este proceso busca analizar las expectativas del cliente, sus preferencias y determinar cuál es el problema que el cliente busca resolver. En esta fase, se realizaron entrevistas directas con el cliente [51] para obtener esta información. Basándose en la información obtenida en estas entrevistas, se generó la

Tabla 2, en la cual se enlistan las necesidades interpretadas, y además se les asigna una categoría de importancia según las preferencias del cliente, donde 3 representa el mayor grado de importancia y uno el menor grado de importancia.

Tabla 2. Interpretación de las necesidades del cliente.

Tema	Necesidades Interpretadas	Importancia
Eficiencia energética del modelo	La cantidad de energía disipada por cada inferencia de la y potencia disipada por el modelo debe disminuir.	3
Cantidad de recursos de Hardware	Cantidad de LUTs, bloques BRAM, DSP debe disminuir.	2
Precisión del modelo	La precisión de clasificación del modelo debe no disminuir notablemente..	3
Tiempo de Ejecución	La latencia debe no aumentar notablemente con respecto a la implementación actual	1
Compatibilidad con los aceleradores de HW ya diseñados	El SD debe ser compatible con los aceleradores de HW ya implementados en el ECAS Lab e integrable en el FAL (Flexible Accelerator Library)	3

De esta se puede concluir que el sistema diseñado debe tener como prioridad el aumento de la eficiencia energética y el mantenimiento de la precisión de clasificación. Estas dos características de hecho se relacionan entre sí de manera inversa. Como se ha observado en la

revisión del estado del arte de la sección 2.4, comúnmente una disminución de los recursos de HW conlleva un aumento de la eficiencia energética y un empeoramiento en la precisión del modelo.

Por otro lado, el impacto en el throughput al disminuir el gasto energético puede ser negativo o positivo, dependiendo del diseño y la optimización de los aceleradores de HW. La velocidad de inferencia puede aumentar si el diseño se paraleliza e implementa de forma efectiva pero disminuir si no es así. Todos estos trade-offs y la importancia que el cliente brinda a cada una de las necesidades interpretadas serán tomados en cuenta en las secciones posteriores para establecer las especificaciones del producto, cuales conceptos se analizarán como posibles soluciones, y cuál de estos conceptos se seleccionará como concepto ganador para implementarlo en la FPGA.

3.3 Establecimiento de las especificaciones del producto

Otra etapa fundamental es la de establecer las especificaciones del producto. Esta tiene como objetivo principal traducir las necesidades del cliente a requisitos técnicos y cuantificables. Este proceso, en general, consiste en definir, para cada una de las necesidades del cliente, atributos técnicos medibles que el producto final debe alcanzar. A cada uno de estos atributos técnicos corresponde una métrica, un nivel de importancia, una unidad de medición, y valores marginales e ideales.

El valor ideal representa el estándar deseado y óptimo y el valor marginal corresponde al límite aceptable definido para al menos cumplir con las expectativas mínimas de calidad por parte del cliente. Es importante mencionar que los valores marginales e ideales son determinados por el diseñador no solo utilizando la información obtenida del cliente, sino también un análisis profundo del estado del mercado o, para el caso de investigaciones académicas como esta, el estado del arte. La determinación de estos dos valores permite al diseñador tomar mejores decisiones de diseño, ya que hay muchos casos en los que, se deben dar “trade offs” entre distintas métricas. Además, en el proceso de validación, los valores marginales e ideales permiten evaluar el desempeño de la solución propuesta.

Debido a que la presente investigación corresponde a una optimización de un conjunto de aceleradores de HW que ya han sido diseñados, la determinación de los valores marginales e ideales se basa principalmente en dos tipos de factores. Primero, los factores internos, los cuales corresponden al criterio y las recomendaciones del cliente/asesor industrial el cual es un experto

en el campo con años de experiencia, y segundo, en los factores externos, los cuales son presentados en la investigación de las secciones 2.3 y 2.4.

Por ejemplo, para el valor marginal e ideal de la métrica #7 se analizaron los resultados de [22] [29] [33] [37]. En el caso de [22], [29] y [42], los cuales utilizan cuantización tipo int8, bfloat16 y FP16 respectivamente, la pérdida de precisión de clasificación es prácticamente de 0%, inclusive hay aumento de la precisión en casos específicos [42]. [33] obtiene resultados similares con una caída de solamente 2%. Sin embargo, en otros casos como [37], la caída al utilizar cuantización de punto fijo es del 6%, llegando hasta a un 32% en casos de sets de datos más desafiantes como lo es detección de cáncer de seno. Basándose en estos resultados, en el valor de importancia de "3" que el cliente brinda a la precisión, y en que el cliente explícitamente solicita que la precisión se mantenga, o disminuya muy poco, se elige una disminución del 5% al 2% como valor marginal y menor al 2% como valor ideal.

Otro ejemplo es el caso de la métrica #2, la cual corresponde a la potencia disipada al realizar inferencias. En este caso, se tomaron en consideración varios "papers", pero con mayor prioridad a los que presentaban resultados cuantitativos más claros, tal como [34]. En el caso de la potencia, la literatura también presenta grandes variaciones en los resultados. En este caso, el cliente da el máximo grado de importancia "3" a esta optimización y menciona explícitamente que la potencia que en estos momentos disipa los aceleradores de HW gracias a las optimizaciones arquitectónicas que ya han sido implementadas es aceptable, pero no ideal, se establece un valor marginal de una caída de 6% a 10% y un valor ideal mayor al 10%.

Siguiendo procesos similares a los expuestos en los dos párrafos anteriores, se determinan los valores marginales e ideales de cada una de las métricas. Estos resultados se muestran en la

Tabla 3, la cual además presenta el número de métrica, el número de necesidad del cliente que corresponde a cada una de estas métricas, la importancia asignada por el cliente, y la unidad de medida de cada una de estas. En esta tabla el diminutivo dis. corresponde a una disminución y aum. a un aumento.

Tabla 3. Especificaciones del producto por diseñar

# de Métrica	# de Necesidad	Métrica	Importancia	Unidad de medida	Valor Marginal	Valor Ideal
1	1	Potencia disipada al realizar inferencias	3	Miliwatts (mW)	Dis. [6-10]%	Dis. >10%
2	2	Cantidad de LUTs	2	Medida discreta	Dis. [8-20]%	Dis. >20%
3	2	Cantidad de DSP	3	Medida discreta	Dis. [8-20]%	Dis. >20%
4	2	Cantidad de Bloques BRAM	3	Medida discreta	Dis. [8-20]%	Dis. >20%
5	2	Cantidad de Flip-Flops	2	Medida discreta	Dis. [8-20]%	Dis. >20%
6	3	Precisión top 1 de inferencia del ViT.	3	% de precisión	Dis. [5-2]%	Dis. <2%
7	4	Latencia	2	Milisegundos (ms)	Aum. [50-20]%	Aum. <20%
8	5	Implementado usando HLS	3	Binario	True	True
9	5	Uso de los aceleradores de HW previamente diseñados en el ViT	3	Binario	True	True

3.4 Selección de conceptos

La siguiente etapa corresponde a la selección de conceptos. En la metodología Ulrich Epinger esta etapa comúnmente se denomina Generación de Conceptos, sin embargo, por la naturaleza académica e investigativa de este proyecto, se tomó la decisión de enfocar esta sección a una selección de estrategias de cuantización que han sido implementados en el pasado de manera exitosa, y que, considerando las características de este proyecto, podrían también ser implementadas en este.

Se realizó una profunda revisión de la literatura de métodos de cuantización que, basándose en criterio del diseñador, pueden ser los que se adapten de mejor manera para cuantizar el ViT, tal como se muestra en las secciones 2.3 y 2.4. El proceso de selección de conceptos analizó muchas técnicas con diversas ventajas y desventajas entre ellas, las cuales se presentan a continuación:

- Cuantización binaria
- Cuantización Ternaria
- Cuantización tipo INT4
- Cuantización tipo INT8
- Cuantización tipo INT16
- Cuantización tipo BFloat16
- Cuantización FP16
- Cuantización tipo punto fijo
- Cuantización tipo Posit
- Cuantización logarítmica
- Cuantización en potencias de dos.

Se tomó la decisión de analizar de manera más profunda seis de estas técnicas, y finalmente, comparar estas seis técnicas entre sí de manera más metódica para elegir una de ellas como ganadora e implementarla en la FPGA. Se tomaron en cuenta cuatro tres criterios distintos para realizar la elección de las 5 técnicas más convenientes, los cuales se presentan a continuación.

3.4.1 Recomendaciones del asesor técnico/experto en el campo el doctor Luis León Vega

El Dr. León cuenta con una vasta experiencia en esta área, respaldada por su tesis de maestría y de doctorado, ambas enfocadas en el desarrollo de los aceleradores de HW que se cuantizan en este proyecto. Por estas razones, su opinión y criterio experto son de suma importancia para elegir los conceptos más adecuados. Además de su conocimiento técnico, el Dr. León posee un entendimiento profundo de estos aceleradores ya que él y su equipo fueron los que los diseñaron, lo cual aporta una perspectiva valiosa y única en el proceso de decisión.

3.4.2 Revisión del estado del arte

Se realizó una primera etapa de la revisión bibliográfica en la cual se llevó a cabo un análisis general y preliminar de las técnicas de cuantización para elegir los 6 conceptos ganadores. Principalmente, se tomaron en cuenta dos factores para tomar esta decisión: la similitud de las condiciones de cuantización de los "papers" a las de este estudio, como por ejemplo que se aplicara en redes neuronales profundas similares al ViT o que la cuantización fuese implementada en FPGA, y los resultados de optimización, dando especial énfasis en la reducción de recursos energéticos y el mantenimiento de la precisión de inferencia

3.4.3 Distribución de valores de los tensores de los pesos y activaciones del ViT.

Como se mencionó de manera breve en el marco teórico, el estudio de la distribución de los pesos y las activaciones del modelo ViT son de suma importancia. Al comprender las

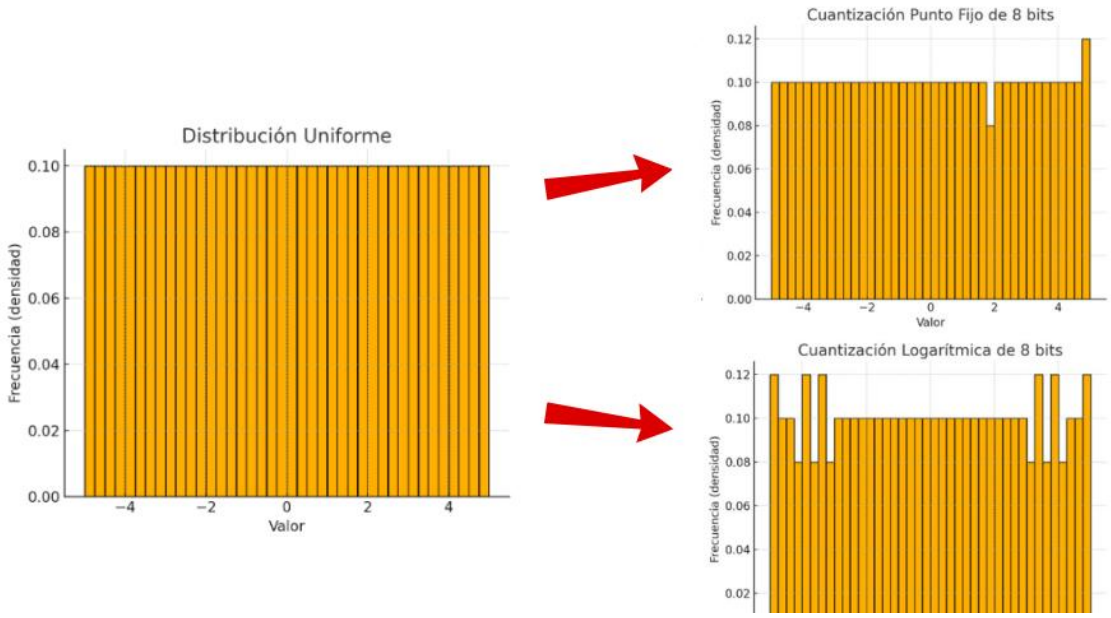


Figura 16. Cuantización de punto fijo y logarítmica para una distribución uniforme.

características estadísticas de esta distribución, como por ejemplo el rango, la desviación estándar, la resolución, a que tipo de distribución se asemeja, se puede elegir una técnica de cuantización en la cual, utilizando la menor cantidad de bits, los datos se representen sin perder su valor informativo.

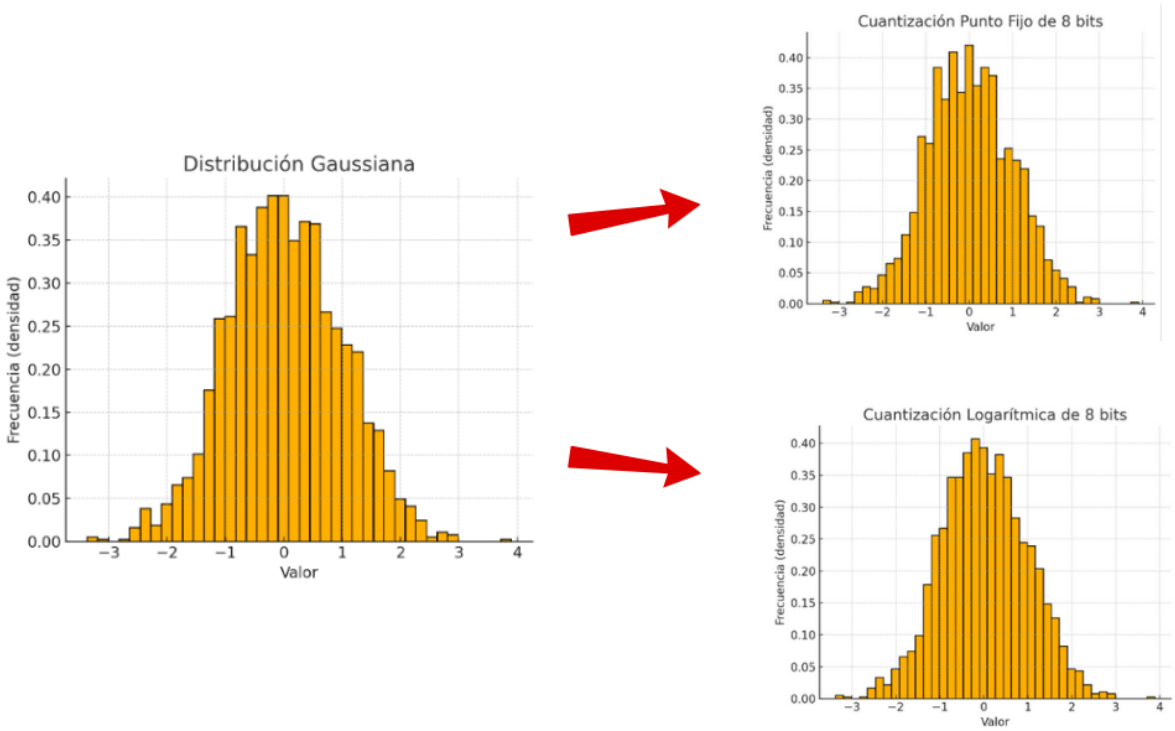


Figura 17. Cuantización de punto Fijo y Logarítmica para distribución Gaussiana

A manera de ejemplo para comprender esta importancia, se presentan la Figura 16 y la Figura 17. Una distribución uniforme, en la cual los datos se encuentran distribuidos de manera homogénea, será mejor cuantizada haciendo uso de una cuantización de punto fijo al compararla con una logarítmica, tal como se observa en la Figura 16. Esto se debe a que la cuantización logarítmica proporciona mayor precisión a los valores cercanos a cero, mientras la cuantización de punto fijo brinda precisión constante para todos los valores del rango. Por otra parte, una distribución uniforme centrada en cero, en la cual la mayor densidad de datos se encuentra en los valores más cercanos a cero, se cuantizará de manera más fiel utilizando cuantización logarítmica, tal como se aprecia en la Figura 17.

Por estas razones, se obtuvieron los histogramas de los pesos y las activaciones del ViT. Estos histogramas se presentan en los apéndices al final de este documento. Además de los histogramas, se obtuvieron los datos de precisión (número más pequeño distinto de cero) y rango

dinámico (diferencia entre el número mayor y número menor de los datos), los cuales también serán importantes en la etapa del desarrollo de la solución. Los histogramas se dividen en tres:

1) **Activaciones (Apéndice A)**:

Se eligió al azar una muestra de 66 imágenes divididas en 13 clases relacionadas a ADAS. Los histogramas muestran todas las activaciones del modelo, desde la entrada al modelo como imagen RGB hasta su salida como el vector de clasificación de 1000 dimensiones. Las clases corresponden a: Street sign, traffic light, trailer truck, stone wall, sports car, school bus, pole, pickup, minivan, manhole cover, garbage truck y mountain bike.

- Histogramas de 4 imágenes de muestra en distintos rangos: Este histograma tiene la finalidad de demostrar que las distribuciones de las activaciones son muy similares para las distintas imágenes del set de datos. **(Apéndice A.1).**
- Histogramas de la distribución de datos de todas las 66 imágenes de la muestra combinadas: Estos histogramas muestran, para distintas escalas, la distribución de las activaciones de todas las 66 imágenes. **(Apéndice A.2).**
    - Precisión: 7.839488502803249e-10
    - Rango Dinámico: 803.9630126953125
- Histograma de la distribución de precisiones y rangos dinámicos de las 66 imágenes: Se obtuvo también un histograma que representa la distribución de las precisiones y rango dinámico para las 66 imágenes de la muestra. **(Apéndice A.3)**

2) **Pesos (Apéndice B)**

Corresponde a histogramas de pesos, “biases” y cualquier otro tipo de parámetro entrenable que tenga el modelo.

- Histogramas de cada peso del modelo. Se agrupan según la operación matemática específica del ViT en la que operan. En el caso de los histogramas de varios colores, cada color representa uno de los 12 ciclos (12 capas de atencion) en los cuales se repiten los módulos "Attention" y MLP. **(Apéndice B.1)**.
- Histogramas que muestran rango dinámico y precisión de los histogramas del Apéndice B.1. **(Apéndice B.2).**
- Histograma que combina todos los pesos. Este histograma condensa todos los histogramas del Apéndice B.1 en uno solo **(Apéndice B.3).**
    - Precisión: 5.960464477539063e-08

- Rango Dinámico: 17.470661640167236

3) Histograma Combinado de activaciones y pesos

Este Histograma presenta absolutamente toda la información estadística del ViT, ya que combina la distribución de las activaciones y los pesos. Como los datos de activaciones corresponden a 66 imágenes de muestra, se sumaron todos estos datos y se dividieron entre 66 para "normalizar" y obtener un histograma que represente la distribución al realizar la inferencia del ViT para una imagen. Este histograma se presenta en la Figura 18, en la cual el color azul representa los pesos y el verde a las activaciones.

- Precisión: 7.839488502803249e-10
- Rango Dinámico: 803.9630126953125

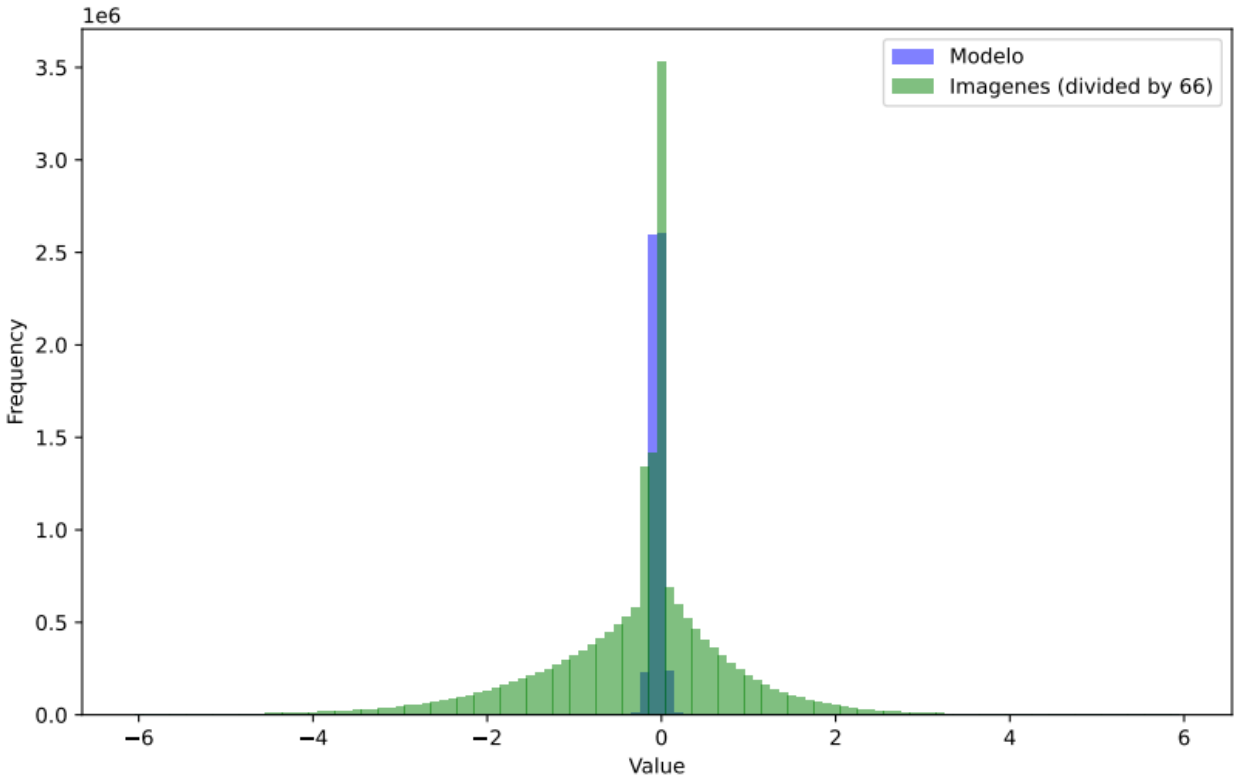


Figura 18. Distribución estadística de todos los datos del ViT: Activaciones y pesos

3.4.4 Conceptos elegidos

Entonces, tomando en consideración estos tres factores, se eligieron los tipos de cuantización: Cuantización logarítmica, Cuantización INT8, Cuantización BFloat16, Cuantización de punto fijo, Cuantización Posit y Cuantización FP16. Estas técnicas se comparan entre sí de

manera exhaustiva y se elige entre ellas como concepto ganador la que se adapte de mejor manera a las necesidades del cliente y el contexto del proyecto. En la siguiente sección se muestra esta comparación y selección del concepto ganador.

3.5 Selección del concepto ganador

Finalmente, los 5 conceptos seleccionados se comparan entre sí para elegir cual será el concepto ganador y el tipo de cuantización que será implementada. En esta etapa, se decide utilizar la matriz de Selección de conceptos planteada en la metodología Ulrich-Epinger. En esta matriz, las 6 técnicas de cuantización se comparan por medio de criterios de selección. Los criterios de selección se plantean de manera que todas las necesidades del cliente se vean reflejadas en estos. Esta matriz de selección se presenta en la

tabla 4, en la cual "C" significa clasificación y "EP" evaluación ponderada.

Tipos de cuantización													
Criterios de Selección	Peso	INT8		Punto Fijo		BFloat16		Posit		Log		FP16	
		C	EP	C	EP	C	EP	C	EP	C	EP	C	EP
Eficiencia energética	18%	5	0.9	3	0.54	4	0.72	3	0.54	3	0.54	3	0.54
Costo de HW	12%	5	0.6	3	0.36	4	0.48	2	0.24	3	0.36	3	0.36
Precisión de la inferencia	18%	3	0.54	4	0.72	3	0.54	5	0.9	4	0.72	5	0.9
Velocidad de inferencia	8%	5	0.4	4	0.32	3	0.24	3	0.24	3	0.24	3	0.24
Compatibilidad con el FAL	24%	5	1.2	5	1.2	5	1.2	4	0.96	4	0.96	5	1.2
Compatibilidad con condiciones de diseño	20%	2	0.4	4	0.8	2	0.4	1	0.2	1	0.2	5	1
Total de puntos	100%	4.04		3.94		3.58		3.08		3.02		4.24	
Lugar	-	2		3		4		6		5		1	
Implementar	-	No		No		No		No		No		Sí	

El funcionamiento de la matriz de selección es el siguiente: Primeramente, se asocia a cada criterio de selección un peso específico. El peso de cada criterio se basa en la importancia que brinda el cliente a cada una de sus necesidades y en el criterio experto del diseñador que toma en consideración el contexto en el cual se implementa la cuantización. Para cada criterio de selección asociado a cada técnica de cuantización se asigna una calificación que va del 1 al 5, donde 5

corresponde a que cumple con el criterio de selección de manera sobresaliente, y 1 que la técnica es deficiente en el cumplimiento de este criterio de selección. Posteriormente, esta calificación se multiplica por el peso, y se obtiene una evaluación ponderada. Finalmente, se suman todas las evaluaciones ponderadas, y la evaluación ponderada más alta indica la técnica de cuantización ganadora y la que debe ser implementada.

Como se puede observar en la Tabla 4, los criterios de selección consideran dos grupos de

Tipos de cuantización													
Criterios de Selección	Peso	INT8		Punto Fijo		BFloat16		Posit		Log		FP16	
		C	EP	C	EP	C	EP	C	EP	C	EP	C	EP
Eficiencia energética	18%	5	0.9	3	0.54	4	0.72	3	0.54	3	0.54	3	0.54
Costo de HW	12%	5	0.6	3	0.36	4	0.48	2	0.24	3	0.36	3	0.36
Precisión de la inferencia	18%	3	0.54	4	0.72	3	0.54	5	0.9	4	0.72	5	0.9
Velocidad de inferencia	8%	5	0.4	4	0.32	3	0.24	3	0.24	3	0.24	3	0.24
Compatibilidad con el FAL	24%	5	1.2	5	1.2	5	1.2	4	0.96	4	0.96	5	1.2
Compatibilidad con condiciones de diseño	20%	2	0.4	4	0.8	2	0.4	1	0.2	1	0.2	5	1
Total de puntos	100%	4.04		3.94		3.58		3.08		3.02		4.24	
Lugar	-	2		3		4		6		5		1	
Implementar	-	No		No		No		No		No		Sí	

características: las características cuantitativas que describen el rendimiento de la red cuantizada, las características cualitativas que toman en cuenta el entorno del diseño. En el primer grupo se encuentran la eficiencia energética, el costo de HW, la precisión de la inferencia y la velocidad de inferencia. En el segundo, la compatibilidad de la solución con los aceleradores del FA y la compatibilidad con las condiciones de diseño.

El criterio de diseño: “Compatibilidad con las condiciones de diseño” considera que la cuantización sea implementable tomando en consideración posibles limitaciones de recursos del presente proyecto. Por ejemplo, la cuantización Bfloat16 ha demostrado buenos resultados en el estado del arte en términos de eficiencia energética y baja pérdida de precisión, pero en todos estos “papers” analizados, se utiliza la técnica QAT, la cual no es posible de implementar porque para volver a entrenar al ViT aplicando la cuantización, y con los recursos computacionales disponibles,

el modelo tardaría semanas o hasta meses entrenándose, además de que no se tiene acceso al set de datos de entrenamiento.

Tabla 4. Matriz de Selección del Concepto Ganador

De esta manera, se concluye que la cuantización a implementar será la floating point de 16 bits o FP16. Este tipo de cuantización destaca al considerar las necesidades y especificaciones, ya que en la literatura muestra mejoras energéticas considerables y mantiene la precisión intacta, o incluso la mejora.

4. Diagnóstico del estado actual del modelo ViT con los aceleradores del ECAS Lab.

4.1 Desafíos enfrentados

Originalmente, el laboratorio ECAS había implementado los aceleradores de tipo EW, Multiplicación matricial, "Unary" (Operaciones unarias tal como la función de activación GelU), Softmax y Normalización RMS en la FPGA Alveo U250 por medio del acceso remoto al ETH de Zurich, tal como se describe en el capítulo de metodología de este informe. Sin embargo, durante el proceso de realización de este proyecto, la FPGA llegó a su EOL o "end of life". Esto significa que la empresa que fabrica esta FPGA, AMD, no brinda más soporte a este tipo de FPGA y, en consecuencia, fue retirada del programa HACC de AMD. Por lo tanto el ECAS lab perdió acceso a este dispositivo.

Una de las alternativas ante este problema corresponde a hacer uso del dispositivo MPSoC Zynq UltraScale+, ya que el asesor industrial Luis León es propietario de uno de estos. Este dispositivo combina un procesador ARM de cuatro núcleos Cortex-A53, un procesador de tiempo real de doble núcleo Cortex-R5F y una matriz de lógica programable basada en FPGA. Debido a que el proceso de migración de los aceleradores de una FPGA a otra requiere de un refinamiento

en los parámetros de diseño, además de una nueva síntesis de los aceleradores, y por cuestiones de tiempo de entrega de este proyecto, solamente se implementó el acelerador tipo EW para sumas y multiplicaciones en el ViT. Por lo tanto, la cuantización de tipo FP16 fue diseñada para este acelerador.

Otro desafío en el desarrollo del proyecto consistió en que los aceleradores tipo EW no habían sido diseñados por parte del equipo del ECAS lab para trabajar con operaciones en las cuales una de sus entradas correspondiese a una matriz y la otra a un vector. El ViT, tal como se puede observar en las Figuras 2, 3, 5 y 7, realiza operaciones EW en las cuales una de sus entradas es un vector. Debido a esto, y a qué la modificación de la arquitectura fundamental de los aceleradores escapa del alcance de este proyecto, se implementó en SW (CPU) la técnica conocida como “Broadcasting” para que el acelerador fuese incorporable a el ViT.

El “broadcasting” corresponde a una operación pesada computacionalmente, en la cual un vector se “expande” para tomar la forma y las dimensiones de una matriz específica. Por ejemplo, si se tiene una matriz de 192x197 elementos, y un vector de 192 elementos, el “broadcasting” consistiría en expandir el vector de 192 elementos 197 veces, para generar una matriz de 197 columnas idénticas con los 192 elementos del tensor original.

4.2 Precisión y Tiempo de inferencia

A continuación se presentan las mediciones de precisión y tiempo de inferencia obtenidos al implementar el acelerador de HW diseñado por el ECAS Lab en el ViT. Estos aceleradores no han tenido ningún tipo de cuantización aplicada, por lo cual todos sus datos son procesados en FP32. Las pruebas de validación utilizadas en este caso son iguales a las que se utilizarán para medir el rendimiento de la solución que será diseñada. La metodología de estas pruebas es explicada más a fondo en la primera subsección del capítulo 5, pero, en resumen, se realizaron inferencias sobre 650 imágenes elegidas al azar de 13 categorías relacionadas a ADAS del set de datos de validación de imagenet1k. En la Figura 19 se muestran los resultados.

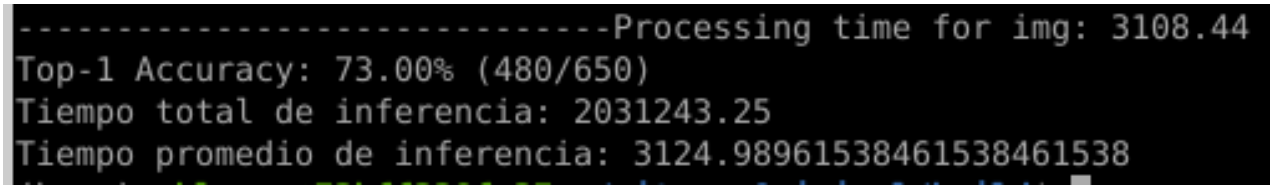


Figura 19.Resultados de precisión top1 y tiempo de inferencia promedio para el acelerador EW FP32

En este caso, el tiempo de inferencia promedio de las 650 imágenes es mucho más elevado que el que se obtiene al ejecutar el algortimo del ViT completamente en la CPU. Un tiempo de inferencia de 3s es hasta 30 veces mayor que el resultado que se muestra en la tabla 6 para el modelo "small". Tiempos de inferencia tan elevados no son aceptables para aplicaciones críticas tales como ADAS. Este resultado se atribuye principalmente a dos factores:

- **Implementación parcial del modelo en la FPGA:** Como se ha mencionado varias veces, debido al "EOL" de la FPGA Alveo y los desafíos que esto desencadenó, solo el acelerador tipo EW fue implementado en la "Kria". La constante transferencia de datos entre el CPU y la FPGA genera overhead, ya que cada trasferencia de CPU a FPGA o de FPGA a CPU se ejecuta por medio del bus AXI, lo cual introduce una latencia mucho mayor a una transferencia de datos dentro de la CPU o dentro de la FPGA. Además puede ocurrir frecuentemente que la FPGA esté esperando datos de la CPU y viceversa, por lo cual parte del HW queda inactivo y se añade tiempo de espera
- **"Broadcasting" ejecutado en CPU:** El "broadcasting" programado en SW introduce también latencia al proceso. Si el acelerador de HW hubiera sido diseñado para recibir operandos tanto matriz-matriz como matriz-vector, la ejecución sería mucho más veloz y eficiente.

4.3 Potencia Disipada

En el caso de la potencia disipada, se obtuvo el dato de la potencia específicamente para el acelerador EW FP32, no la potencia disipada por todo el modelo, ya que la optimziación propuesta se aplicará específicamente al acelerador EW, no al resto del modelo que se ejecuta en la CPU. En la Figura 20 se muestran los detalles de la potencia disipada por el acelerador elementwise FP32. Esta simulación de potencia disipada se realizó con exactamente los mismos parámetros de ambiente que se realizó la simulación para obtener la potencia del acelerador con cuantización

FP16 incorporada. En la sección 6.2 se muestran más detalles relacionados a esta simulación y sus características.

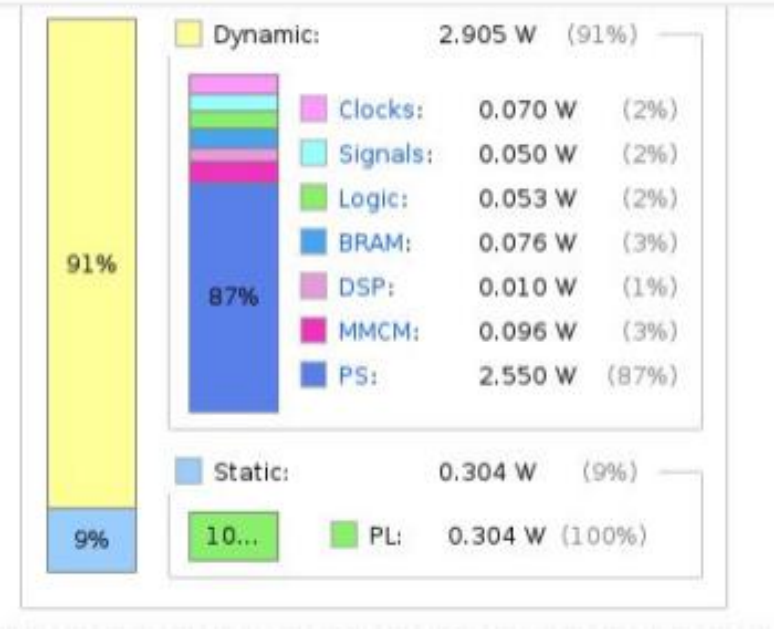
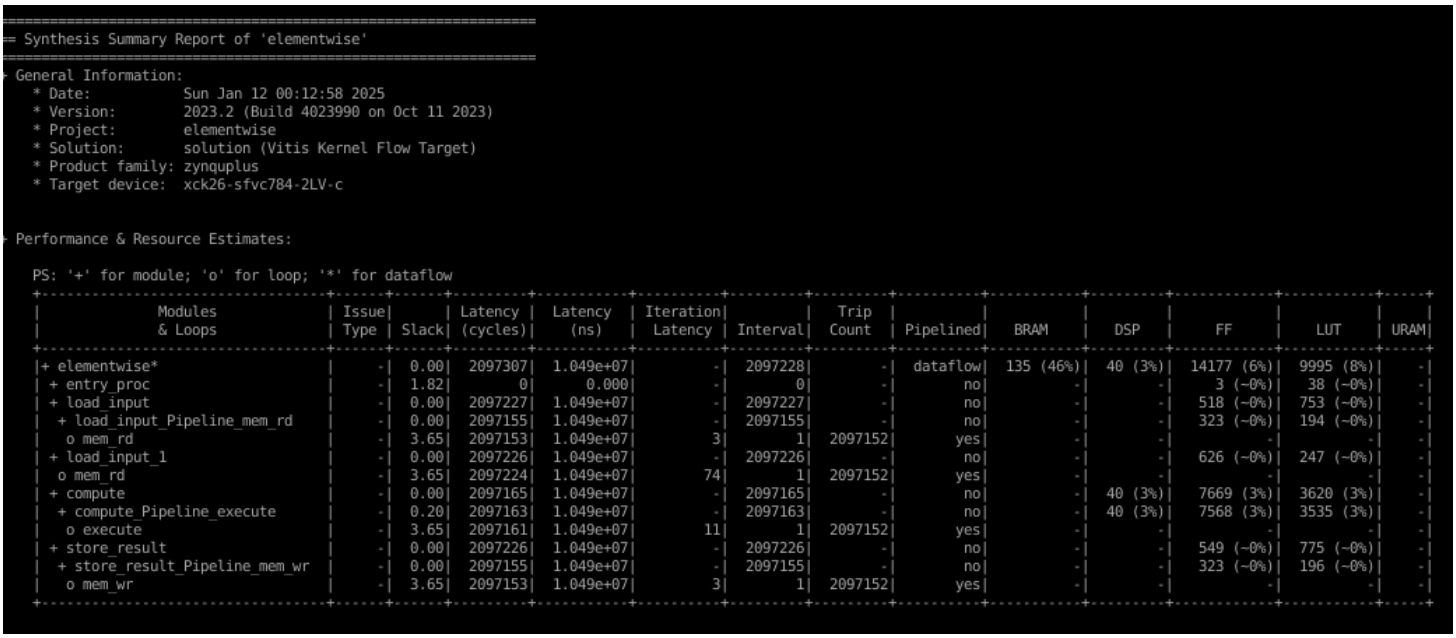


Figura 20. Potencia disipada por el acelerador EW FP32.

Para obtener el consumo real de potencia del acelerador elementwise, se deben sumar la potencia dinámica y la potencia estática y restar la potencia PS. Esto se debe a que la ptencia PS corresponde a la potencia que disipa el CPU ARM del SoM, CPU que no se utiliza para los aceleradores, ya que estos se implementan en su totalidad en la FPGA. En este caso, el consumo de potencia simulado para el acelerador EW FP32 es de 0.659W o 659mW.



```
================================================================
== Synthesis Summary Report of 'elementwise'
================================================================
+ General Information:
    * Date:           Sun Jan 12 00:12:58 2025
    * Version:        2023.2 (Build 4023990 on Oct 11 2023)
    * Project:        elementwise
    * Solution:       solution (Vitis Kernel Flow Target)
    * Product family: zynquplus
    * Target device:  xck26-sfvc784-2LV-c


+ Performance & Resource Estimates:

    PS: '+' for module; 'o' for loop; '*' for dataflow
    +---------------------------------+------+------+---------+----------+----------+---------+--------+----------+-----------+---------+------------+-----------+-----+
    |             Modules             | Issue|      | Latency | Latency  | Iteration|         |  Trip  |          |           |         |            |           |     |
    |             & Loops             | Type | Slack| (cycles)|   (ns)   |  Latency | Interval| Count  | Pipelined|   BRAM    |   DSP   |     FF     |    LUT    | URAM|
    +---------------------------------+------+------+---------+----------+----------+---------+--------+----------+-----------+---------+------------+-----------+-----+
    |+ elementwise*                   |     -|  0.00|  2097307|  1.049e+07|         -|  2097228|       -|  dataflow| 135 (46%)|  40 (3%)|  14177 (6%)|  9995 (8%)|    -|
    | + entry_proc                    |     -|  1.82|        0|     0.000|         -|        0|       -|        no|          -|        -|       3 (~0%)|    38 (~0%)|    -|
    | + load_input                    |     -|  0.00|  2097227|  1.049e+07|         -|  2097227|       -|        no|          -|        -|     518 (~0%)|   753 (~0%)|    -|
    |  + load_input_Pipeline_mem_rd   |     -|  0.00|  2097155|  1.049e+07|         -|  2097155|       -|        no|          -|        -|     323 (~0%)|   194 (~0%)|    -|
    |   o mem_rd                      |     -|  3.65|  2097153|  1.049e+07|         3|        1|  2097152|       yes|          -|        -|           -|           -|    -|
    | + load_input_1                  |     -|  0.00|  2097226|  1.049e+07|         -|  2097226|       -|        no|          -|        -|     626 (~0%)|   247 (~0%)|    -|
    |  o mem_rd                       |     -|  3.65|  2097224|  1.049e+07|        74|        1|  2097152|       yes|          -|        -|           -|           -|    -|
    | + compute                       |     -|  0.00|  2097165|  1.049e+07|         -|  2097165|       -|        no|          -|  40 (3%)|    7669 (3%)|  3620 (3%)|    -|
    |  + compute_Pipeline_execute     |     -|  0.20|  2097163|  1.049e+07|         -|  2097163|       -|        no|          -|  40 (3%)|    7568 (3%)|  3535 (3%)|    -|
    |   o execute                     |     -|  3.65|  2097161|  1.049e+07|        11|        1|  2097152|       yes|          -|        -|           -|           -|    -|
    | + store_result                  |     -|  0.00|  2097226|  1.049e+07|         -|  2097226|       -|        no|          -|        -|     549 (~0%)|   775 (~0%)|    -|
    |  + store_result_Pipeline_mem_wr |     -|  0.00|  2097155|  1.049e+07|         -|  2097155|       -|        no|          -|        -|     323 (~0%)|   196 (~0%)|    -|
    |   o mem_wr                      |     -|  3.65|  2097153|  1.049e+07|         3|        1|  2097152|       yes|          -|        -|           -|           -|    -|
    +---------------------------------+------+------+---------+----------+----------+---------+--------+----------+-----------+---------+------------+-----------+-----+
```

Figura 21. Resumen de reporte de sintesis "Csynth.rpt" generado por Vitis para el acelerador elementwise FP32

4.4 Recursos Computacionales

Finalmente, se muestra en la Figura 21 el uso de recursos computacionales reportados por Vitis para la síntesis del acelerador Elementwise. La primera columna denominada “Modules & Loops enumera las funciones del acelerador programadas en C++. La primer fila denominada “elementwise” presenta la totalidad de recursos lógicos utilizados. Las siguientes filas reportan de manera más granular cuantos recursos utiliza cada función del acelerador, las cuales corresponden a: Load_Input1, Load_Input2, compute y store_result. En la Tabla 5 se presenta un resumen de los datos de mayor interés, los cuales son la cantidad de BRAMs totales, cantidad de DSP totales, cantidad de FFs totales, y cantidad de LUTs totales.

Tabla 5. Recursos Computacionales utilizados por el acelerador EW FP32

	BRAM	DSP	Flip-Flops	Look-Up Tables
Cantidad	135	40	14177	9995
Porcentaje de utilización	46%	3%	6%	8%

5. Diseño e implementación de la solución

En este capítulo se pretende abordar de manera detallada todo el proceso de diseño, implementación y validación de la cuantización tipo FP16.

5.1 Modelo del ViT

El algoritmo de visión por computadora a optimizar por medio de cuantización tipo FP16 corresponde a un ViT. El funcionamiento de este modelo se explica a fondo en la primera subsección del marco teórico de este informe. El ECAS lab eligió trabajar con la implementación del Vision Transformer elaborada por "staghado", la cual es accesible por medio del repositorio de github que se presenta en el Anexo A. Esta implementación tiene como objetivo principal "desarrollar una interfaz de inferencia en C/C++ utilizando la biblioteca de tensores GGML para mejorar el rendimiento, particularmente en dispositivos en el borde" [52].

Esta implementación del ViT presenta varios modelos que varían en tamaño, es decir, en cantidad de parámetros entrenables (pesos y biases). Los 4 modelos disponibles corresponden a: "Tiny", "Small", "base" y "large". Entre más grande sea el modelo, el proceso de inferencia será más preciso, pero también se hará uso de más de recursos computacionales, y el tiempo de inferencia será mayor. Todos estos modelos fueron entrenados por los investigadores haciendo uso del set de datos imagenet21k y ajustados ("fine-tuned"), con el set de datos imagenet1k. Cabe mencionar que todos estos modelos utilizan para sus datos la representación estándar FP32.

Para comparar el rendimiento y las características de estas 4 opciones y elegir el modelo idóneo a cuantizar, se realizaron pruebas de precisión y tiempo de inferencia del ViT en su implementación en CPU (Para especificaciones de la CPU ver "xrp france" en la sección 3.1.2). Por medio de los resultados de estas pruebas, se puede elegir el modelo que obtenga un mejor balance entre tiempo de inferencia y precisión de clasificación top-1 para los requerimientos del SD. La precisión top-1 se refiere a la precisión obtenida cuando la predicción del modelo se considera correcta si la clase con la probabilidad más alta coincide con la clase verdadera.

Las pruebas se realizaron a un set de 650 imágenes tomadas al azar del set de datos image1knet de 13 categorías distintas relacionadas a ADAS [53]. Estas categorías corresponden a: Street sign, traffic light, trailer truck, stone wall, sports car, school bus, pole, pickup, minivan, manhole cover, garbage truck y mountain bike. En la Tabla 6 se presentan los resultados obtenidos.

Tabla 6. Precisión y tiempo de inferencia de los 4 modelos

Modelo	Tiempo Promedio de inferencia (ms)	Precisión top-1 obtenida (%)

Tiny	53.40	44.15
Small	119.05	73.85
Base	389.13	75.38
Large	1343.05	81.69

Basándose en una posible aplicación de este modelo en ADAS, y en la normativa internacional y el consenso ingenieril presentado en la sección 2.5, se considera que el modelo que obtiene los mejores resultados corresponde al modelo “small”. Los modelos “Base” y “Large”, aunque presentan las precisiones más elevadas, tienen tiempos de inferencia demasiado elevados para aplicaciones en carretera. Por otro lado el modelo “tiny”, aunque presenta un tiempo de inferencia adecuado para aplicaciones ADAS, tiene una precisión top-1 demasiado baja del 44.15%.

5.2 Decisiones preliminares del diseño de la cuantización

Primeramente se tomaron ciertas decisiones con respecto a cuáles van a ser los atributos de la cuantización FP16. Esras decisiones se tomaron basándose en las condiciones del proyecto y necesidades del cliente. En esta subsección se discuten la selección de las siguientes dos propiedades de cuantización: cuantización PTQ y cuantización estática.

A) Cuantización PTQ

Como se describió en la sección 2.3.4, la cuantización PTQ cuantiza un modelo de IA que ya ha sido entrenado anteriormente, y la cuantización QAT incorpora la cuantización durante el proceso del entrenamiento. La cuantización PTQ es mucho más conveniente para este proyecto principalmente por dos razones. Primero, la cuantización tipo QAT requiere que se tenga acceso al conjunto de datos de entrenamiento. Para obtener acceso al conjunto de datos imagenet21k se debe contactar con las universidades de Princeton y Stanford [53]. Además, el set de datos de imagenet21k tiene un peso muy elevado, en el orden de los TB, lo cual lo hace poco conveniente en términos de almacenamiento.

Por otra parte, si se buscara utilizar cuantización QAT, el modelo debería ser re-entrenado. El modelo “Small” del ViT posee 2.017.616 parámetros de entrenamiento. El re-entrenar el modelo con un set de datos tan grande como imagenet21k podría tomar una gran cantidad de tiempo con la cual no se dispone para la realización de este proyecto. Suponiendo que se utiliza

una CPU de propósito general de alta gama, como a la que se tiene acceso por medio de “xrp france”, que esta alcanza un estimado conservador de procesamiento de 10 imágenes por segundo, y además tomando en consideración que el set de datos image1knet tiene más de un millón de imágenes con definición de 224x224 pixeles, el tiempo de entrenamiento sería de semanas, tal como se muestra en la ecuación 21.

$$Tiempo\ entenamiento = \frac{1{,}281{,}167 imágenes}{10 imágenes/s} * 50 epocas \approx 74\ días \qquad (21)$$

Entonces, bajo las condiciones de este proyecto, se opta por realizar la cuantización del modelo tras el entrenamiento mediante el enfoque de Post-Training Quantization (PTQ). La pérdida en la precisión de inferencia al utilizar PTQ en lugar de Quantization-Aware Training (QAT) se considera aceptable al evaluar las necesidades del cliente.

B) Cuantización Estática

La cuantización estática, esta presenta dos ventajas fundamentales para este proyecto. La primera de ellas y la que posee más peso, es que tal como se mencionó en la sección 2.3.3, la cuantización dinámica incurre en un “overhead” adicional ya que se requiere computo en tiempo real de las características estadísticas del set de datos. Además, el cómputo de estos valores en FPGA introduce lógica adicional que aumenta la cantidad de recursos computacionales y energéticos necesarios, lo cual es completamente contrario al objetivo principal de esta investigación. También la dificultad de implementación aumenta porque hay que tomar en cuenta que se deben modificar los factores de escalado a medida que los datos atraviesan el pipeline de la FPGA.

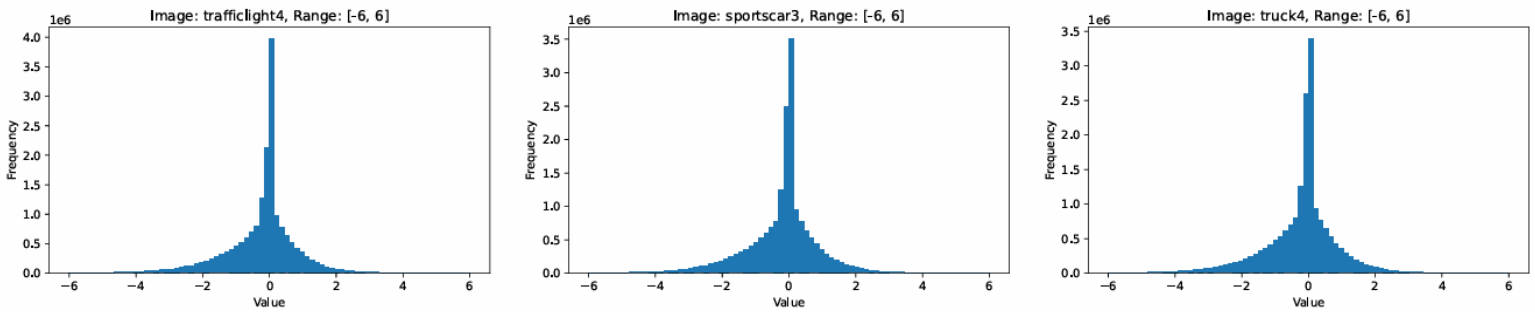


Figura 22. Comparación de histograma de pesos y activaciones para 3 imágenes distintas relacionadas a ADAS del set de datos imagenet1k

La otra razón por la cual se elige cuantización estática es que, según el estudio de histogramas presentado en la sección 2.4.3 y en el Apéndice A.1 se puede concluir que distintas imágenes de distintas categorías poseen activaciones muy similares, tal como se aprecia en la Figura 22. Esta pequeña variabilidad estadística hace que la cuantización dinámica no sea necesaria, ya que es probable que la representación de los valores no pierda mucha precisión aunque las imágenes de entrada sean distintas.

5.3 Cantidad de bits de Mantisa y Exponente para FP16.

Dos parámetros de vital importancia al diseñar cuantización de tipo FP16 es el número de bits de la mantisa y el número de bits del exponente. Es necesario, o al menos altamente deseable, que la cantidad de bits de la mantisa y la cantidad de bits del exponente permitan representar el rango dinámico y la precisión del conjunto de datos. Por esta razón, se decidió realizar nuevamente un estudio estadístico por medio de histogramas, pero en este caso, solamente para los operandos de las operaciones tipo EW, ya que solamente estas se ejecutan en el acelerador de HW.

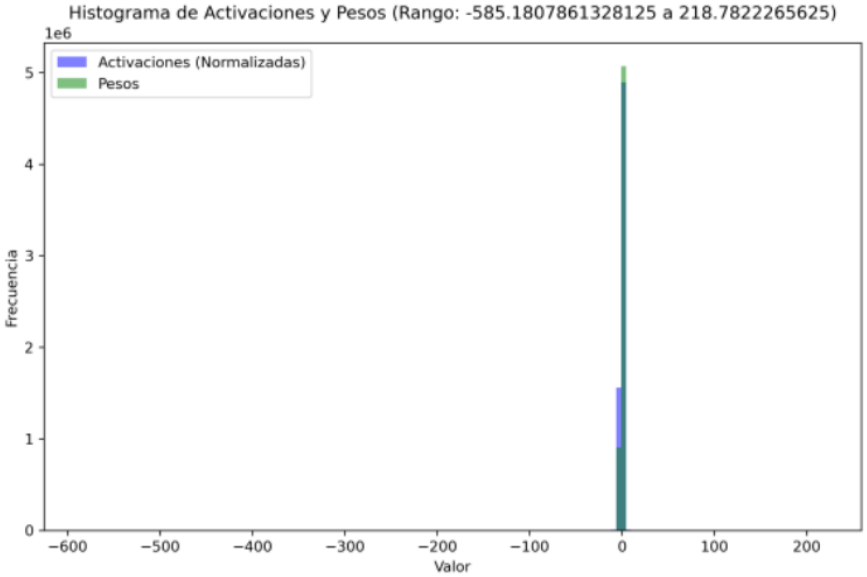


Figura 23.Histograma de activaciones y pesos operandos de EW con rango igual al rango dinámico de los datos.

El objetivo de este análisis gráfico es determinar si la distribución estandar de la normativa IEEE754, la cual se compone de 1 bit de signo, 5 bits de exponente y 10 bits de mantisa [38], se adapta de manera adecuada a la distribución de datos, principalmente al rango dinámico y la precisión de los datos. Esta distribución de 5 bits de exponente y 10 bits de mantisa se elije de

manera preliminar por motivos de conveniencia, ya que los aceleradores se codifican en C++ por medio de HLS, y el tipo estándar de dato “half” se encuentra ya disponible como en cualquier lenguaje de programación de alto nivel, no es necesario definirlo.

El rango dinámico de FP16 IEEE754 se puede obtener de la ecuación 20 y corresponde a [-65504,65504]. Por otro lado, la precisión de FP16 varía conforme los valores se acerquen a cero, entre más cercano de cero esté el valor, mayor será su precisión. La precisión de mayor interés es la de valores más cercanos a cero, ya que en las distribuciones de datos de modelos de IA la densidad de la distribución aumenta en estos valores. La precisión en datos cercanos a cero para FP16 IEEE754 corresponde a 5.97x10-8. Entonces, se debe confirmar con los histogramas si el rango dinámico es menor a [-65504,65504] y la precisión mayor a 5.97x10-8

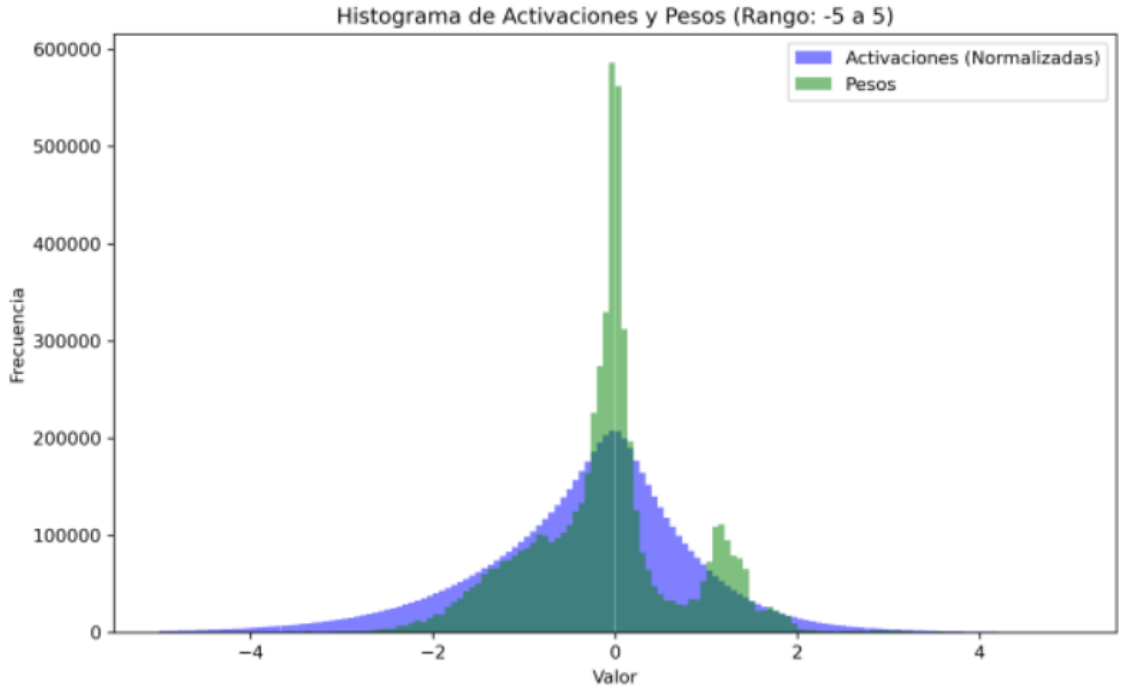


Figura 24. Histograma de activaciones y pesos operandos de EW con rango de -5 a 5

En las Figuras 22, 23 , 24 y 25 se muestran los histogramas obtenidos. En estos, solo se consideran las activaciones y los pesos que actúan en el modelo del ViT como entradas a la operación EW. Cada uno de estos histogramas representa la misma distribución de datos pero en distintas escalas de interés. El primer histograma presente en la Figura 23 fue programado para que la escala fuera calculada automáticamente y se mostrara el valor más positivo y más negativo de la distribución. De este histograma se puede concluir que el rango dinámico es

aproximadamente de 800 unidades. Se concluye así que el rango dinámico de FP16 IEEE754 es suficiente para representar los datos a cuantizar.

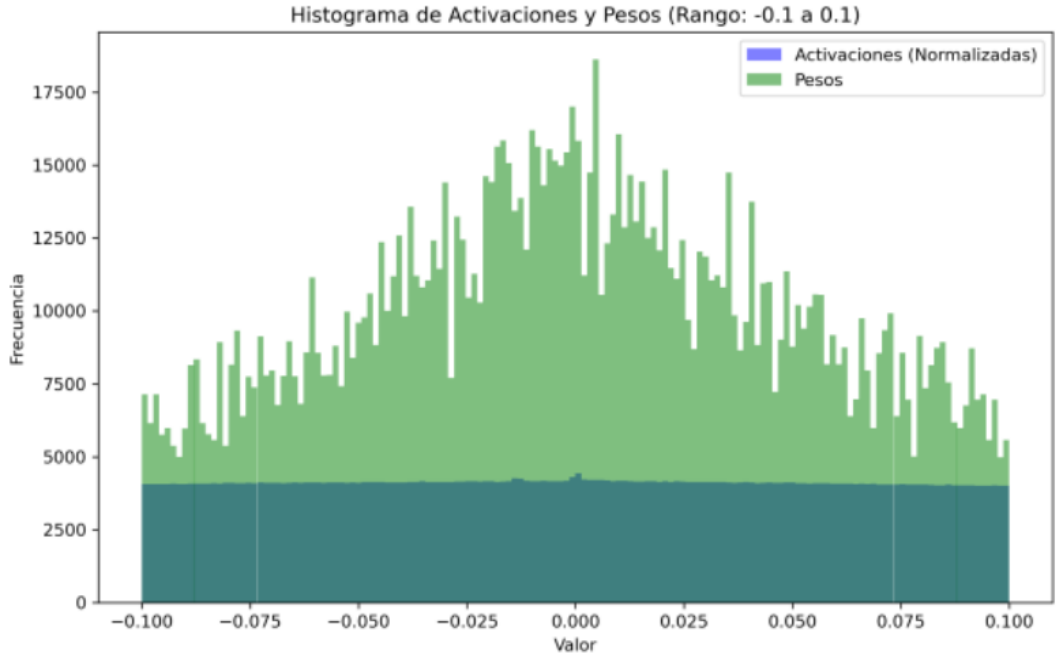


Figura 25. Histograma de activaciones y pesos operandos de EW con rango de -0.1 a 0.1

En la Figura 24 y Figura 25 se puede observar como la distribución de los datos tiene la mayor densidad en los datos cercanos a cero, y como esta densidad va aumentando paulatinamente conforme los datos se acercan a cero. Esto también es un indicativo de que la cuantización FP16 es una buena alternativa para cuantizar el conjunto de datos, ya que la precisión de FP16 cerca de cero es muy alta y va disminuyendo progresivamente conforme los datos se alejan de cero.

Finalmente, se puede observar en la Figura 26 como la distribución de datos en la escala de -0.01 a 0.01 ya no sigue algún patrón específico, es decir, no hay algún punto en el cual se observe una mayor densidad. Aunque la distribución no es uniforme, la distribución de los datos muestra que hay una misma probabilidad, o al menos una probabilidad muy similar, que un dato específico se encuentre en cualquier punto del rango. De esta observación se puede concluir que no es necesario que la precisión siga aumentando notablemente al acercarse más a cero, y que la precisión de 5.97x10-8 de FP16 IEEE754 es suficiente para representar esta distribución estadística. De esta manera, se concluye que la división de 1 bit de signo, 5 de exponente y 10 de mantisa es adecuada para representar el conjunto de datos que interactúan con el acelerador de HW.

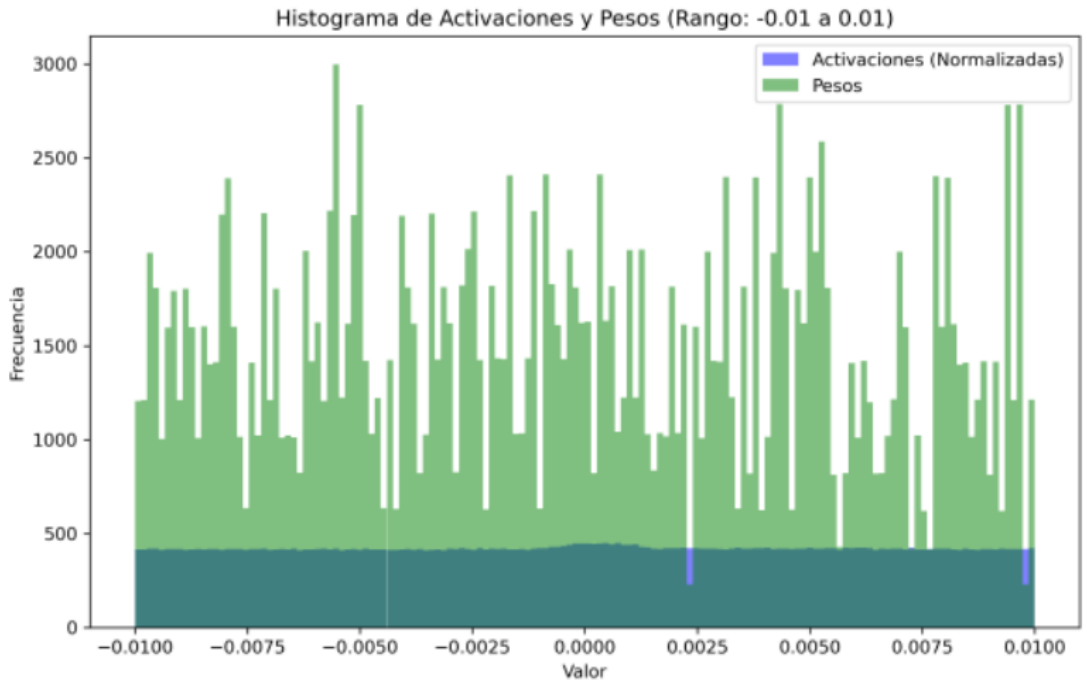


Figura 26. Histograma de activaciones y pesos operandos de EW con rango de -0.01 a 0.01

5.4 Implementación de la cuantización FP16 en el código del acelerador EW

La implementación del acelerador EW FP32 en C++ se compone de 3 funciones,:

1) Load_input: La primer función tiene como objetivo primordial cargar los datos de entrada desde una memoria externa a un flujo de datos "*stream"*, para posteriormente procesarlos. Esta función convierte los datos estáticos de la memoria externa en el formato de un flujo, lo cual permite un procesamiento continuo y eficiente de los datos de entrada.
2) Compute: La función realiza las operaciones aritméticas EW entre los dos conjuntos de datos que salen de la función Load_input como un flujo. Su objetivo es aplicar la operación de suma o multiplicación en paralelo, lo cual optimiza la velocidad de procesamiento.
3) Store_result: Finalmente Store_result almacena los resultados desde un flujo de datos recibido por Compute de nuevo hacia la memoria externa. Esta función cierra el flujo de procesamiento.

Debido a que solo el acelerador EW será utilizado por el ViT, la nueva implementación cuantizada del acelerador debe recibir datos en formato FP32, cuantizar los datos a FP16, operar sobre ellos, y luego volver a descuantizar los datos a FP32 nuevamente, para enviarlos de nuevo a la CPU. De esta manera, el código de SW, que trabaja todos sus datos en FP32, puede continuar

operando sobre ellos. De las tres funciones, la de mayor interés para la implementación de la cuantización FP16 corresponde a Compute, ya que es la que se debe modificar y a la que se deben agregar las funcionalidades de cuantización y descuantización.

Se introdujeron dentro de la función compute cinco funciones más para obtener el comportamiento deseado: quantize, execute_add, execute_multiply, dequantize y process. El código que introduce estas funciones se denomina elementwise.cpp y puede ser accesado por medio del Apéndice E. Se implementó una función para cada operación debido a que, en muchas circunstancias, la división de una función madre en varias subfunciones ayuda a que el diseño se comporte como un pipeline. Las funciones quantize, execute_add, execute_multiply y dequantize se muestran en la Figura 27 y son cortas y sencillas. La función quantize recibe como parámetros un dato de tipo FP32 y un dato de tipo FP16 y, a nivel de C++, simplemente realiza un cast de este dato de entrada FP32 al dato de salida FP16. La función dequantize tiene exactamente al mismo funcionamiento, pero viceversa, y las funciones execute_add y execute_multiply reciben dos datos

```
static void quantise(const float input_op, half &output_op) {
#pragma HLS INLINE off
  output_op = (half)input_op;
}

static void dequantise(const half input_op, float &output_op) {
#pragma HLS INLINE off
  output_op = (float)input_op;
}

static void execute_add(const half operand1, const half operand2, half &result){
#pragma HLS INLINE off
result=operand1+operand2;
}

static void execute_multiply(const half operand1, const half operand2, half &result){
#pragma HLS INLINE off
result=operand1*operand2;
}
```

Figura 27. Funciones quantize, execute_multiply, execute_add y dequantise del código en C++ del acelerador EW FP16

de tipo FP16 y realizan la operación aritmética entre ellos.

Como se puede observar en la Figura 27 todas las funciones implementan pragmas HLS INLINE off. Estos pragmas se utilizan para que estas funciones se mantengan como bloques independientes cada vez que la función Compute las llama, lo cual permite que se reutilice la misma instancia del HW de la función. Las funciones quantize, dequantize, execute_add y

execute_multiply son llamadas dentro de bucles for con muchas repeticiones, por lo que es fundamental utilizar este pragma para que no haya duplicación innecesaria de los recursos de FPG. La función process simplemente declara las variables necesarias para ejecutar las funciones execute, quantize y dequantize, tal como se presenta en la Figura 29.

Inicialmente se intentó de no implementar la función process, y en lugar de eso incorporar este código dentro de la función compute, pero al compilar y sintetizar el acelerador se obtuvo un error en el proceso de linking, más especificamente durante la generacion del bitstream. El error obtenido se muestra en la Figura 28. El error "[VPL 101-2] – Design failed to meet timing" implica que las rutas críticas del circuito no pudieron satisfacer las restricciones temporales para la frecuencia especificada, es decir, el tiempo de propagación a lo largo de una "etapa" del pipeline, la cual corresponde a una ruta de circuitos combinacionales, es mayor que el periodo de un ciclo de reloj.

```
===>The following messages were generated while  Compiling (bitstream) accelerator binary: design.link
 Log file: /home/gblanco/hls-fpga-accelerators/elementwise/tmp.hw/link/vivado/vpl/prj/prj.runs/impl_Ar
ea_Explore/runme.log  :
ERROR: [VPL 101-2] design did not meet timing - Design failed to meet timing.
    Failed timing checks (paths):
        {system_i/elementwise_1/inst/compute_U0/grp_compute_Pipeline_execute_fu_62/grp_process_r_fu_14
8/grp_execute_add_fu_74/hadd_16ns_16ns_16_4_full_dsp_1_U25/elementwise_hadd_16ns_16ns_16_4_full_dsp_1_
ip_u/inst/i_synth/ADDSUB_OP.ADDSUB/DSP.OP/B_IP_DELAY/i_pipe/opt_has_pipe.first_q_reg[5]/C --> system_i
/elementwise_1/inst/compute_U0/grp_compute_Pipeline_execute_fu_62/grp_process_r_fu_148/grp_execute_add
_fu_74/hadd_16ns_16ns_16_4_full_dsp_1_U25/elementwise_hadd_16ns_16ns_16_4_full_dsp_1_ip_u/inst/i_synth
/ADDSUB_OP.ADDSUB/DSP.OP/DSP48E1_BODY_HP.EXP/NOT_LOW_LATENCY_NORM_DIST.NORM_EXP_DELAY/i_pipe/opt_has_p
ipe.first_q_reg[5]/D}
```

Figura 28. Error de timing en la generación del bitstream para el acelerador EW FP16

Se logró eliminar el error de timing por medio de dos estrategias que se ejecutaron de forma simultánea. La primera estrategia consistió en la implementación de la función process. Aunque a primera impresión pudiese parecer que esta función no es necesaria, su implementación contribuye a que se generen más etapas de pipeline. Esto conlleva que las rutas críticas se subdividan en más secciones y, por lo tanto, el tiempo de propagación a lo largo de cada una de estas secciones disminuya.

La segunda estrategia consistió en modificar la frecuencia del reloj. El acelerador EW FP32 fue originalmente compilado con una directriz que especificaba una frecuencia de 200MHz. La "Kria" tiene la posibilidad de usar una frecuencia de 100MHz, 200MHz o 300MHz. Estas

frecuencias son seleccionadas por medio de flags en el comando de compilación. La opción mas intuitiva, y por lo tanto, la primera a implementar, fue disminuir la frecuencia de reloj a 100MHz para que estas rutas críticas cuenten con más tiempo para que la señal se propague. Sin embargo, aún con una frecuencia de 100MHz y la adición de la función “process” el error de “timing” en la síntesis se siguió presentando. Luego, se utilizó con una frecuencia de 300MHz y esto solucionó el error de timing y permitió la síntesis exitosa del acelerador EW con cuantización FP16.La configuración de la frecuencia del reloj se configura por medio de la modificación del archivo “Makefile”, el cual está presente en el Apéndice C del presente infrome

```
static float process(float in1, float in2, int op) {
#pragma HLS INLINE off
#pragma PIPELINE

  //1 Quantise (FP32->FP16)
  half input1_f16, input2_f16;
  quantise(in1, input1_f16);
  quantise(in2, input2_f16);

  //2 Execute
  half result_f16;
  switch (op) {
    case OP_ADD:
        execute_add(input1_f16,input2_f16,result_f16);
        break;
    case OP_MULT:
        execute_multiply(input1_f16,input2_f16,result_f16);
        break;
    default:
        break;
   }


  //3 Dequantise (FP16->FP32)
  float result_f32 = 0.f;
  dequantise(result_f16,result_f32);

  return result_f32;
}
```

Figura 29. Función "process" del código en C++ del acelerador EW FP16

Aunque el aumentar la frecuencia a 300MHz parece contraintuitivo, ya que el periodo de reloj se disminuye, al tomar esta estrategia es probable que el sintetizador de Vitis reestructurara ciertas rutas lógicas para optimizarlas y cumplir con las nuevas restricciones de frecuencia. Probablemente, ciertas operaciones combinacionales fueron divididas en etapas más cortas de manera automática, es decir, se aumentaron las etapas de pipelining. Esto permite que las rutas

críticas se subdividan y vuelvan más pequeñas, lo cual permite que el tiempo de propagación sea menor al periodo de reloj de 3.33ns.

```
static void compute(hls::stream<RawDataT> &in1_stream,
                    hls::stream<RawDataT> &in2_stream,
                    hls::stream<RawDataT> &out_stream, uint64_t size, int op) {
// The kernel is operating with vector of NUM_WORDS integers. The + operator
// performs an element-wise add, resulting in NUM_WORDS parallel additions.
execute:
  for (uint64_t i = 0; i < size; i += kPackets) {
#pragma HLS LOOP_TRIPCOUNT min = kTotalMaxSize max = kTotalMaxSize avg =       \
    kTotalMaxSize
#pragma HLS PIPELINE
    RawDataT raw_in1 = in1_stream.read();
    RawDataT raw_in2 = in2_stream.read();
    RawDataT raw_out = 0;
    for (int p = 0; p < kPackets; ++p) {
#pragma HLS UNROLL
      // Offsets
      const int offlow = p * kDataWidth;
      const int offhigh = offlow + kDataWidth - 1;

      // Data
      AccT in1, in2, out;
      in1.i = raw_in1(offhigh, offlow);
      in2.i = raw_in2(offhigh, offlow);

      out.f = process(in1.f, in2.f, op);

      // Write back
      raw_out(offhigh, offlow) = out.i;
    }
    out_stream << raw_out;
  }
}
```

Figura 30. Función "compute" del código en C++ del acelerador EW FP16

Como se ha mencionado anteriormente, una de las principales ventajas de utilizar High-Level Synthesis (HLS) es la posibilidad de diseñar hardware complejo sin necesidad de definir explícitamente los circuitos lógicos ni las conexiones eléctricas necesarias para implementar el cuantizador, decuantizador, sumador y multiplicador. En lugar de esto, HLS se encarga de manera automática y optimizada de generar toda la lógica de estos circuitos con solo definir las funciones en C++. Todos los códigos utilizados para la síntesis del diseño pueden ser accesados en el Apéndice C el cual puede ser accesado por medio del link al final de este documento.

6. Resultados y Análisis

6.1 Pruebas de Validación: Precisión y Tiempo de inferencia

Con el fin de medir el rendimiento del acelerador EW FP32 incorporado al ViT que hace y el rendimiento del acelerador EW FP16 diseñado y poder comparar ambos resultados, se definieron pruebas de validación que miden las tres características de interés: Tiempo de inferencia, precisión de clasificación top-1 y potencia disipada.

Básicamente, se diseñó un script en el Shell de Linux, el cual realiza la inferencia de imágenes de manera cíclica y toma mediciones de potencia disipada, tiempo de inferencia y calcula la precisión del modelo al finalizar la inferencia de todas las imágenes. Las imágenes sobre las cuales se realiza la inferencia corresponden a imágenes seleccionadas de manera aleatoria del set de datos Image1knet de categorías que se relacionan a ADAS, es decir, objetos que se encuentran comúnmente en carreteras.

La precisión corresponde a una variable aleatoria binomial, por lo que el tamaño de muestra puede ser calculado haciendo uso de la ecuación 22 [54]. En esta, "n" representa el tamaño de muestra necesario, "p" la precisión esperada, "e" el margen de error tolerado y "Z" el valor crítico que es determinado por la confiabilidad deseada.

$$n = \frac{Z^2 * p * (1 - p)}{e^2} \qquad (22)$$

El porcentaje de precisión esperada es de 73.85%, tal como se muestra en la tabla 5. La confiabilidad deseada se elige en 95%, que corresponde a un valor común en este tipo de experimentos, lo cual corresponde, si se asume una distribución normal de la variable aleatoria, a un valor crítico de 1.96. El margen de error tolerado se determina que debe ser al menos menor al 4%, de esta manera, se tiene una certeza del 95% de que el valor de precisión medido no difiere en más de un 4% al valor real de precisión. Se determina entonces utilizar una muestra de 650 imágenes, la cual corresponde a un margen de error de 3.4%.

6.2 Simulación por medio de Vitis: Potencia disipada por el acelerador

Por otro lado, y a diferencia de la precisión y el tiempo de inferencia, la medición directa de la potencia en runtime al ejecutar la inferencia del ViT presenta varias dificultades. El SoM utilizado (System on module) no está diseñado para permitir una separación física o eléctrica clara entre la FPGA y sus otros componentes, lo cual imposibilita obtener medidas de potencia, voltaje o corriente, específicamente de solo la FPGA, por medio de la línea de comandos del SoM.

Otra alternativa para medir la potencia de forma directa es utilizar un multímetro o medidor de potencia para analizar el voltaje y la corriente de la FPGA mediante una conexión directa a los pines de entrada de alimentación. Sin embargo, el dispositivo se encuentra actualmente en la ciudad de Liberia, en Guanacaste, lo cual imposibilita su acceso. Por estas razones, para obtener la potencia disipada por el acelerador, se hizo uso de herramientas de simulación disponibles en Vitis HLS.

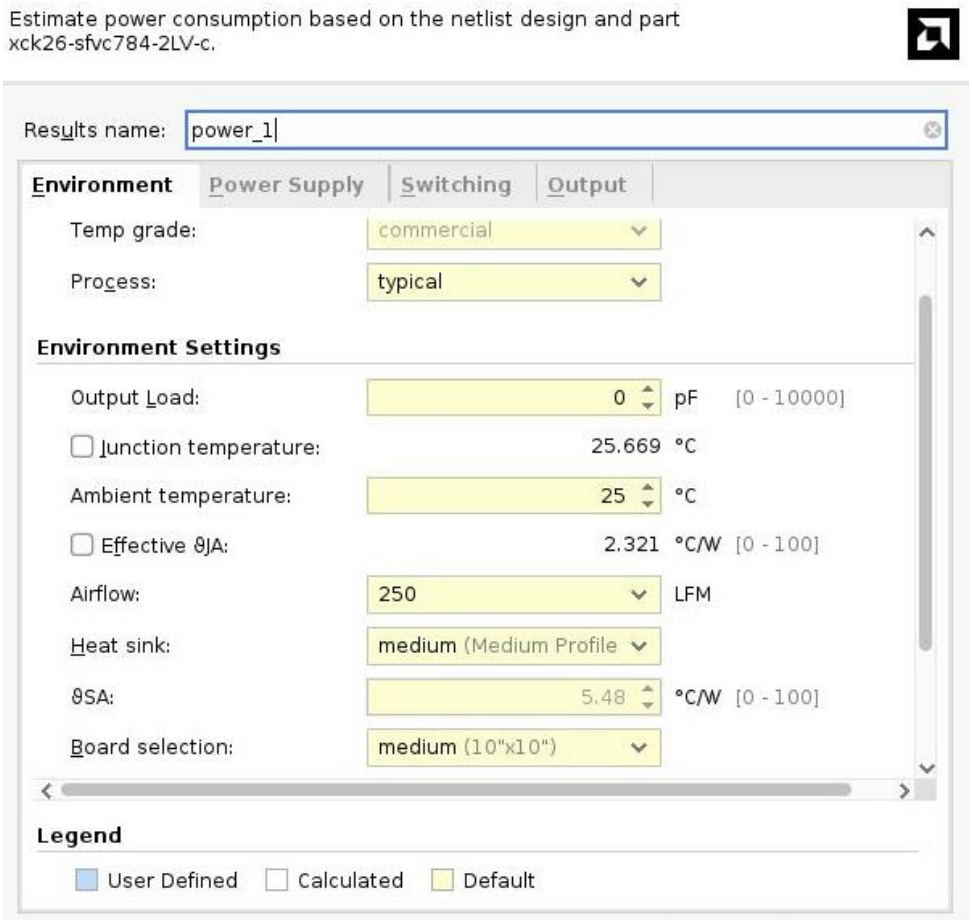


Figura 31. Condiciones ambientales para simulación de la potencia disipada por el acelerador EW FP16

Las condiciones de simulación utilizadas fueron las predeterminadas de la herramienta, las cuales se muestran en la Figura 31 y corresponden a condiciones conservadoras y comunes: Temperatura ambiente de 25°C, un flujo de aire de 250 Pies lineales por minuto, un disipador de calor de perfil medio y un valor de resistencia térmica de 5.4°C/W, la cual corresponde a una resistencia típica para disipadores de perfil medio.

6.3 Resultados: Precisión, Tiempo de inferencia, Potencia disipada y Recursos computacionales.

6.3.A)Precisión

En esta sección se presentan los resultados que caracterizan el rendimiento de la optimización propuesta y se analizan de manera objetiva. Primeramente, en la Figura 32 se observa los resultados de precisión top-1 y tiempo promedio de inferencia. Como se puede observar, la precisión del modelo está reportada como un 73%. Realmente, la precisión es un poco más elevada, de un 73.85%, que se obtiene al dividir 650 inferencias totales entre 480. El dato reportado indica 73% por un problema del código el cual realiza la división entera, en lugar de fraccionaria, del número de aciertos entre el número de intentos totales. Esta precisión es exactamente igual a la obtenida con el acelerador original que opera en FP32.

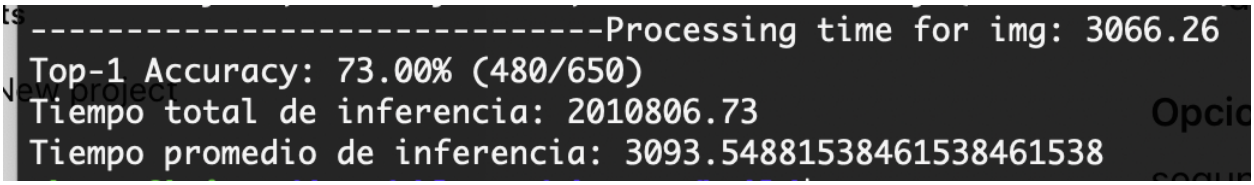


Figura 32. Resultados de precisión top1 y tiempo de inferencia promedio para la solución propuesta.

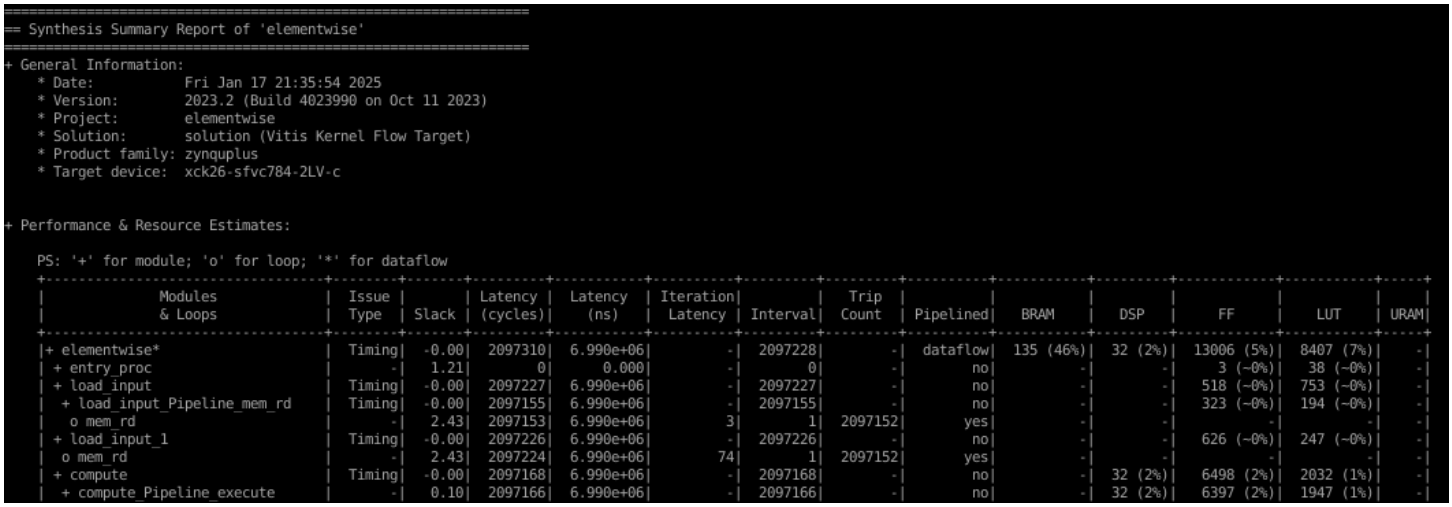
Un punto interesante a analizar, es que no solo se mantuvo la precisión de clasificación, sino que también, al analizar todas las inferencias, todas se ejecutaron de la misma forma, o sea, la clase en la cual el algoritmo clasificó a cada imagen del set de 650 datos fue exactamente la misma. Esto indica que la elección de la cuantización tipo FP16 fue óptima en términos de precisión. El algoritmo, aunque no se comportó de manera completamente idéntica desde un punto de vista aritmético, se comportó completamente igual en términos de su funcionalidad y objetivo principal, que es la clasificación de imágenes.

Se puede concluir de esta caída nula de precisión que el modelo del ViT es realmente robusto ante cuantizaciones poco agresivas como lo es FP16. Además, también se concluye que

las características aprendidas por el modelo durante el proceso de entrenamiento tienen una margen de tolerancia suficiente a errores numéricos, lo cual permite que las predicciones se mantengan completamente estables.

Por otra parte, la robustez de la precisión del modelo también indica que el rango dinámico y la precisión de la configuración elegida de 5 bits de exponente, 10 bits de mantisa y 1 bit de signo, son suficientes, tal como se había hipotetizado en el capítulo 5 por medio del análisis de los histogramas. Aunque se notó en estos histogramas que la precisión del modelo FP32 (en este contexto, el número más pequeño del histograma distinto de cero) era bastante menor que la de FP16. La hipótesis que se basa en la Figura 26, la cual afirma que no es necesaria una precisión tan pequeña porque la distribución de los datos a esta escala se asemeja a una distribución aleatoria se prueba como correcta.

Finalmente, con respecto a la precisión, y sin necesidad de analizar los resultados relacionados a potencia y cantidad de recursos utilizados, se puede concluir que, es posible explorar cuantizaciones más agresivas como INT8, INT4, o similares. El hecho de que se haya mantenido estable la precisión, y además la clasifiación top-1 haya sido la misma para todos los casos, sugiere que la red tiene suficiente tolerancia al error como para soportar una reducción adicional en la precisión numérica sin necesidad de observar una reducción notable de la precisión de clasificación.



```
================================================================
== Synthesis Summary Report of 'elementwise'
================================================================
+ General Information:
    * Date:           Fri Jan 17 21:35:54 2025
    * Version:        2023.2 (Build 4023990 on Oct 11 2023)
    * Project:        elementwise
    * Solution:       solution (Vitis Kernel Flow Target)
    * Product family: zynquplus
    * Target device:  xck26-sfvc784-2LV-c


+ Performance & Resource Estimates:

    PS: '+' for module; 'o' for loop; '*' for dataflow
    +---------------------------------+--------+-------+---------+----------+---------+---------+---------+----------+----------+---------+------------+-----------+-----+
    |             Modules             | Issue  |       | Latency | Latency  | Iteration|         |  Trip   |          |          |         |            |           |     |
    |             & Loops             | Type   | Slack | (cycles)|   (ns)   |  Latency | Interval| Count   | Pipelined|   BRAM   |   DSP   |     FF     |    LUT    | URAM|
    +---------------------------------+--------+-------+---------+----------+---------+---------+---------+----------+----------+---------+------------+-----------+-----+
    |+ elementwise*                   | Timing| -0.00| 2097310| 6.990e+06|        -| 2097228|        -| dataflow| 135 (46%)| 32 (2%)| 13006 (5%)| 8407 (7%)|    -|
    | + entry_proc                    |       -|  1.21|        0|     0.000|        -|        0|        -|        no|         -|        -|     3 (~0%)|   38 (~0%)|    -|
    | + load_input                    | Timing| -0.00| 2097227| 6.990e+06|        -| 2097227|        -|        no|         -|        -|   518 (~0%)|  753 (~0%)|    -|
    |  + load_input_Pipeline_mem_rd   | Timing| -0.00| 2097155| 6.990e+06|        -| 2097155|        -|        no|         -|        -|   323 (~0%)|  194 (~0%)|    -|
    |   o mem_rd                      |       -|  2.43| 2097153| 6.990e+06|        3|        1| 2097152|       yes|         -|        -|           -|          -|    -|
    | + load_input_1                  | Timing| -0.00| 2097226| 6.990e+06|        -| 2097226|        -|        no|         -|        -|   626 (~0%)|  247 (~0%)|    -|
    |  o mem_rd                       |       -|  2.43| 2097224| 6.990e+06|       74|        1| 2097152|       yes|         -|        -|           -|          -|    -|
    | + compute                       | Timing| -0.00| 2097168| 6.990e+06|        -| 2097168|        -|        no|         -| 32 (2%)|  6498 (2%)| 2032 (1%)|    -|
    |  + compute_Pipeline_execute     |       -|  0.10| 2097166| 6.990e+06|        -| 2097166|        -|        no|         -| 32 (2%)|  6397 (2%)| 1947 (1%)|    -|
```

Figura 33. Resumen de reporte de síntesis "Csynth.rpt" generado por Vitis para el acelerador elementwise FP16

6.3.B) Tiempo de inferencia

Con respecto al tiempo promedio de inferencia se puede observar que este se mantiene practicamente igual a el ViT con el acelerador EW FP32. En muchos casos, como se evidencia en el marco teórico de este informe, una disminución de la cantidad de bits disminuye la latencia de inferencia del modelo, ya que las operaciones en FP16 comúnmente requieren de menos ciclos de reloj debido al menor tamaño de datos.

En este caso específico surge la incógnita de porqué el tiempo de inferencia se mantuvo practicamente igual. Es probable que la latencia se haya mantenido debido a las modificaciones de pipeline que el compilador HLS tuvo que realizar para cumplir con los requerimientos de timimng. En este caso, la latencia probablemente se ve determinada por la estructura y la profundidad del pipeline en lugar de por el tamaño del tipo de dato. Además, las decisiones de diseño de agregar forzosamente más etapas de pipeline por medio de funciones adicionales en C++ y la modificación de la frecuencia del sistema de 200MHz a 300MHz también pudieron haber influido en este resultado.

Tabla 7. Recursos Computacionales de la solución propuesta.

	BRAM	DSP	Flip-Flops	Look-Up Tables
Cantidad	135	32	13006	8407
Porcentaje de utilización	46%	2%	6%	8%
Porcentaje de disminución	0%	20%	8.01%	15.89%

6.3.C) Recursos Computacionales

En la Figura 33 y en la Tabla 7 se muestra de manera resumida los resultados relacionados al uso de recursos computacionales del acelerador EW FP16. Se puede notar una disminución considerable en la cantidad de DSPs (20%) y LUTs (15.89%) y una disminución moderada en la cantidad de Flip Flops (8%). Sim embargo, la cantidad de bloques de memoria BRAM se mantienen completamente constantes. La disminución de las LUTs y DSPs se atribuye a una simplificación directa del HW necesario para realizar las operaciones aritméticas. Los circuitos digitales de sumadores y multpilicadores de 16 bits requieren de menos componentes lógicos y generan condiciones más compactas. Esto presenta ventajas ya que al liberarse tantos DSPs y

LUTs se permite una mayor paralelización para futuros diseños y también permite utilizar estos recursos en otras funcionalidades que futuros diseños puedan necesitar.

El hecho de que la BRAM no haya variado parece contra intuitivo, pero al realizar un análisis más profundo se pueden descubrir diversas causas. La principal de ellas es el hecho de que, tal como se explica en el capítulo 5, los accesos a la memoria interna de la BRAM se generan con las funciones "Load_input y Store_result", las cuales generan el stream de datos. Estas funciones no fueron modificadas del todo, ya que, por las razones que ya se han explicado anteriormente, los datos se reciben en formato FP32. Por esta razón, estas dos funciones mantienen exactamente la misma funcionalidad y configuración que con el acelerador EW FP32, lo cual explica que se necesite la misma cantidad de BRAMs ya que los datos cuentan con el mismo formato y la misma cantidad de bits.

6.3.D) Potencia disipada

Finalmente, con respecto a la potencia estimada por parte de la simulación de Vitis, es necesario realizar un cálculo de potencia siguiendo el mismo procedimiento matemático que se siguió para calcular la potencia del acelerador EW FP32. Se debe de restar a la suma del valor dinámico y estático de potencia la potencia reportada como PS ya que, como se mencionó anteriromente, esta potenica corresponde a la disipada por el CPU interno ARM del SoM el cual no tiene ninguna relación con el acelerador implementado. Entonces, como se evidencia en la Figura 34, la potencia total simulada corresponde a 2.876-2.550+0.303=0.629W o 629mW, lo cual refleja una disminución del 4.55% en el consumo de potencia.

El resultado obtenido se acerca pero no cumple con el valor marginal estipulado en las especificacioes del proyecto y, además, esta disminución no alcanza los resultados que reportan los estudios del estado del arte del capítulo del marco teórico. Esto se puede deber a varias razones, pero principlamente se atribuye a que el acelerador de HW EW FP32 ya había sido diseñado por el equipo del ECAS lab para utilizar la menor cantidad de energía posible. Esta disminución del 4.55% se atribuye en su totalidad al uso de la cuantización y no a modificaciones arquitectonicas o de funcionamiento fundamental del acelerador, ya que este ya se encuentra lo más paralelizado optimizado posible en estos ámbitos. Se concluye de este resultado de potencia y de el de precisión

que para futuras iteraciones es posible y recomandable utilizar un tipo de cuantización màs agresiva, tal como INT8 o INT4.

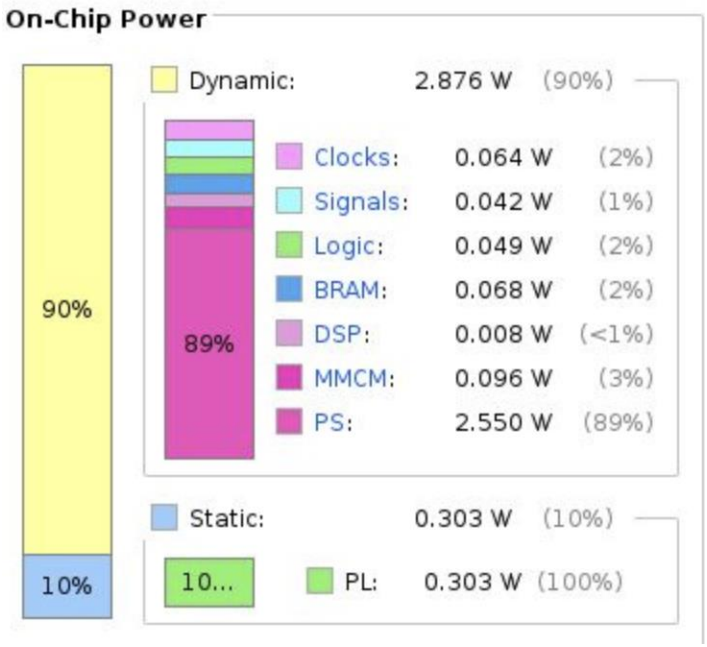


Figura 34. Estimación de uso de potencia de la solución propuesta

6.4. Cumplimiento de las especificaciones propuestas

En la tabla 8 se presentan nuevamente las especificaciones planteadas inicialmente. En esta tabla se representa en la última columna en color verde las especificaciones que alcanzaron los valores ideales, en amarillo las que alcanzaron el rango de los valores marginales y en rojo las especificaciones que no lograron alcanzar el valor marginal.

Tabla 8. Valor obtenido de la solución implementada para cada una de las especificaciones

# de Métrica	Métrica	Importancia	Unidad de medida	Valor Marginal	Valor Ideal	Valor Obtenido
1	Potencia disipada al realizar inferencias	3	Miliwatts (mW)	Dis. [6-10]%	Dis. >10%	4.55%
2	Cantidad de LUTs	2	Medida discreta	Dis. [8-20]%	Dis. >20%	15.89%
3	Cantidad de DSP	3	Medida discreta	Dis. [8-20]%	Dis. >20%	20%
4	Cantidad de Bloques BRAM	3	Medida discreta	Dis. [8-20]%	Dis. >20%	0%
5	Cantidad de Flip-Flops	2	Medida discreta	Dis. [8-20]%	Dis. >20%	8.01%
6	Precisión top 1 de inferencia del ViT.	3	% de precisión	Dis. [5-2]%	Dis. <2%	0%
7	Latencia	2	Milisegundos (ms)	Aum. [50-20]%	Aum. <20%	1%
8	Implementado usando HLS	3	Binario	True	True	True
9	Uso de los aceleradores de HW previamente diseñados en el ViT	3	Binario	True	True	True

Como se puede observar, la mayoría de métricas alcanzan el valor ideal. Dos de ellas alcanzan el rango de valores marginales y dos no alcanzan el valor marginal. Estas dos métricas que no alcanzan el valor marginal tienen un nivel de importancia alto. Primero, para mejorar la

potencia disipada, se puede concluir que para futuras mejoras del sistema es imprescindible elegir un método de cuantización más agresivo. Con respecto a la métrica de cantidad de bloques BRAM, se teoriza que para futuras iteraciones se puede intentar de implementar un acelerador que ya reciba los datos cuantizados. Así, los bloques de BRAM manejarán datos de 16 bits para FP16, o incluso menos para otras cuantizaciones más agresivas, lo cual disminuirá la cantidad de bloques BRAM necesarias para el acelerador.

6.5. Análisis Financiero

A continuación, se presenta un análisis financiero y del desarrollo del proyecto en términos generales. El propósito principal de este análisis es detallar los egresos asociados con la ejecución del proyecto en un entorno comercial o industrial, así como los ingresos que podrían generarse al implementar las optimizaciones propuestas en dicho contexto.

Cabe destacar que, debido al enfoque académico de esta investigación, su principal objetivo fue enriquecer la literatura en el área de optimización de DNNs implementadas en FPGA, en lugar que generar un beneficio económico directo o inmediato. No obstante, este trabajo proporciona una base teórica y práctica que podría ser aprovechada en investigaciones futuras para obtener beneficios económicos más significativos. Asimismo, la posible escalabilidad de estos proyectos a sistemas masivos podría traducirse en impactos económicos y ambientales sustanciales a gran escala.

6.5.A) Egresos

Primeramente se enumeran los egresos necesarios para llevar a cabo el desarrollo del proyecto. Como se mencionó en el capítulo 3, los dispositivos electrónicos necesarios para efectuar el proyecto son principalmente 2: Una CPU que pueda ejecutar el SW de Vitis HLS para sintetizar los aceleradores de HW y una FPGA en la cual ejecutar y probar los aceleradores de HW. Tomando en consideración la CPU utilizada para este proyecto, la cual corresponde a una E3-1270 de 4 núcleos y 3.8GHz, se buscaron computadoras en el mercado que utilizaran este CPU y tuvieran caracteristicas similares a la Computadore utilizada. La E3-1270 es una computadora con ya unos cuantos años de antigüedad, por lo que ya no se venden workstations nuevas con esta CPU. Sin

embargo, se realizó una busqueda breve en el mercado de segunda mano, y se encontró que el precio de la Dell Precision T1600, un workstation que posee esta CPU, ronda los 300$.

Por otra parte, el SoC utilizado, la "Kria" tiene un precio de 422$ según el distribuidor "Digikey" [56] lo cual equivale a 216.583 colones utilizando el tipo de cambio del BCCR del 3 de enero del 2025. Con respecto al acceso al SW Vitis HLS, la licencia a este SW varía en precio, pero el acceso a xrt France permite su uso de manera gratuita por lo que no es necesario tomar en consideración este egreso. Igualmente el set de datos imagenet1k es de acceso gratuito por lo que este gasto también se considerará de 0 dolares.

La mayor parte de los costos del presente proyecto corresponden a los gastos operativos. El proyecto fue realizado por un ingeniero "Junior" sin experiencia previa laboral ni conocimientos técnicos en el área de desarrollo e implementación de HW programable por medio de HLS. Por esta razón, un periodo considerable del proyecto correspondió a que el ingeniero se familiarizara con al menos la teoría fundamental de esta área. El proceso para el desarrollo del proyecto fue de aproximadamente 6 meses, desde Julio del 2024 hasta enero del 2025.

Durante todo este proceso, el ingeniero Junior tuvo a disposición un ingeniero "Senior" con alto nivel de conocimiento en el área el cual posee una maestría y, para enero del 2025, un doctorado en el área. Este ingeniero fue guiando todo el proceso de desarrollo del ingeniero Junior. Se considera que para una empresa sería mucho más rentable simpelmente contratar al ingeniero Senior para que llevase a cabo el proyecto porque este lo podría finalizar en un periodo de tiempo mucho más corto, y además porque en el caso de contratar un ingeniero Junior, también se tendrían que pagar horas de consulta del ingeniero Senior para que este brinde acompañamiento y guía.

Se calcula entonces que el ingeniero Senior podría realizar la totalidad del proyecto en un periodo de 2 meses. El salario para un ingeniero Senior varía, pero se utiliza una aproximación del salario bruto de 2.500.000, o sea un salario neto de 2.002.300 al considerar las cargas sociales de la Caja Costarricense del Seguro Social y el impuesto de renta. El salario mensual bruto en dolares corresponde a 4871.11$ con el tipo de cambio de 513.23 colones por dolar del BCCR del 3 de enero del 2025.

En la Tabla 9 se muestra un desglose de los gastos necesarios para ejecutar el proyecto así como la sumatoria de todos estos gastos. El precio de la electricidad necesaria para correr la inferencia del ViT no se toma en cuenta ya que lo que interesa en términos del análisis económico es la utilidad que pueda llegar a tener el proyecto, y para calcular esta utilidad, como el proyecto

se basa en disminuir la energía eléctrica a utilizar, solo es necesario considerar la cantidad de disminución de potencia y tomar esta disminución para calcular los ingresos que podría generar el proyecto.

Tabla 9. Resumen de gastos del proyecto

Gastos	Monto Colones	Monto Dolares	Periodo de tiempo (meses)	Total en Colones
Gastos de Capital (CapEx)				
SoC "Kria"	216.583	422	-	216.583
Workstation "Dell Precision T1600"	153.969	300	-	152.969
Gastos operativos (OpEx)				
Salario Neto de Ingeniero Senior	2.002.300	3.901,37	2	4.004.600
Cargas Sociales del salario de Ingeniero Senior	497.700	969,74	2	995.400
TOTAL GASTOS (Colones)				5.369.552

6.5.B) Ingresos

Como se mencionó anteriormente, por el carácter investigativo del proyecto, este no genera utilidades económicas directas e inmediatas. Sin embargo, el proyecto a largo plazo puede generar muchos beneficios que a su vez pueden desencadenar en beneficios económicos, los cuales dependen directamente del nivel de escalamiento de la implementación, es decir, en cuantos dispositivos se implementen las optimizaciones.

Las optimizaciones propuestas por si solas e implementadas en pequeña escala parecen no tener una gran utilidad económica. Por ejemplo, si las optimizaciones propuestas son implementadas en un dispositivo de FPGA, y este realiza una inferencia, se puede calcular de manera sencilla el ahorro económico que se obtiene a partir del ahorro de la energía eléctrica. Para realizar este cálculo, se toman en cuenta los datos oficiales de la página web de la Comisión Nacional de Fuerza y Luz [55]. Para Tarifas Industriales T-IN y Tarifas de Comercios y servicios T-CO, a partir de un gasto mayor mensual de energía de los 3 mil kWh, la tarifa corresponde a ₡69,00 por kWh. Tomando en cuenta que el tiempo promedio de inferencia en el estado actual del

modelo es de 3s, cada inferencia equivaldría a un ahorro económico de solamente ₡0.001725, una cantidad prácticamente igual a cero. Si el modelo en lugar de simplemente realizar una inferencia, se utiliza para que realice la inferencia de forma cíclica durante 6 horas 6 días de la semana, el ahorro aumenta pero sigue siendo muy pequeño, y corresponde a ₡3.885 por año.

Ahora, siempre que se implementan modelos de visión por computadora en la industria, estos no se implementen en un solo dispositivo, si no en miles o hasta cientos de miles. Si las optimizaciones fuesen implementadas en 1000 dispositivos el ahorro correspondería a ₡3.885.685 anuales, lo cual implicaría una recuperación de la inversión inicial (los gastos reportados en la Tabla 9) en 1.4 años. Si las optimizaciones se implementaran de manera masiva en 100.000 dispositivos el ahorro sería de ₡388.565.500, lo cual implica un retorno de la inversión en solamente 0.0138 años o 5 días. En la tabla Tabla 10 se muestra un resumen del ahorro y el tiempo de retorno de inversión para distintas escalas de implementación, suponiendo que todas estas trabajan realizando inferencia de manera continua 6 horas al día 6 días a la semana.

Tabla 10. Resumen de ahorro económico y retorno de inversión para diferentes escalas de implementeación.

Cantidad de dispositivos	**Ahorro Anual**	**Tiempo de retorno de inversión**
1	3.885,00	1382 años
100	388.500,00	13.8 años
1.000	8.885.000,00	1.38 años
10.000	38.850.000,00	1.6 meses
100.000	388.500.000,00	5 días
1.000.000	3.885.000.000,00	Menos de 1 día

Se puede concluir entonces de la Tabla 10 que la utilidad económica depende de manera directa al nivel de escalamiento al cual se implementen las optimizaciones propuestas en este proyecto. Por ejemplo, para una empresa no valdría la pena implementar las optimizaciones solamente en 100 dispositivos, pero sería extremadamente rentable el aplicar la optimización en un millón, cien mil o inclusive diez mil dispositivos.

7. Conclusiones y Recomendaciones

7.1. Conclusiones

1) La cuantización FP16 demuestra ser excelente para mantener la precisión del modelo de Inteligencia Artificial “Vision Transformer”. Esta logra garantizar que las clasificaciones sean consistentes con respecto a la implementación en FP32 a pesar de la reducción de la precisión numérica.

2) La cuantización FP16 disminuye el uso de recursos computacionales y el consumo de potencia, sin embargo la disminución obtenida no logra cumplir con las expectativas iniciales. Para lograr una disminución de potencia más significativa, se deben explorar técnicas de cuantización más agresivas como INT8 o INT4, o explorar modificaciones en la arquitectura del acelerador.

3) La latencia no depende únicamente del tipo de cuantización utilizado, factores como el uso de técnicas de “pipelining” o modificaciones en la arquitectura tienen un impacto mucho mayor. Además la implementación del acelerador en el ViT, si no es optimizada, puede aumentar considerablemente la latencia. Por esto, una mejora en la implementación del acelerador EW en el ViT es clave para reducir el tiempo de inferencia.

4) En su estado actual, el modelo no puede ser utilizado en aplicaciones de ADAS debido a su alta latencia, lo cual es causado por la implementación de CPU-FPGA y el “broadcasting” que se ejecuta de manera no optimizada desde la CPU, lo que introduce overhead significativo.

5) Se demuestra que FP16 puede ser una excelente opción para aplicaciones de ADAS de baja energía que tienen como máxima prioridad mantener la precisión, siempre y cuando el modelo sea implementado en FPGA por completo y los aceleradores EW hayan sido diseñados para recibir datos en formato vector-matriz.

6) La transmisión de datos a través de interfaces de alta velocidad, como AXI, entre la CPU y la FPGA, debe ser minimizada tanto como sea posible para obtener latencias aceptables.

7) El modelo de Vision Tranformer (ViT) demuestra ser muy robusto frente a cuantizaciones poco agresivas como FP16, lo cual refuerza su viabilidad en aplicaciones de eficiencia energética, siempre y cuando las implementaciones de HW sean optimizadas.

8) Las optimizaciones propuestas pueden suponer un gran ahorro económico para una empresa que necesite un algoritmo de visión por computadora de clasificación de imágenes, siempre y cuando la implementación tenga una escala grande o masiva.

7.2. Recomendaciones

1) Se recomienda analizar, explorar y medir técnicas de cuantización más agresivas tales como INT8, INT4 o inclusive cuantización binaria. De esta manera se podrá reducir aún más el uso de recursos computacionales y el consumo de potencia.

2) Si es posible, se recomienda realizar mediciones directas del consumo de potencia usando herramientas físicas como multímetros en lugar de solo utilizar las simulaciones de Vitis. De esta manera se pueden obtener datos más confiables y realistas para el análisis del rendimiento energético.

3) Se recomienda investigar optimizaciones a nivel de arquitectura como pipelining más agresivo, partición de datos y estrategias avanzadas de síntesis y paralelismo, lo cual puede contribuir a disminuir aún más el consumo de potencia.

3) Se recomienda Implementar más aceleradores del ECAS Lab al ViT. De esta manera, se reducirá notablemente la latencia que en la actualidad es causada por la transferencia de datos constante entre CPU y FPGA por medio de la interfaz AXI.

5) Finalmente se recomienda diseñar un acelerador EW con soporte de broadcasting, el cual pueda recibir como operandos un vector y una matriz. Esto elimina la necesidad de que el CPU realice este proceso, lo que en la actualidad aumenta el tiempo de inferencia notablemente.

Referencias

[1] S. Castro, *Diseño de unidad de procesamiento configurable en FPGA para aceleración de convolución,* 2024.

[2] L. D. Prieto, *Desarrollo de un backend de ejecución para la biblioteca GGML,* Cartago, 2024.

[3] L. G. León, *Aceleración flexible de algoritmos basados en DNN en el Edge,* 2022.

[4] D. Alexey, L. Beyer, A. Kolesnikov, D. Weissenborn y X. Zhai, *AN IMAGE IS WORTH 16X16 WORDS:,* 2021.

[5] L. G. León, Interviewee, *Entrevista Reunión para plantear PFG.* [Entrevista]. Junio 2024.

[6] V. Ashish, S. Noam, P. Niki y U. Jakob, *Attention Is All You Need,* 2017.

[7] M. Nielsen, Neural Networks and Deep Learning: A Textbook, Determination Press, 2015.

[8] I. Goodfellow, Y. Bengio y A. Courville, Deep Learning, Cambridge: MIT Press, 2016.

[9] U. Sarwai, C. Iwendi y M. Mital, «Analyzing the efectiveness and Contribution of each Axis of Tri-Axis Accelerometer Sensor for Accurate Activity Recognition,» *Research Gate,* 2020.

[10] D. Hendrycks y K. Gimpel, «GAUSSIAN ERROR LINEAR UNITS (GELUS),» 2016.

[11] S. Harris y D. Harris, Digital Design and Computer Architecture. ARM Edition, Waltham: Morgan Kaufmann, 2016.

[12] C. Johnson, S. Hauck, H. Shih-Chieh y W. Khan, «Quantifying the Efficiency of High-Level Synthesis for,» 2023.

[13] A. Takach, «High-Level Synthesis: Status, Trends, and,» *IEEE,* 2016.

[14] AMD, «AMD Vitis™ HLS,» 2024. [En línea]. Available: https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vitis/vitis-hls.html.

[15] J. Han y M. Orshansky, «Approximate Computing: An Emerging Paradigm For,» *IEEE,* 2013.

[16] D. Ayad, H. Amjad, A.-M. Ammar y H. Al-Raweshidy, «Approximate Computing: Concepts, Architectures, Challenges, Applications and Future Directions,» *IEEE,* 2022.

[17] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. Mahoney y K. Keutzer, «A Survey of Quantization Methods for Efficient,» *IEEE,* 2021.

[18] I. Kadota y E. Modiano, «Lecture 5 - Quantization,» de *Communication Systems and Networks*, Cambridge, 2019.

[19] L. Wei, Z. Ma, C. Yang y Q. Tao, «Advances in the Neural Network Quantization:,» *IEEE,* 2024.

[20] Qualcomm Technologies Inc/, «Enabling power-efficient,» 2020.

[21] M. Nagel, M. Fournarakis, R. Ali Amjad y Y. Bondarenko, «A White Paper on Neural Network Quantization,» *IEEE,* 2021.

[22] D. Pohl, B. Vogel-Heuser, M. Kryger y M. Echtler, «Quantization Effects of Deep Neural Networks on a FPGA platform,» *IEEE,* 2024.

[23] Y. Yao, B. Dong, Y. Li, W. Yang y H. Zhu, «EFFICIENT IMPLEMENTATION OF CONVOLUTIONAL NEURAL NETWORKS WITH,» *IEEE,* 2019.

[24] Y. Zhang, B. Sun, W. Jian, Y. Ha, M. Hu y W. Zhao, «WSQ-AdderNet: Efficient Weight Standardization based Quantized AdderNet FPGA Accelerator Design with High-Denisty INT8 DSP-LUT Co-Packing Optimization,» *IEEE,* 2022.

[25] Intel Corporation, «Intel and Deci hit 11.8x acceleration at MLPer,» 30 Octubre 2020. [En línea]. Available: https://community.intel.com/t5/Blogs/Tech-Innovation/Artificial-Intelligence-AI/Intel-and-Deci-hit-11-8x-acceleration-at-MLPerf/post/1335729. [Último acceso: Setiembre 2024].

[26] R. Yates, «Fixed-Point Arithmetic: An Introduction,» *Digital Sound Labs,* 2001.

[27] S.-E. Chang, Y. Li, M. Sun, W. Jiang, R. Shi, X. Qian, X. Lin y Y. Wang, «MSP: An FPGA-Specific Mixed-Scheme, Multi-Precision Deep Neural Network Quantization Framework,» *IEEE,* 2020.

[28] P. Li y C. Che, «Mapping YOLOv4-Tiny on FPGA-Based DNN Accelerator by Using Dynamic Fixed-Point Method,» *IEEE,* 2021.

[29] V. Rajagopal, K. Ramasamy, A. Vishnoi y R. Gadde, «ACCURATE AND EFFICIENT FIXED POINT INFERENCE FOR DEEP NEURAL NEWTORKS,» *IEEE,* 2018.

[30] A. Sajid, H. Kyuyeon y S. Wonyong, «FIXED POINT OPTIMIZATION OF DEEP CONVOLUTIONAL NEURAL NETWORKS FOR OBJECT RECOGNITION,» *IEEE,* 2015.

[31] S. Wang y P. Kanwar, «BFloat16: The secret to high performance on Cloud TPUs,» 13 Agosto 2019. [En línea]. Available: https://cloud.google.com/blog/products/ai-machine-learning/bfloat16-the-secret-to-high-performance-on-cloud-tpus. [Último acceso: Setiembre 2024].

[32] J. Jhonson, «Rethinking floating point for deep learning,» *IEEE,* 2018.

[33] A. Hagiescu, M. Langhammer, B. Pasca, P. Colangelo, J. Thong y N. Ilkhani, «BFLOAT MLP Training Accelerator for FPGAs,» *IEEE,* 2019.

[34] T.-H. Tsai y D.-B. Lins, «An On-Chip Fully Connected Neural Network Training Hardware Accelerator Based on Brain Float Point and Sparsity Awareness,» *IEEE Open Journal of Circuits and systems,* 2023.

[35] S. Mishra, A. Tiwari, S. Shekhawat, P. Guha, P. Jan y Z. Nemec, «Comparison of Floating-point Representations for the Efficient Implementation of Machine Learning Algorithm,» *IEEE,* 2022.

[36] J. Gustafson y I. Yonemoto, «Beating Floating Point at its Own Game: Posit Arithmetic,» *SuperFri.org*, 2017.

[37] Z. Carmichael, H. Langroudi, C. Khazanov, J. Lillie, J. Gustafson y D. Kudithipudi, «Deep Positron: A Deep Neural Network Using the Posit Number System,» *IEEE*, 2019.

[38] IEEE Computer Society, 754-2019 - IEEE Standard for Floating-Point Arithmetic, New York, 2019.

[39] C. Mei, Z. Liu, Y. Biu y X. Ji, «A 200MHZ 202.4GFLOPS@10.8W VGG16 ACCELERATOR IN XILINX VX690T,» 2017.

[40] A. Talebi y M. Morteza, «Variable precision, mixed fixed/floating point MAC unit for DNN accelerators,» *The 4th Iranian International Conference on Microelectronics*, 2022.

[41] O. Shin-ichi, F. Hiroshi, I. Tsutomu, N. Wakana, M. Takashi y K. Tomohiro, «Image-Classifier Deep Convolutional Neural Network Training by 9-bit Dedicated Hardware to Realize Validation Accuracy and Energy Efficiency Superior to the Half Precision Floating Point Format,» 2018.

[42] S. Zhourui, D. Jiwu, Z. Wei y J. Xiangyang, «High-Performance FPGA-Based CNN Accelerator With Block-Floating-Point Arithmetic,» 2019.

[43] M. Antony y R. Whenish, «Advanced Driver Assistance Systems (ADAS),» *Research Gate*, 2021.

[44] Euro NCAP, «About Euro NCAP,» 2024. [En línea]. Available: https://www.euroncap.com/en/.

[45] C. Solís, *Diseño y validación de algoritmos de detección de objetos para la capa de percepción del Sistema Avanzado de Asistencia para Conducción (ADAS) en vehículos autónomos*, 2023.

[46] ISO, «ISO 26262-1:2018,» 2018. [En línea]. Available: https://www.iso.org/obp/ui/en/#iso:std:iso:26262:-1:ed-2:v1:en. [Último acceso: Noviembre 2024].

[47] M. Ilas y I. Constantin, «Towards real-time and real-life image classification and detection using CNN: a review of practical applications requirements, algorithms, hardware and current trends,» *IEEE*, 2020.

[48] K. Ulrich, S. Epinger y M. Yang, Product Design and Development 7th ed., McGraw-Hill Education, 2020.

[49] L. Prieto, Interviewee, *Insumos necesarios para realizar el proyecto.* [Entrevista]. Agosto 2024.

[50] AMD, «Heterogeneous Accelerated Compute Cluster (HACC) Program,» 2024. [En línea]. Available: https://www.amd.com/en/corporate/university-program/aup-hacc.html.

[51] L. G. L. Vega, Interviewee, *Entrevista para plantear necesidades del cliente.* [Entrevista]. Agosto 2024.

[52] T. Said, «vit.cpp,» 2024. [En línea]. Available: https://github.com/staghado/vit.cpp. [Último acceso: 3 Setiembre 2024].

[53] Stanford Vision Lab, «Imagenet,» 2020. [En línea]. Available: https://www.image-net.org/download.php. [Último acceso: 2024].

[54] R. Walpole, R. Myers y S. Myers, Probabilidad y estadística para Ingenieros, Pearson , 2012.

[55] L. G. L. Vega, *A User-Friendly Ecosystem for AI FPGA-based*, 2024.

[56] L. Leon Vega y L. Prieto-Sibaja, «LLM Acceleration on FPGAs: A Comparative Study of Layer and Spatial Accelerators,» 2024.

Apéndices

Todos los Apéndices referenciados en este documento están disponibles mediante el siguiente link: https://drive.google.com/drive/folders/1V0Ol-_f586n9Y6C8ZVXeIqqbZ_wqKEJS?usp=sharing

Anexos

A. Implementación en CPU del ViT de Staghado: https://github.com/staghado/vit.cpp