

Instituto Tecnológico de Costa Rica

Carrera de Ingeniería Mecatrónica

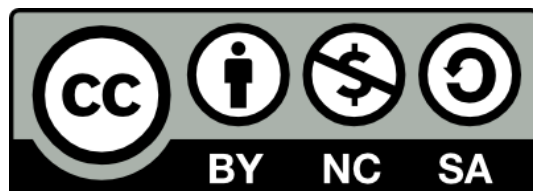


**Diseño de unidad de procesamiento configurable en FPGA para aceleración de convolución
en aplicación de visión por computador**

**Informe de Proyecto de Graduación para optar por el título de Ingeniero en Mecatrónica
con el grado académico de Licenciatura**

Samuel Daniel Castro Villalobos

Cartago, marzo de 2024



Este trabajo titulado Diseño de unidad de procesamiento configurable en FPGA para aceleración de convolución en aplicación de visión por computador por Samuel Daniel Castro Villalobos, se encuentra bajo la Licencia Creative Commons [Atribución-ShareAlike 4.0 International](https://creativecommons.org/licenses/by-nc-sa/4.0/).

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.

A handwritten signature in black ink, appearing to read 'Samuel Daniel Castro Villalobos', with a stylized, cursive script.

Cartago, Costa Rica
19 de marzo de 2024

Firma del autor

Samuel Daniel Castro Villalobos

Céd: 117890016


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: Samuel Daniel Castro Villalobos

Proyecto: Diseño de unidad de procesamiento configurable en FPGA para aceleración de convolución en aplicación de visión por computador

Miembros del jurado evaluador



Ing. Róger Meléndez Poltronieri
Jurado



Ing. Paula Monge Chanto
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, Costa Rica

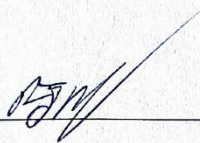
13 de marzo de 2024

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

El Profesor Asesor del presente trabajo final de graduación, indica que el documento presentado por el estudiante cumple con las normas establecidas por la Carrera de Ingeniería Mecatrónica, como requisito para optar por el título de Ingeniero en Mecatrónica, con el grado académico de Licenciatura, del Instituto Tecnológico de Costa Rica.

Estudiante: Samuel Daniel Castro Villalobos

Proyecto: Diseño de unidad de procesamiento configurable en FPGA para aceleración de convolución en aplicación de visión por computador



Ing. Rodolfo Piedra Camacho

Asesor

Cartago, Costa Rica

13 de marzo de 2024

Resumen:

El presente documento contempla el informe del proyecto final de graduación para optar por el título de Ingeniería Mecatrónica en el Instituto Tecnológico de Costa Rica. El desarrollo del proyecto se centró en la implementación de una unidad de procesamiento para el algoritmo de convolución, con el propósito de utilizarlo en aplicaciones de visión por computadora para clasificación.

El sistema diseñado se trata de un acelerador de hardware capaz de ejecutar la operación de convolución por medio de una técnica de convolución separable para su uso en funciones de aprendizaje profundo. El sistema fue diseñado por medio de Vivado HLS 2018.2 para su uso en FPGAs de bajo perfil. Se desarrolló con una arquitectura flexible al estar integrado a la estructura del framework de Flexible Accelerators Library del MHPC Luis León Vega y el ECASLab del Tecnológico de Costa Rica. Este se evaluó por medio de un estudio de *Design Space Exploration* que evaluó su consumo de recursos, de latencia y de calidad operativa en el procesamiento de imágenes en comparación con aproximaciones basadas enteramente en software.

Además, este se integró a una red neuronal basada en el modelo de MobileNetV2 en donde se validó su aplicabilidad en escenarios de clasificación de imágenes propios de la visión por computador donde se obtuvieron resultados que demostraron su versatilidad y óptima aplicación en este contexto.

Palabras Clave: FPGA, Red Neuronal, Convolución, MobileNet, HLS, C++, Visión por Computadora

Abstract:

This document encompasses the final graduation project report to qualify for the title of Mechatronic Engineering at the Costa Rica Institute of Technology. The project's development focused on implementing a processing unit for the convolution algorithm, aiming to utilize it in computer vision applications for classification.

The designed system is a hardware accelerator capable of performing the convolution operation using a separable convolution technique for deep learning functions. The system was designed using Vivado HLS 2018.2 for low-profile FPGAs. It was developed with a flexible architecture integrated into the structure of the Flexible Accelerators Library framework of MHPC Luis León Vega and the ECASLab of the Costa Rica Institute of Technology. It was evaluated through a Design Space Exploration study that assessed its resource consumption, latency, and operational quality in image processing compared to entirely software-based approaches.

Furthermore, it was integrated into a neural network based on the MobileNetV2 model, where its applicability in computer vision image classification scenarios was validated. Results were obtained that demonstrated its versatility and optimal application in this context.

Keywords: FPGA, Neural Network, Convolution, MobileNet, HLS, C++, Computer Vision

A mi familia, que se merece todo y más

Tabla de Contenido

Lista de Figuras	V
Lista de Tablas.....	VIII
Lista de Abreviaciones	X
1. Introducción	1
1.1 Contexto	1
1.2 Descripción del Problema	3
1.3 Síntesis del Problema	4
1.4 Objetivos	4
1.4.1 Objetivo General.....	4
1.4.2 Objetivos Específicos	4
1.5 Estructura del documento	5
2. Marco Teórico	6
2.1 Redes Neuronales Convolucionales	6
2.1.1 Capa de convolución	7
2.1.2 Padding	9
2.1.3 Stride.....	9
2.1.4 Capa de pooling	10
2.1.5 Función de activación	10
2.1.6 Batch Normalization.....	11
2.1.7 Capa Densa	12
2.1.8 Aproximaciones novedosas en convolución	13
2.1.9 Consumo de recursos y número de operaciones.....	18
2.1.10 Consumo de recursos y operaciones de red densa.....	22

2.2	FPGA.....	22
2.2.1	Recursos en la FPGA	23
2.2.2	High-Level Synthesis	23
2.2.3	Interfaces y Frameworks de Integración	29
2.2.4	Aceleradores	30
2.3	Flexible Accelerators Library	30
2.3.1	Approximate Computing (AxC)	31
2.3.2	AxC Executer.....	32
2.3.3	Frameworks populares para DL.....	32
2.4	Estado del Arte.....	34
2.4.1	Modelos en la academia.....	34
2.4.2	Métodos de convolución optimizados	35
2.4.3	MobileNet	36
3.	Metodología	44
3.1	Planeación	44
3.2	Identificación de necesidades	44
3.3	Estado del arte.....	45
3.4	Selección de Modelos	46
3.5	Establecimiento de Especificaciones	47
3.6	Generación de conceptos	48
3.7	Desarrollo final	49
3.7.1	Gestión de Costo del Proyecto	50
4.	Desarrollo de la solución.....	51
4.1	Interpretación de las necesidades	51
4.1.1	Requisitos mínimos de hardware y software.....	53

4.1.2	Estudio de los requisitos para integración en FAL	54
4.2	Definición de Especificaciones	57
4.3	Selección de conceptos	58
4.4	Codificación del algoritmo de convolución separable en software.	60
4.4.1	Prototipo de algoritmo de convolución en C++	61
4.4.2	Prototipo de red neuronal convolucional en Python	63
4.5	Planteamiento y Diseño del Modelo a nivel modular	66
4.6	Diseño en HLS	72
4.6.1	Codificación base	73
4.6.2	Implementación de aproximaciones.....	76
4.6.3	Funcionalidades añadidas	77
4.6.4	Directivas de Optimización	82
4.7	Testbench y Reporte de Vivado HLS.....	84
4.7.1	Design Space Exploration.....	86
4.8	AxC Executer y Simulación de red en FPGA.....	91
5.	Resultados y Análisis	93
5.1	Pruebas de consumo recursos.....	93
5.1.1	Consumo de recursos en Convolución Depthwise.....	93
5.1.2	Consumo de recursos en Convolución Pointwise	95
5.1.3	Consumo de recursos en aproximaciones anteriores	96
5.2	Pruebas de Latencia	98
5.2.1	Latencia en Convolución Depthwise	98
5.2.2	Latencia en Convolución Pointwise	99
5.2.3	Latencia en aproximaciones anteriores	99
5.3	Pruebas de sobre la calidad de las imágenes obtenidas	101

5.3.1	Calidad de imagen obtenida en Convolución Depthwise	101
5.3.2	Calidad de imagen obtenida en Convolución Pointwise	105
5.4	Pruebas de los aceleradores en FPGA	109
5.5	Pruebas de la red en simulación	111
5.6	Requerimientos establecidos vs requerimientos cumplidos.....	114
5.7	Análisis Financiero	119
5.7.1	Caso 1: Costo del proyecto de ejecutarse desde cero	119
5.7.2	Caso 2: Costo del proyecto implementando el sistema desarrollado 121	
5.7.3	Caso 3: Costo del proyecto ejecutado utilizando alternativas en el mercado.....	123
5.7.4	Consideraciones del estudio económico.....	123
5.8	Impacto Ambiental.....	124
5.8.1	Consumo energético.....	124
6.	Conclusiones	126
6.1	Recomendaciones	129
	Bibliografía	130
	Anexos.....	138
	Anexo 1. Algoritmos.....	138
	Anexo 2. Figuras	139
	Anexo 3. Repositorios y códigos de consulta.....	139

Lista de Figuras

Figura 2.1. Estructura Básica de una red neuronal convolucional.	7
Figura 2.2. Estructura Básica de la capa convolucional.	8
Figura 2.3. Aplicación de un kernel (cubo celeste) de 3x3x3 a un volumen (imagen) de 10x10x3.	8
Figura 2.4. Zero-padding a una imagen (matriz)	9
Figura 2.5. Stride de 2 a una imagen.	10
Figura 2.6. Pooling a un conjunto de imágenes.....	10
Figura 2.7. Función ReLU vs ReLU6.....	11
Figura 2.8. Red Neuronal Densa.....	13
Figura 2.9. Aplicación de un kernel (en verde) de 3x3x1 al canal verde de una imagen (10x10).	14
Figura 2.10. Aplicación de un kernel por canal a una imagen (10x10).....	14
Figura 2.11. Aplicación de un kernel de 1x1 a una imagen.	15
Figura 2.12. Kernels de una convolución separable en profundidad y una convolución estándar.	15
Figura 2.13. Diferentes tipos de convolución estándar	16
Figura 2.14. Diagrama del Algoritmo de Winograd	18
Figura 2.15. Niveles de abstracción del diseño digital.	25
Figura 2.16. Lógica de una arquitectura en serie.	28
Figura 2.17. Lógica de una arquitectura pipeline.	28
Figura 2.18. Operación de line buffer en convolución en 2D.	36
Figura 2.19. Capas de la convolución estándar y la MobileNet aplicando batch-normalization y ReLU. [7].....	38
Figura 2.20. Estructura de convolución para la V1 y V2 del modelo MobileNet	39
Figura 2.21. Estructura básica de MobileNetV2 y MobileNetV3	40
Figura 2.22. Función de activación “sigmoid-linear units” (SiLU).....	41
Figura 2.23. Función de activación “hard swish” (h-swish).....	41
Figura 2.24. Comparación entre la última etapa de la arquitectura de la MobileNetV2 y la MobileNetV3.....	42

Figura 4.1. Diagrama de flujo de algoritmo de convolución separable en C++	62
Figura 4.2. Comparación de resultados entre procesamiento de imagen mediante algoritmo personalizado vs librerías estándar.....	63
Figura 4.3. Estructura base del prototipo de Python de la Red Neuronal Convolutiva.	64
Figura 4.4. Predicciones realizadas para el modelo de la MobileNet con el set de datos de Intel Image Classification.....	66
Figura 4.5. Diseño modular del acelerador de convolución.	68
Figura 4.6. Ejecución de Line Buffers para caso de acelerador de convolución depthwise.....	74
Figura 4.7. Ejecución de Line Buffers para caso de acelerador de convolución pointwise.....	75
Figura 4.8. Padding a kernel para uso general en convolución depthwise.	78
Figura 4.9. Padding a kernel para uso general en convolución pointwise.	79
Figura 4.10. Convolución pointwise realizada sobre capas de +16 canales.	79
Figura 4.11. El diseño por dataflow involucra añadir interfaces FIFO para comunicar cada módulo.	82
Figura 4.12. Ambiente de análisis para evaluación del flujo del sistema	82
Figura 4.13. Cambio en el flujo de operación a partir de directiva de array partition	83
Figura 4.14. Estructura en capas de la red MobileNetV2. Denotado en rojo las capas de convolución.	92
Figura 5.1. Latencia por unidad de hardware testada durante el DSE para convolución depthwise.....	98
Figura 5.2. Latencia por unidad de hardware testada durante el DSE para convolución pointwise.	99
Figura 5.3. Resultados de la convolución depthwise según el tamaño de dato para un tamaño de parte entera de 2.	102
Figura 5.4. Resultados de la convolución pointwise en distintos tamaños de dato para un tamaño de parte entera de 2.	106
Figura 5.5. Configuración de la FPGA para el procesamiento de imágenes.....	109

Figura 5.6. Consumo energético de la FPGA en operación de convolución de una imagen. 111

Figura 5.7. Histograma de actividad de la magnitud de los datos presentes en la red. 112

Lista de Tablas

Tabla 2.1. Directivas de diseño en HLS 2018.2. [12]	27
Tabla 3.1. Interpretación de Necesidades	45
Tabla 3.2. Filtrado de los conceptos que se tomaron en cuenta.	47
Tabla 3.3. Métricas definidas para cumplir con las necesidades encontradas.	48
Tabla 3.4. Definición de las características del concepto.....	49
Tabla 4.1. Necesidades interpretadas para la realización del acelerador de convolución.....	52
Tabla 4.2. Métricas de evaluación para conceptos.	58
Tabla 4.3. Algoritmos de convolución evaluados para su desarrollo en FAL	59
Tabla 4.4. Modelos de convolución que emplean convolución separable evaluados para su desarrollo en software y hardware.....	65
Tabla 4.5. Consideraciones base para el desarrollo de la arquitectura.	67
Tabla 4.6. Consideraciones adicionales para el desarrollo de la arquitectura.	69
Tabla 4.7. Algoritmos de convolución evaluados para su desarrollo en FAL	70
Tabla 4.8. Límites operativos del acelerador depthwise	81
Tabla 4.9. Límites operativos del acelerador pointwise.....	81
Tabla 4.10. Tabla Resumen para Optimizaciones.....	83
Tabla 4.11. Tabla de consumo de recursos y latencia para optimizaciones propuestas en convolución depthwise.....	85
Tabla 4.12 Tabla de consumo de recursos y latencia para optimizaciones propuestas en convolución pointwise	86
Tabla 5.1. Consumo del porcentaje total de recursos en FPGA Artix7 para acelerador de convolución depthwise.	94
Tabla 5.2. Consumo del porcentaje total de recursos en FPGA Artix7 para acelerador de convolución pointwise.	96
Tabla 5.3. Consumo del porcentaje total de recursos en FPGA Artix7 para acelerador de convolución por stream.	97
Tabla 5.4. Latencia por unidad de hardware testada durante el DSE para convolución por stream.	100

Tabla 5.5. Latencia en ciclos por píxel para los aceleradores presentes en FAL.	101
Tabla 5.6. Error cuadrático medio en convolución depthwise para cada unidad según el tamaño de dato.	103
Tabla 5.7. Peak Signal to Noise Ratio para convolución depthwise	104
Tabla 5.8. Índice de Similitud Estructural para convolución depthwise	105
Tabla 5.9. Error cuadrático medio en convolución pointwise para cada unidad según el tamaño de dato.	107
Tabla 5.10. Peak Signal to Noise Ratio para convolución pointwise	107
Tabla 5.11. Índice de Similitud Estructural para convolución pointwise.....	108
Tabla 5.12. Latencia en ms para los distintos tamaños de imagen.	109
Tabla 5.13. Consumo de Recursos para distintas	110
Tabla 5.14. Resultados de pruebas realizadas para determinar la combinación óptima del tamaño de dato.	113
Tabla 5.15. Valor obtenido vs el valor meta para las métricas de los aceleradores.	115
Tabla 5.16. Costos del Sistema para Escenario Base	121
Tabla 5.17. Costos del Sistema para Escenario con existencia de implementación en FAL.	122

Lista de Abreviaciones

I.	CNN	<i>Convolutional Neural Network</i>
II.	HLS	<i>High Level Synthesis</i>
III.	FPGA	<i>Field Programmable Gate Array</i>
IV.	CNN	<i>Convolutional Neural Networks</i>
V.	GPU	<i>Graphics Processing Unit</i>
VI.	TPU	<i>Tensor Processing Unit</i>
VII.	DL	<i>Deep Learning</i>
VIII.	LUT	<i>Look-up Table</i>
IX.	FF	<i>Flip Flop</i>
X.	DSP	<i>Digital Signal Processor</i>
XI.	BRAM	<i>Block Random Access Memory</i>
XII.	CI	<i>Circuito Integrado</i>
XIII.	HDL	<i>Hardware Description Language</i>
XIV.	SP	<i>Sistema de Procesamiento</i>
XV.	DSE	<i>Design Space Exploration</i>
XVI.	AxC	<i>Approximate Computing</i>
XVII.	HVS	<i>Human Visual System</i>

1. Introducción

1.1 Contexto

Las redes neuronales convolucionales (CNN por sus siglas in inglés) son uno de los modelos computacionales de mayor impacto en los últimos años. Estos lograron optimizar el procesamiento de imágenes a tal punto que no se puede hablar de visión por computadora sin hablar de estos algoritmos de Deep Learning. Para tareas de clasificación [1] [2], detección de objetos [1] [3], segmentación semántica [3] y extracción de imágenes [4], han mostrado ser una buena opción para abordar problemas en el área.

Uno de los campos de interés resulta en la oportunidad de poder utilizar estos modelos y algoritmos en contextos con mayor portabilidad y en aplicaciones a tiempo real. Sin embargo, estos modelos están asociados a un alto consumo energético, de recursos y limitado en muchas ocasiones a la necesidad de internet. La necesidad de ejecutar una gran cantidad de operaciones es paradójico a la necesidad de ejecutarse en un tiempo y espacio reducido. A medida que crece la necesidad de estos algoritmos en el mercado, muchos optan por llevarlos a arquitecturas, tales como GPU's o TPU's [5] que se diseñan para soportar mayores exigencias. Otros utilizan estrategias para simplificar estos modelos y utilizar opciones más simples; simplificaciones tales como pooling, pruning [6] o variaciones de algoritmos de CNNs tradicionales como la MobileNet [7] [8], ShuffleNet [9], CondenseNet [10] desarrolladas para optimizar el uso de recursos.

Una opción para acelerar y optimizar los modelos de Deep Learning es el uso de FPGAs. A pesar de que el mercado en este momento es dominado por las GPUs debido a su menor costo de ingeniería, las FPGAs tienen características que las hacen sumamente competitivas. Principalmente, estas son una opción con mucha flexibilidad, que permiten a ingenieros reconfigurar el hardware a niveles de bits. Otra característica es su baja latencia [11] en comparación con las GPUs ya que estas pueden alcanzar la escala de los nanosegundos mientras que las GPU suelen

rondar el rango de los microsegundos. Por último, el consumo energético es ampliamente inferior al de una GPU.

El ECASLab es un laboratorio de investigación en el Instituto Tecnológico de Costa Rica que ha venido trabajando una serie de proyectos alrededor de inteligencia artificial y los algoritmos utilizados en este campo. Estos con el fin de desarrollar arquitecturas de hardware en FPGA's de bajo perfil, con un bajo consumo energético y de recursos, con la capacidad de acelerar dichos algoritmos propios del aprendizaje de máquina. Estas arquitecturas están diseñadas para ser implementadas, en problemas específicos conexos con la inteligencia artificial y la visión por computadora asistida por IA.

El MHPC. Luis G. León Vega es un investigador que actualmente está realizando su doctorado en la Universidad de Trieste, Italia. Este colabora activamente con el ECASLab y se ha especializado en problemas de este tipo, especialmente en aquellos dentro del campo de la computación aproximada, el *Edge computing* y la aceleración de hardware. En estas áreas se ha centrado su desarrollo profesional durante los últimos años. En su tesis de maestría, se enfocó en el estudio de la aceleración flexible de algoritmos basados en redes neuronales profundas (DNN) en entornos de *Edge computing* mediante el uso de FPGA's y técnicas de computación aproximada. Para su doctorado decidió continuar con el trabajo que tenía y buscar generar nuevas alternativas para hacer uso de los sistemas que ha ido generando.

El ECASLab y el trabajo de maestría de Luis León, han trabajado en un repositorio conformado y enfocado en la creación de una biblioteca de aceleradores genéricos de algoritmos basados en DNN's para FPGA's de bajo perfil. Dicha biblioteca se denomina FAL ("Flexible Accelerator Library"), cuenta con varias unidades de procesamiento codificadas y capaces de realizar operaciones con el objetivo de explotar el bajo consumo de recursos que brindan las distintas FPGA's [12]. Entre las operaciones que realiza, se puede destacar la suma, multiplicación y convolución nativa de matrices.

1.2 Descripción del Problema

El procesamiento de imágenes y señales es un área crítica para muchas aplicaciones en la actualidad. Una de las operaciones fundamentales en el procesamiento de imágenes es la convolución, que se utiliza en una amplia variedad de aplicaciones de visión por computador, como la clasificación o detección de objetos en una imagen. Sin embargo, la convolución es una operación computacionalmente costosa y puede requerir mucho tiempo de procesamiento, lo que limita su uso en aplicaciones de naturaleza crítica (ubicaciones remotas, disponibilidad de internet o seguridad).

Este proyecto aborda el diseño y desarrollo de aceleradores de hardware de alta velocidad para la operación de convolución a partir del uso de una técnica de convolución más eficiente en el consumo de recursos y energía. Según [13], estos aceleradores de hardware deben ser capaces de procesar datos en tiempo real, reduciendo significativamente el tiempo de procesamiento y mejorando el rendimiento del sistema en general. Además, se espera que la arquitectura desarrollada sea de bajo costo y fácil de integrar en sistemas de procesamiento de imágenes y señales existentes. Con la solución de este problema se espera mejorar el rendimiento de los sistemas de procesamiento de imágenes y señales en tiempo real, lo que puede tener un impacto significativo en una amplia variedad de aplicaciones, desde la medicina hasta la industria.

1.3 Síntesis del Problema

Inexistencia de una unidad de procesamiento que sea capaz de realizar la operación de convolución de forma eficiente, y que adicionalmente tenga compatibilidad con las implementaciones de la librería Flexible Accelerators Library (FAL) para el procesamiento de imágenes.

1.4 Objetivos

1.4.1 Objetivo General

- Diseñar una unidad de procesamiento eficiente en el consumo de energía y recursos capaz de realizar la operación de convolución para el análisis proveniente en aplicaciones de visión por computador.

1.4.2 Objetivos Específicos

- Analizar los requerimientos y limitaciones para la implementación de la unidad de procesamiento del modelo de convolución dentro de la librería FAL.
- Desarrollar un modelo de convolución en software para evaluación y comparación de resultados con la unidad de procesamiento de hardware.
- Describir una unidad de hardware utilizando High Level Synthesis en FPGA para la operación de convolución, que sea integrable y aplicable dentro de una red neuronal para la clasificación de objetos.
- Integrar la unidad de procesamiento de convolución dentro de una red neuronal en una simulación por medio del Approximate Computing Executer de la librería FAL.
- Validar el rendimiento y viabilidad de la unidad de procesamiento en una aplicación de clasificación de visión por computador, garantizando su correcto funcionamiento y cumpliendo con los requerimientos específicos de la aplicación.

1.5 Estructura del documento

A continuación, se resume el contenido de los siguientes capítulos en el presente informe.

En el Capítulo 2, correspondiente al Marco Teórico, se presentarán los conceptos fundamentales relacionados con el proyecto, para una comprensión íntegra de los temas y campos que este comprende. Se profundizará en el estado del arte, los algoritmos relevantes y las definiciones esenciales.

La Metodología se abordará en el Capítulo 3. En esta se discute cómo se desarrolló el proyecto, así como los pasos y procedimientos utilizados en el desarrollo de éste. En este caso se consideró la metodología “Ulrich Eppinger” que se basa en el proceso de diseño de ingeniería.

El Capítulo 4 abarca la propuesta de diseño, en donde se evidencia como se llevó a cabo el desarrollo de la metodología propuesta y como se diseñó e implementó la solución. Todos los estudios, implementaciones y elecciones que se hicieron para llegar al resultado se justifican en esta sección.

Los resultados y su análisis se presenta en el siguiente capítulo. En este se verificó y validó que la solución cumpla con los criterios planteados inicialmente. En el caso del presente proyecto, el sistema se sometió a pruebas que evaluaron su consumo de recursos, latencia y calidad de los resultados que se obtenían en la operación para determinar que este fuera de calidad respecto a aproximaciones anteriores en la librería. Además, se incluye el análisis en los distintos contextos envueltos dentro del proyecto.

El Capítulo 6 se compone de dos secciones, conclusiones y recomendaciones. En este se evalúa el cumplimiento del proyecto y se discute el camino para futuros desarrollos alrededor del tema.

Las últimas 2 secciones corresponden a las referencias bibliográficas y los anexos en ese orden.

2. Marco Teórico

2.1 Redes Neuronales Convolucionales

Las redes neuronales convolucionales (CNN) son un modelo de Deep Learning que ha desempeñado un papel importante en el campo de la visión por computadora, enfocadas en la clasificación de imágenes, la detección de objetivos y la segmentación semántica en escenas complejas. Las CNN se utilizan ampliamente en seguridad, conducción autónoma, reconocimiento de objetos y otras industrias [14].

Para explicar la forma en que estos algoritmos funcionan dentro del procesamiento de imágenes, es necesario verlo a partir de dos fases, el entrenamiento y la inferencia. Como algoritmo de aprendizaje supervisado, las CNN utilizan un conjunto de imágenes etiquetadas para entrenar. El proceso de entrenamiento implementa el algoritmo de retropropagación que actualiza los parámetros de la CNN. Después de que el modelo ha sido afinado y entrenado correctamente, el modelo aprendido es capaz de clasificar nuevas muestras. Esto se conoce como inferencia. La estructura y los parámetros de una red neuronal quedan fijados una vez finalizado el proceso de entrenamiento, mientras que la inferencia se lleva a cabo cada vez que se presenta un nuevo dato de muestra. Por lo tanto, esta es la fase que interesa acelerar [15].

Las CNN están diseñadas para aprender automáticamente, y de manera adaptativa, jerarquías espaciales de características durante el entrenamiento [16]. De esta forma una imagen puede ser clasificada cuando las características apuntan a la clase más probable a la que pertenece la imagen [15].

Hay distintas aproximaciones para realizar estos modelos. La forma en que se llevan a cabo las operaciones se puede mejorar con mejores algoritmos sofisticados que reduzcan el tiempo de ejecución o el consumo de recursos. De esta forma redes más novedosas surgen para jugar entre la precisión, el tiempo de ejecución, el consumo energético, de recursos y la complejidad. Algunas de ellas son la MobileNet [7] [8], ShuffleNet [9], CondenseNet [10] las cuales se abarcarán con

detalle más adelante. Lo más importante a destacar es el uso de algoritmos más eficientes, que se explican en las secciones posteriores.

Las redes neuronales convolucionales están organizadas en capas. Dentro de una red convolucional estándar, se espera a la entrada una imagen y a la salida un valor esperado de lo que puede representar esa imagen. Para poder realizar ese procesamiento existen una serie de capas internas que se encargan de operaciones como convoluciones, multiplicaciones de matrices, funciones de activación, capas de pooling y/o capas densas que se encargan de lograr el concepto mencionado. Una idea general de este proceso se presenta en la Figura 2.1.

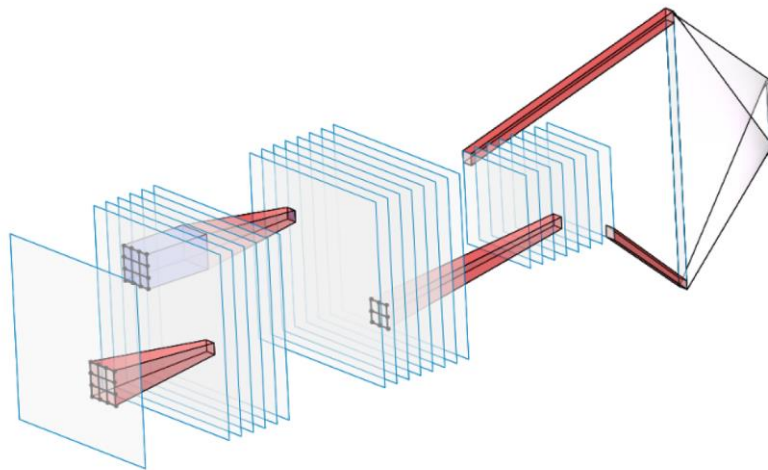


Figura 2.1. Estructura Básica de una red neuronal convolucional.

En teoría cada capa se encarga de realizar una operación distinta por lo que, bajo esta idea general, se procede a describir la forma en la que cada capa opera.

2.1.1 Capa de convolución

La capa de convolución consiste en el proceso de aplicar un 'kernel' (también conocido como filtro) sobre un plano o volumen, donde se realiza una multiplicación elemento por elemento entre los valores del kernel y los valores de los píxeles correspondientes en esa posición. Luego, se suman todos los resultados de estas multiplicaciones para obtener un único valor que representa la contribución de esa posición en la imagen resultante [17]. Este filtro se va desplazando por todos los

pixeles presentes en la imagen y arrojan como resultado un llamado mapa de características.

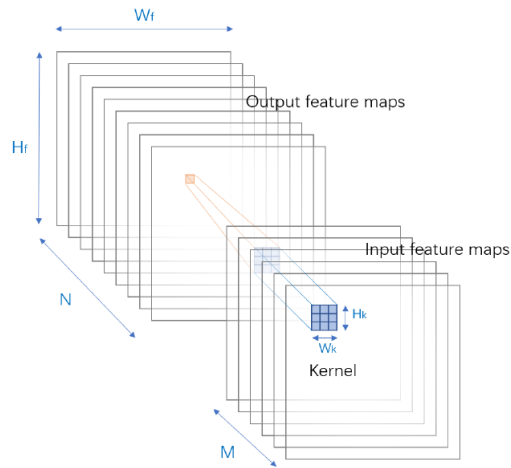


Figura 2.2. Estructura Básica de la capa convolucional.

Como se presenta en la Figura 2.2, a la imagen con M canales se le aplica un kernel de dimensión $W_k \times H_k \times 1$, que da como resultado el valor de un espacio en la siguiente capa. Este kernel es plano, sin embargo, también se puede presentar un kernel con un volumen como el ejemplo en la Figura 2.3. Esta convolución ejecutada de esta forma se le conoce como convolución espacial y es la más común y básica.

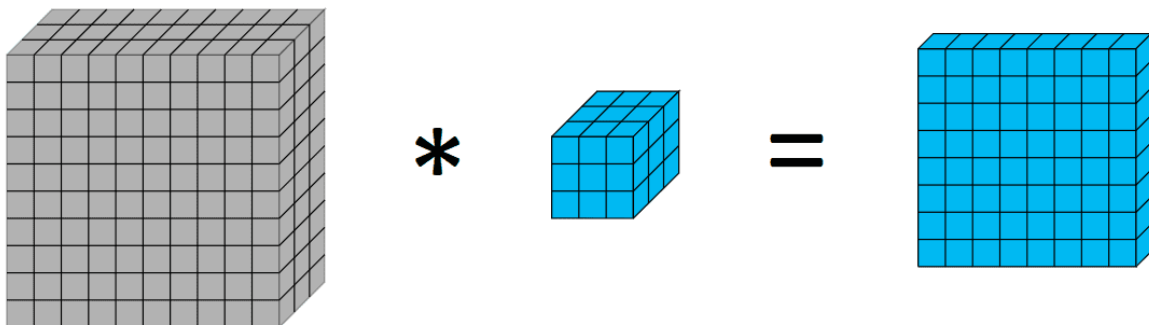


Figura 2.3. Aplicación de un kernel (cubo celeste) de $3 \times 3 \times 3$ a un volumen (imagen) de $10 \times 10 \times 3$.

Dependiendo de la aplicación y la necesidad se puede utilizar un caso o el otro. Se debe tomar en cuenta que se puede utilizar varios filtros, lo que da como resultado una capa de N canales, como se aprecia en la Figura 2.2.

La convolución se puede ejecutar de distintas formas siempre con la idea de extraer las características importantes de una imagen. A través de los algoritmos de Deep Learning se pretende lograr interpretar la información de la mejor forma para obtener los mejores resultados. Distintos tipos de convolución, aproximaciones novedosas y conceptos teóricos de las mismas para su investigación se presentan más adelante.

2.1.2 Padding

Como se puede apreciar en la Figura 2.3, el resultado de la convolución (también conocido como mapa de características) es de un tamaño menor al de la imagen de entrada. La imagen tiene una altura y un ancho de 10 x 10 pero el mapa de características tiene dimensiones de 8 x 8. El proceso de convolución tiende a encoger el resultado si no se aplica un procedimiento para evitarlo. Este procedimiento se conoce como padding y consiste en añadir valores a los bordes de la imagen para aumentar la altura y el ancho y que cuando se aplique el filtro el resultado quede del mismo tamaño que se planteaba. El padding más común es el *zero-padding* que consiste en añadir ceros alrededor, como se presenta en la Figura 2.4.

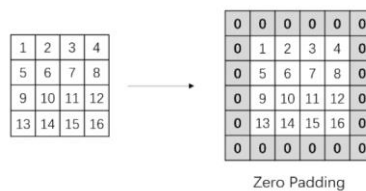


Figura 2.4. Zero-padding a una imagen (matriz)

2.1.3 Stride

A la hora de utilizar un filtro en una imagen, este se aplica varias veces. Empezando en algún punto de la imagen, el filtro se irá desplazando para ir realizando la operación de convolución y obtener el mapa de características final. Este desplazamiento se va realizando según un valor, conocido como stride. En la Figura 2.5, se muestra un stride de 2 en donde aplicando una convolución como la previamente explicada, se obtiene un mapa de características de 3 x 3.

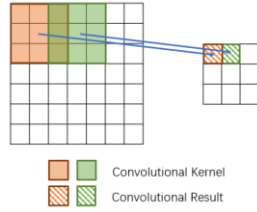


Figura 2.5. Stride de 2 a una imagen.

2.1.4 Capa de pooling

La capa de pooling cumple el objetivo de reducir espacialmente la imagen o el mapa que se tiene para reducir el número de operaciones y el procesamiento que se debe realizar. Entre más pequeñas sean las matrices de las imágenes menos operaciones se deben realizar. Para esto, se deben definir nuevamente una serie de parámetros tales como los introducidos anteriormente; conceptos como el stride, el tamaño de ventana o el método de pooling. Este último es importante pues existen varios tipos de pooling con los que se pueden obtener distintos resultados según el caso, dos muy comunes son el *max-pooling* y el *average-pooling*. El *max-pooling* toma de la ventana escogida el valor con mayor tamaño mientras que el *average-pooling* calcula el promedio de todos los valores evaluados en la ventana.

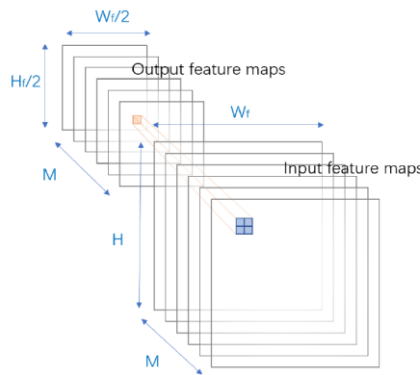


Figura 2.6. Pooling a un conjunto de imágenes

2.1.5 Función de activación

Al mapa de características o la imagen se le aplica una función que introduce no-linealidad al sistema. Este resultado se envía a la siguiente capa como una entrada. De esta forma el modelo trabaja en un mayor orden y es capaz de realizar tareas

más complejas. Dentro de la convolución la función de activación más común es la *Rectified Linear Units* (ReLU), ecuación que se presenta a continuación.

$$ReLU(x) = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases} \quad [1]$$

Como parte de algunos aportes novedosos utilizados en modelos como MobileNet, se desarrolla una función de activación basada en la ReLU, conocida como ReLU6. Esta será importante más adelante cuando se introduzcan algunos conceptos.

$$ReLU6(x) = \min(\max(0, x), 6) \quad [2]$$

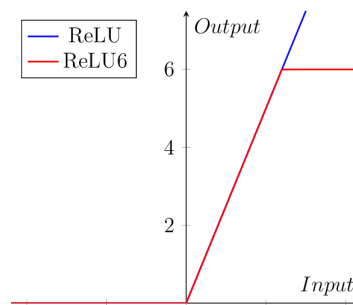


Figura 2.7. Función ReLU vs ReLU6

2.1.6 Batch Normalization

Batch Normalization es una técnica utilizada en el aprendizaje profundo para normalizar las activaciones de una capa de una red neuronal. Esto ayuda a mejorar el entrenamiento de la red y su capacidad para generalizar.

Cuando se entrena una red neuronal profunda, las distribuciones de las activaciones en cada capa pueden cambiar a medida que se actualizan los pesos. Esto puede hacer que el entrenamiento sea más lento y difícil. Batch Normalization soluciona este problema normalizando las activaciones, es decir, ajustando sus valores para que tengan una media y una varianza fijas.

Al normalizar las activaciones, Batch Normalization hace que el entrenamiento sea más estable y permite que la red aprenda más rápido. También actúa como un regularizador, lo que significa que ayuda a prevenir el sobreajuste y mejora la capacidad de generalización de la red. [18] El procedimiento para calcular el batch normalization se presenta a continuación.

$$\mu = \frac{1}{m} \sum_{i=1}^m x_i$$

$$\sigma^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu)^2$$

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

$$y_i = \gamma \hat{x}_i + \beta \quad [3]$$

2.1.7 Capa Densa

Esta capa no es propiamente parte de la red neuronal convolucional, sino más bien un modelo independiente que apoya a la red convolucional. Para que ambos modelos trabajen como un único y robusto sistema, en la última capa, el mapa de características se envuelve en un proceso conocido como *flattening* en donde se lleva a una estructura que una red neuronal pueda manejar. En la Figura 2.1 se da una idea de este concepto, en donde se lleva una serie de mapas de características a una única dimensión.

Para una comprensión mayor de redes neuronales se recomienda revisar literatura relacionada, sin embargo, se define el funcionamiento de una red neuronal artificial según la interconexión de neuronas artificiales y el procesamiento de información a través de capas. Cada neurona combina linealmente las entradas ponderadas por los pesos de cada conexión y aplica una función de activación no lineal. Estas operaciones se repiten en cada capa, permitiendo la propagación de información y el aprendizaje mediante el ajuste de los pesos con algoritmos de optimización como el descenso del gradiente [19].

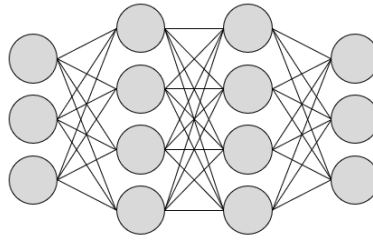


Figura 2.8. Red Neuronal Densa.

2.1.8 Aproximaciones novedosas en convolución

Uno de los principales retos del Deep Learning es su alto consumo de recursos. Si bien todo el proceso es altamente costoso, la convolución puede llegar a ser la etapa más pesada. Para cualquier flujo de operaciones, se puede considerar la frase "La cadena es tan fuerte como su eslabón más débil". La optimización de esta etapa es un área de estudio muy amplio y que, alineado con los objetivos del proyecto, resulta imperativo revisar los avances ya propuestos.

Distintas aproximaciones se han realizado para hacer más eficiente esta etapa, desde el uso de algoritmos eficientes de convolución [20], técnicas de compresión de modelos como la poda de pesos y cuantización del modelo [21] o el uso de capas convolucionales dilatadas [22]. Para el presente trabajo el enfoque recaerá en la búsqueda de algoritmos eficientes de convolución. A continuación, se presentan varios enfoques y técnicas utilizadas.

2.1.8.1 Convolución Separable en Profundidad

La convolución separable en profundidad fue introducida en [23] pero popularizada por los modelos de MobileNet desarrollados por Google [7] [8] [24]. Como tal, consiste en un modelo capaz de desarrollar el proceso de convolución con una menor cantidad de recursos. Esto es gracias a que se divide la convolución estándar en dos pasos, una convolución en profundidad y una convolución puntual.

2.1.8.1.1 Convolución en Profundidad (*Depthwise*)

Primeramente, la convolución en profundidad consiste en aplicar el proceso de convolución a lo largo de cada dimensión (canal) en la imagen, como se observa en la Figura 2.9 y Figura 2.10.

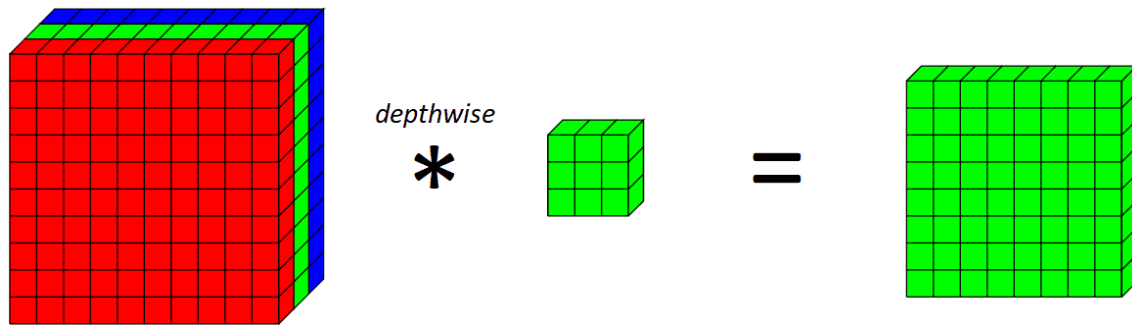


Figura 2.9. Aplicación de un kernel (en verde) de 3x3x1 al canal verde de una imagen (10x10).

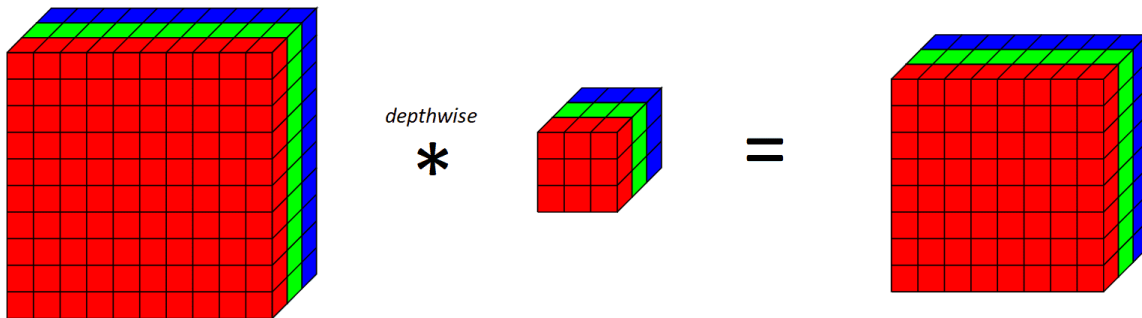


Figura 2.10. Aplicación de un kernel por canal a una imagen (10x10).

La diferencia clave entre una capa convolucional normal y una convolución en profundidad es que la convolución en profundidad estrictamente aplica la convolución a lo largo de solo una dimensión espacial (es decir, el canal) cada vez, mientras que una convolución normal suele aplicarse en todas las dimensiones/canales espaciales en cada paso [17]. Como se aplica un filtro por canal, el número de canales a la salida es el mismo. Gráficamente se observa comparando la Figura 2.3 y la Figura 2.10. Nótese que se aplica un filtro (kernel) único por capa.

2.1.8.1.2 Convolución Puntual (Pointwise)

Después de aplicar la convolución en profundidad, se realiza la convolución puntual. Esta consiste en una convolución normal, pero de 1x1 sobre todos los canales.

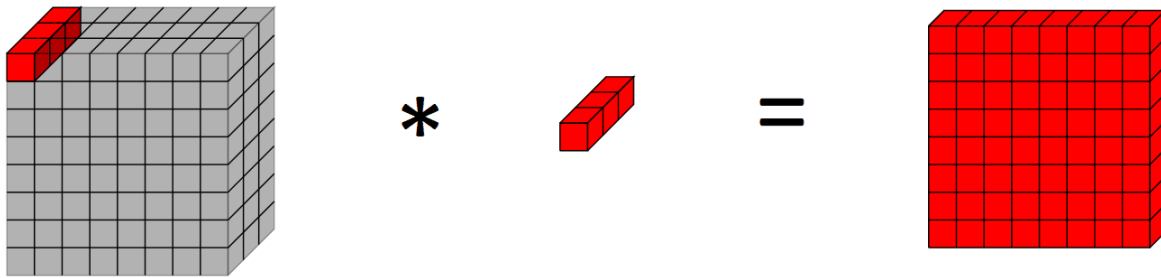


Figura 2.11. Aplicación de un kernel de 1x1 a una imagen.

Al final ambos tipos de convolución, una estándar y una separable en profundidad, bajo los mismos parámetros arrojan el mismo resultado. En ambos casos se pueden utilizar varios filtros para aumentar la dimensionalidad de la red, sin embargo, en la convolución separable en profundidad estos se añaden a partir de los kernels de 1 x 1 en la convolución puntual. Entre más kernels de 1 x 1, más canales a la salida. Este concepto se aprecia en la Figura 2.12.

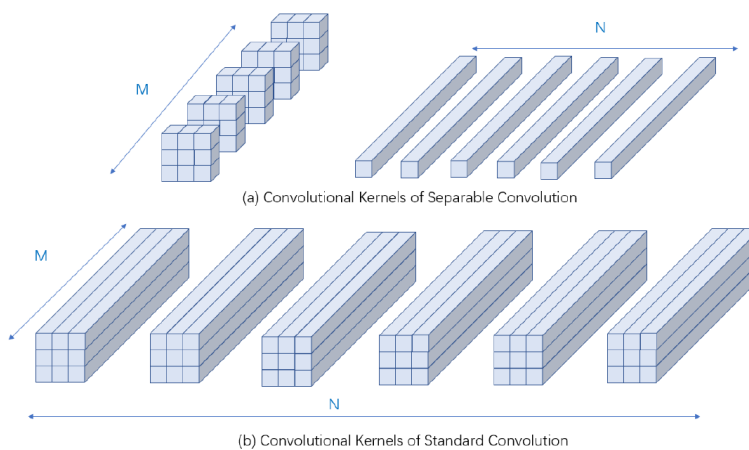


Figura 2.12. Kernels de una convolución separable en profundidad y una convolución estándar.

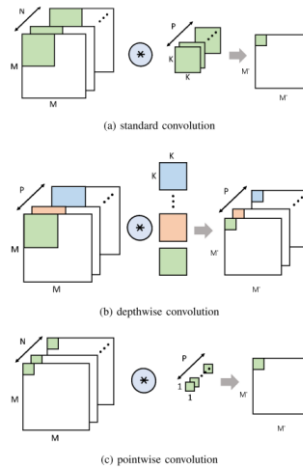


Figura 2.13. Diferentes tipos de convolución estándar

2.1.8.2 Convolución por Transformada rápida de Fourier

Una de las técnicas de convolución para el ahorro de la carga computacional implica el uso de la llamada "Transformada Rápida de Fourier" (FFT). La convolución por FFT se basa en el principio de que la multiplicación en el dominio de frecuencia corresponde con la convolución en el dominio del tiempo. La señal de entrada se transforma en el dominio de frecuencia utilizando la Transformada de Fourier Discreta (DFT), se multiplica por la respuesta en frecuencia del filtro y luego se transforma de nuevo al dominio del tiempo mediante la DFT inversa.

Se puede analizar la FFT como una capa de procesamiento que se mueve por una imagen y un kernel, realizando cálculos específicos en cada ventana de la imagen y el kernel mientras se desplaza por toda la imagen. Cada punto en la imagen es modificado de acuerdo con las operaciones necesarias para realizar una convolución.

La FFT es una técnica que ayuda a acelerar el procesamiento de imágenes y convoluciones al aplicar transformaciones matemáticas a los datos, lo que facilita y agiliza el procesamiento de imágenes en aplicaciones como el procesamiento de señales o la visión por computadora. [25]

2.1.8.3 Convolución por Winograd

Existen varios algoritmos para redes convolucionales y algunas alternativas más eficientes en términos de velocidad y uso de recursos. Sin embargo, estas

alternativas a menudo sacrifican ciertos aspectos, como el ancho de banda o los componentes computacionales. Uno de estos algoritmos eficientes es el algoritmo de Winograd.

Para comprender la velocidad y el ancho de banda requeridos por este algoritmo, se puede comparar con la convolución espacial tradicional. En general, el algoritmo de Winograd es más rápido y utiliza menos recursos computacionales, pero pone una mayor carga en el ancho de banda, lo que significa que puede requerir una mayor capacidad de transferencia de datos.

El algoritmo de Winograd, basado en la teoría de filtrado mínimo, se introduce en capas que involucran kernels pequeños. En comparación con implementaciones más convencionales, el algoritmo de Winograd reduce el número de multiplicaciones al utilizar resultados de un filtrado intermedio.

Cuando se trabaja en hardware, la estrategia clave es aprovechar al máximo la capacidad de paralelización. En algunos casos, esto puede dar lugar a momentos en los que no todos los componentes de la unidad de procesamiento se utilizan al máximo de su capacidad. El algoritmo de Winograd está diseñado para aprovechar al máximo este ancho de banda adicional que se crea a través de la paralelización. A diferencia de los algoritmos convencionales, que a menudo se centran en el uso de una sola dimensión de los recursos, el algoritmo de Winograd tiene como objetivo aprovechar todas las dimensiones disponibles para lograr una mayor eficiencia en términos de operaciones realizadas.

El algoritmo de Winograd puede ser expresado con la siguiente fórmula matemática:

$$Y = A^T [(G_g G^T) \odot (B^T dB)] A \quad [4]$$

siendo g el filtro o kernel, d los datos de entrada, Y los datos de salida, G la matriz de transformación de filtros, B^T la matriz de transformación de los datos, \odot la multiplicación element-wise y A^T la matriz de transformación de salida

El proceso comienza transformando las matrices de datos de entrada y los filtros en matrices de tamaño 4×4 (como se muestra en la Figura 2.14). Luego, se realiza una multiplicación elemento a elemento. En esta operación, cada elemento de la

matriz resultante en la posición (i, j) es el resultado de multiplicar los elementos en las mismas posiciones (i, j) de ambas matrices de datos y filtros.

Después de esta multiplicación, la matriz resultante de 4×4 se transforma en una matriz de 2×2 , generando así 4 valores finales de la convolución. Este enfoque nos permite reducir la cantidad de multiplicaciones, ya que realizamos el producto elemento a elemento, y al mismo tiempo obtenemos 4 píxeles de salida en lugar de solo 1.

Para ilustrar esto, si el tamaño del filtro g es $r \times r$, el tamaño de la salida Y es $m \times m$ y el tamaño de la entrada d es $(m + r - 1) \times (m + r - 1)$, el número total de multiplicaciones requeridas se reduce de 36 a 16. En otras palabras, la cantidad de multiplicaciones se reduce en un factor de 2,25 en comparación con la convolución directa. Además, una iteración de Winograd produce 4 píxeles de salida en lugar de 1, lo que contribuye aún más a la eficiencia del proceso.

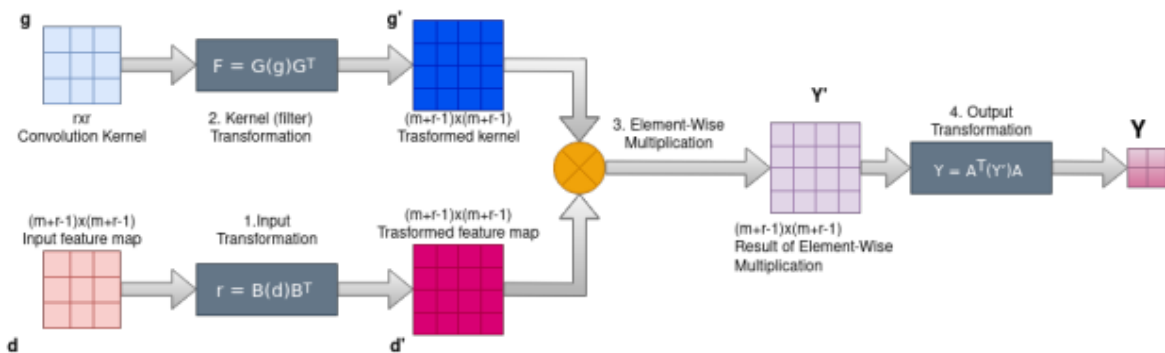


Figura 2.14. Diagrama del Algoritmo de Winograd

Es importante señalar que la implementación y configuración específicas pueden influir en la capacidad del algoritmo de Winograd para utilizar todas las dimensiones de los recursos disponibles. La adaptación precisa variará según la aplicación y hardware utilizado.

2.1.9 Consumo de recursos y número de operaciones.

Como se comentó previamente, las convoluciones propuestas tienen un menor consumo de recursos esto debido a que realiza menor número de operaciones. Para cualquier tipo de sistema de hardware, la operación de multiplicación es altamente

costosa y demandante [26] por lo que si esta se reduce se pueden alcanzar mejoras en la implementación. La idea central de utilizar convolución distinta al algoritmo original es buscar realizar el proceso de convolución de forma óptima y en un menor tiempo.

Dado lo anterior, se procede a realizar un análisis matemático de la cantidad de operaciones que se deben asociar a cada proceso de convolución, el estándar en comparación con los demás. Para esto y la comprensión de las variables a tratar obsérvese la Figura 2.13 [27].

2.1.9.1 Operaciones en la convolución estándar

Primeramente, se parte de una entrada, dada generalmente por una imagen la cual en este caso resulta ser de dimensión $M \times M$ en ancho y altura, su profundidad hace referencia al número de canales, los cuales para imágenes en el estándar RGB generalmente resulta de 3. Conforme se van trabajando las distintas capas, el número de canales tiende a crecer. El número de canales está dado por N .

El siguiente valor que se necesita definir es el de las dimensiones de los kernels. Estos en la Figura 2.13 tienen dimensiones expresadas como $K \times K$ para un total de P kernels. El número total de operaciones se calcula de la siguiente forma.

$$Op_{CE} = K \times K \times P \times N \times M' \times M' = P \times (M')^2 \times K^2 \times N \quad [5]$$

El filtro de tamaño $K \times K \times P$ se aplica un total de N veces según la cantidad de canales y se contempla la cantidad de desplazamientos según el producto de $M' \times M'$. La imagen tiene dimensiones de $M \times M$ sin embargo el kernel al desplazarse realiza $M' \times M'$ operaciones dadas por:

$$M' = \frac{\text{Dimensión de Entrada} - \text{Dimensión de Kernel}}{\text{Desplazamiento}} + 1 \quad [6]$$

2.1.9.2 Operaciones en la convolución separable en profundidad

Para la convolución separable en profundidad, el cálculo total de operaciones se calcula considerando las dos etapas, la convolución separable y la convolución puntual. La convolución separable resulta en un proceso muy similar al descrito

anteriormente por la convolución estándar. Se aplican P filtros de dimensión $K \times K$ según el número de canales y el desplazamiento se considera de igual forma como $M' \times M'$. De esta forma la cantidad de operaciones para la convolución separable resulta en:

$$Op_{CS} = K \times K \times P \times M' \times M' \quad [7]$$

Nótese que como se aplica un filtro por capa no se obtiene un único canal, a diferencia de la convolución estándar, por esta razón se debe aplicar la convolución puntual. Esta viene dado de multiplicar los kernels de 1×1 con profundidad dada por P , y considerando que se aplica un total de $M' \times M'$ según las dimensiones de la imagen. El número de filtros también representa una variable a considerar pues si se desea aplicar N filtros se debe considerar en esta parte de la convolución. La operación queda descrita como:

$$Op_{CP} = N \times P \times M' \times M' \quad [8]$$

Finalmente, si se consideran las dos partes de la convolución, el resultado final corresponde a una suma:

$$Op_{CSP} = K \times K \times P \times M' \times M' + N \times P \times M' \times M' = P \times (M')^2 \cdot (K^2 + N) \quad [9]$$

Si comparamos matemáticamente el consumo de ambas, este se puede calcular a partir del siguiente ratio.

$$Ratio = \frac{Op_{CSP}}{Op_{CE}} = \frac{P \times (M')^2 \cdot (K^2 + N)}{P \times (M')^2 \times K^2 \times N} = \frac{1}{N} + \frac{1}{K^2} \quad [10]$$

Por lo tanto, para un ejemplo básico donde se quiere utilizar 9 filtros de 3×3 , el ratio corresponde a que la convolución separable en profundidad va a realizar 9 veces menos operaciones.

2.1.9.3 Operaciones en la convolución FFT

El proceso computacional de una imagen de tamaño $n \times n$ implica n^2 multiplicaciones y $n(n - 1)$ adiciones. Sin embargo, utilizando la FFT previamente explicada, es posible procesar la misma imagen con $n^2 \log_2(n)$ multiplicaciones y $n \log_2(n)$ adiciones.

$$Op_{FFT} = n^2 \cdot \log_2(n) + n \cdot \log_2(n)$$

Esto representa una mejora significativa, ya que reduce la complejidad computacional de $O(n^2)$ a $O(n^2 \times \log_2(n))$, en comparación con la convolución espacial. [25]

2.1.9.4 Operaciones en la convolución de Winograd

Al aplicar la convolución de Winograd, primero, se transforma la matriz de entrada y filtros en matrices de 4×4 . Luego, se realizan multiplicaciones elemento a elemento en estas matrices más pequeñas, lo que requiere un total de 16 operaciones por cada matriz de 4×4 .

La eficacia de Winograd radica en la transformación de las matrices a un tamaño reducido de 4×4 y la realización de únicamente 16 operaciones, lo que permite obtener 4 píxeles de salida en lugar de tan solo 1. De esta manera, en lugar de realizar $(M')^2 \times K^2$ multiplicaciones, nos limitamos a realizar 4 multiplicaciones por el número total de píxeles de salida, es decir:

$$Op_{CW} = 4 \times m \times m$$

La convolución de Winograd permite reducir drásticamente la cantidad de operaciones necesarias al transformar las matrices y realizar multiplicaciones elemento a elemento en matrices más pequeñas. Además, obtenemos múltiples píxeles de salida con cada iteración, lo que aumenta la eficiencia del proceso. Por lo tanto, podemos decir que se reduce la cantidad de operaciones en un factor de 2,25 en comparación con la convolución directa y que cada iteración de Winograd produce 4 píxeles de salida en lugar de solo 1, lo que contribuye a su eficiencia.

2.1.10 Consumo de recursos y operaciones de red densa.

Aunque el diseño y desarrollo se encuentra fuera del proyecto, esta resulta ser parte fundamental para entender la configuración final de una red neuronal convolucional. La red neuronal densa consiste principalmente de multiplicaciones de matrices. Una red neuronal densa con M entradas y N salidas es una matriz de $M \times N$ que se multiplica por una matriz $M \times 1$ de entradas y obtiene una salida de $N \times 1$. La matriz de $M \times N$ almacena los pesos. El total de multiplicaciones y adiciones (MACs) resulta en:

$$Op_{RND} = M \times N \quad [11]$$

2.2 FPGA

Las Field-Programmable Gate Arrays (FPGAs) han emergido como una tecnología versátil y poderosa en el campo de la electrónica y la computación. Estos dispositivos, que permiten la reconfiguración de hardware mediante programación, han revolucionado la forma en que se desarrollan y despliegan sistemas digitales. Las FPGAs resultan sumamente útiles para diseñar unidades de procesamiento que aborden problemas específicos, es decir, diseñar aceleradores de hardware. En esencia, estos aceleradores corresponden a sistemas computacionales que cuentan con unidades de procesamiento diseñadas para mejorar el tiempo de cálculo de problemas particulares. Un acelerador es más eficiente que un procesador de propósito general en la aplicación que ejecuta, ya que ha sido específicamente diseñado para esa tarea.

Las FPGAs son plataformas flexibles que se moldean de acuerdo con las necesidades de la aplicación, debido a que su funcionamiento interno consiste en un arreglo denso de bloques básicos lógicos programables. Estos bloques permiten obtener resultados similares a los de un circuito integrado especializado en el problema específico [28]. Entre sus limitantes está el espacio físico y cantidad de recursos necesarios para ciertas aplicaciones además de que en comparación con el desarrollo en otros hardwares, el proceso de diseño es más costoso [29].

2.2.1 Recursos en la FPGA

Una de las grandes ventajas del uso de FPGAs es en la reducción en el consumo de recursos. Ahora bien, es importante revisar cuales son los recursos utilizados y sobre los cuales se evalúa la FPGA.

2.2.1.1 LUT

Estos se conocen como *Look-Up Tables*. Estos corresponden a tablas que generan una salida basada en sus entradas, tal como las tablas a nivel de compuertas lógicas. Los valores que describen sus salidas son ajustables y definen y dirigen el comportamiento de la lógica combinacional del chip basado en el código.

2.2.1.2 DSP

Un DSP es un elemento lógico de procesamiento de señales digitales incluido en ciertas familias de dispositivos FPGA. Esta unidad se utiliza para realizar diferentes tipos de operaciones aritméticas y lógicas. La unidad de DSP está optimizada para tener un bajo consumo de energía, alta velocidad y un tamaño reducido, al mismo tiempo que mantiene su flexibilidad. [30]

2.2.1.3 BRAM

La Memoria de Acceso Aleatorio en Bloque (BRAM) es el principal elemento de memoria en las FPGA. Está dispersa e intercalada con otros componentes configurables como DSP y LUT en FPGA. La estrecha interacción con DSP y LUT brinda una gran flexibilidad a la Memoria de Acceso Aleatorio en Bloque. Esta es un módulo de memoria RAM de doble puerto instanciado en la estructura de la FPGA para proporcionar almacenamiento en el chip para un conjunto relativamente grande de datos. [15]

2.2.2 High-Level Synthesis

Para entender qué es la síntesis de alto nivel (high-level synthesis), es necesario partir desde las bases del diseño de hardware en FPGAs, y entender lo que es "register transfer level" (RTL). El RTL es un nivel de abstracción en el diseño de hardware que, con la ayuda de lenguajes como VHDL y Verilog, permite modelar representaciones de un circuito digital. Se describe cómo se realiza la transferencia

de datos de un registro a otro y cómo se organizan y ejecutan las operaciones lógicas y aritméticas en el circuito.

A lo largo de muchos años, el diseño por RTL ha sido ampliamente utilizado debido a su flexibilidad y capacidad para expresar el circuito de manera clara y precisa. Sin embargo, hace apenas unos años, se logró alcanzar un nivel de abstracción aún más alto conocido como la síntesis de alto nivel (high-level synthesis), que permite trabajar con circuitos digitales bajo un nuevo paradigma.

HLS convierte un algoritmo programado en un lenguaje de alto nivel como 'c' o 'c++' en un circuito en hardware que lo ejecuta. Esto permite gran flexibilidad, rendimiento, costo y reducción en potencia sobre los procesos tradicionales [31]. Funciona como un puente entre los dominios del software y el hardware que ayuda en la productividad.

Ahora bien, no todas las FPGAs en el mercado están acondicionadas para soportar HLS, y si se consideran aquellas de bajo perfil son incluso menos. Esto representa un reto en cuanto a la implementación [32].

Parte de las ventajas con respecto al diseño en RTL, viene de los denominados pragmas. Estos pragmas son directivas que se incluyen en el código para controlar el comportamiento y la síntesis del circuito. De esta manera, es posible gestionar la forma en que se manejan los datos y procesos para habilitar operaciones en paralelo o secuenciales, según sea necesario.

2.2.2.1 Niveles de Abstracción: HDL y HLS

Inicialmente, en el diseño de circuitos digitales, se empleaban optimizaciones con mapas de Karnaugh y esquemas de compuertas para diseñar circuitos integrados (CIs) a nivel de transistores. La reducción del tamaño de los transistores impulsó la automatización. Los desarrolladores cambiaron de diseño manual de CIs a descripciones a nivel de compuertas. Luego, surgió la descripción a nivel de transferencia de registros para circuitos digitales síncronos. Posteriormente, la síntesis lógica incrementó la abstracción, permitiendo un flujo de control similar a

lenguajes de programación (if/else, case, while, for, etc.) en lenguajes de descripción de hardware (HDLs) como Verilog y VHDL.

Estos lenguajes de descripción de hardware como Verilog y VHDL siguen siendo populares en el diseño digital, sin embargo, muchos coinciden en que no son la mejor opción para diseñar circuitos electrónicos. Su naturaleza detallada requiere experticia para usarlos eficientemente, son propensos a errores y la depuración es más compleja comparada con lenguajes de programación. Además, con HDLs se deben tomar decisiones menores de diseño. [33]

Altos costos de desarrollo, demoras en el mercado y procesos ineficientes son motivos para elevar el nivel de abstracción. De esa forma se busca un entorno similar a lenguajes de programación para mayor abstracción que los HDLs.

El high-level synthesis (HLS) busca aislar al desarrollador de decisiones de diseño detalladas, como señales de control, direcciones de puertos, cables y registros intermedios. Las herramientas de HLS buscar generar hardware más eficiente y optimizado. La Figura 2.15 ilustra los niveles de abstracción en el diseño digital.

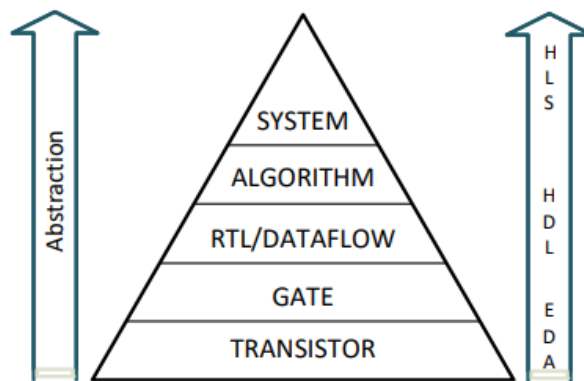


Figura 2.15. Niveles de abstracción del diseño digital.

En [33] se presenta un estudio exhaustivo en cuanto a uso de recursos, rendimiento y el proceso de diseño entre HDL y HLS. En este se explica como si bien se requiere un conocimiento significativo de hardware para implementar un diseño RTL eficiente, con la ayuda de las herramientas HLS, los desarrolladores pueden generar resultados comparables sin adentrarse demasiado en los detalles de arquitectura.

A nivel de RTL se ofrece flexibilidad y un enfoque de diseño a bajo nivel que puede brindar al desarrollador una plataforma de mayor rendimiento. Sin embargo, la eficiencia del diseño depende en gran medida del nivel de competencia del desarrollador en el diseño de hardware. Por lo tanto, el método RTL presenta la posibilidad de errores humanos que pueden resultar en sistemas con un rendimiento y eficiencia más bajos que el método HLS.

El autor indica que incluso los avances tecnológicos nos han demostrado que, en circunstancias específicas, las máquinas pueden tomar decisiones mejores que los humanos. Al aislar al desarrollador de la toma de decisiones de diseño a bajo nivel, las herramientas de HLS tienen el potencial de crear diseños más eficientes.

2.2.2.2 Optimizaciones

Una de las capacidades de HLS es que un único fragmento de código en C++ puede llevar a varias implementaciones de diseño controladas por directivas del compilador (también conocidas como pragmas). En otras palabras, es posible explorar varios diseños utilizando el mismo código y realizar una exploración del espacio de diseño con directivas. [12] En la tabla a continuación se presenta un resumen presentado por Luis León en cuanto a las directivas soportadas y de interés en el proyecto para HLS 2018.2. Cada una de estas tiene un enfoque que según el presente contexto resulta de interés. Los *cores* son recursos de FPGA o componentes de bibliotecas RTL que implementan operaciones como la división, la multiplicación, entre otras. Las funciones, por otro lado, se refieren a las funciones de C++ que se comportan como bloques de hardware, diseños o módulos. Los contenedores de datos suelen ser estructuras en C++ y los loops son a menudo *for loops* (los bucles *while* no son completamente compatibles en HLS).

Tabla 2.1. Directivas de diseño en HLS 2018.2. [12]

Directiva	Descripción	Enfoque
ALLOCATION	Especifica el límite de un recurso o función (forzar el uso compartido de hardware).	Funciones, cores
ARRAY MAP	Combinas arrays pequeños en un solo array para reciclar memoria.	Arrays
ARRAY PARTITION	Particiona arrays grandes en varios arrays pequeños.	Arrays
ARRAY RESHAPE	Combina los elementos de un array en palabras más grandes para mejorar el acceso a la memoria.	Arrays
DATA PACK	Combina los elementos de una estructura en una sola palabra para mejorar el acceso.	Contenedores de datos
DATAFLOW	Permite la concurrencia a nivel de tarea. Hace que las funciones y los bucles funcionen de manera concurrente.	Loops, Funciones
DEPENDENCE	Proporciona una indicación sobre las dependencias. Mejora la canalización de bucles y reduce los intervalos.	Loops
INLINE	Incorpora lógica, evitando la comunicación entre funciones. Reduce la latencia y la lógica.	Funciones
INTERFACE	Dicta cómo deben implementarse los puertos en el RTL.	Argumentos de funciones y returns
LATENCY	Especifica las restricciones en términos de latencia.	Funciones, loops
LOOP FLATTEN	Permite colapsar bucles anidados en un solo bucle con una latencia mejorada.	Loops
LOOP MERGE	Combina bucles consecutivos para una latencia mejorada y un uso compartido de recursos.	Loops
OCCURRENCE	Especifica la ocurrencia de un fragmento de código dentro de un bucle.	Loops, functions
PIPELINE	Reduce el intervalo de inicialización al permitir que las funciones internas se ejecuten de manera concurrente.	Loops, functions
STREAM	Especifica que una matriz debe implementarse como una FIFO.	Arrays
TOP	Declara una función como el módulo superior.	Functions
UNROLL	Desenrolla los bucles para crear unidades o trayectos de ejecución independientes.	Loops

Distintas soluciones y arquitecturas basadas en un mismo código se pueden alcanzar por medio de las combinaciones de distintos pragmas. Algunas de las arquitecturas se representan y discuten a continuación.

2.2.2.2.1 Diseño en Serie

La mayoría del tiempo, cuando no se especifica algún tipo de directiva, el diseño base de la arquitectura corresponde a uno en serie que se representa en la Figura 2.16. En este una instrucción no se ejecuta hasta que la anterior se haya ejecutado.

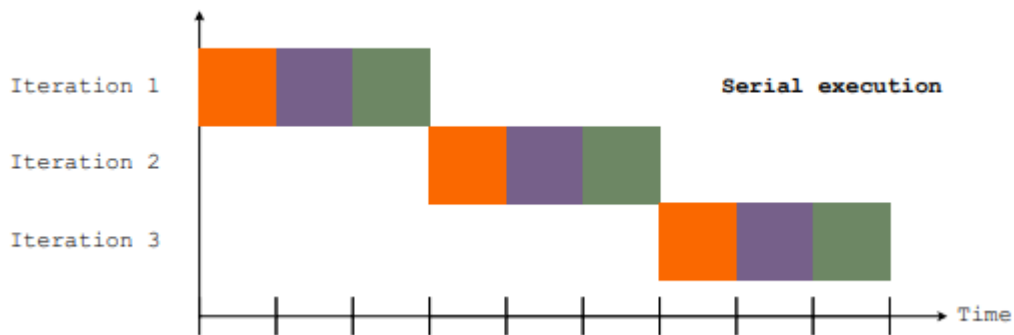


Figura 2.16. Lógica de una arquitectura en serie.

2.2.2.2.2 Diseño Pipeline

La arquitectura pipeline es uno de los diseños más utilizados en la modernidad ya que se trata de una mejor utilización del hardware. Distintas instrucciones se pueden ejecutar sin la necesidad que termine una instrucción distinta anterior, sino que se espera a que termine una iteración de la misma instrucción para ejecutar una nueva. Esto se visualiza en la Figura 2.17.

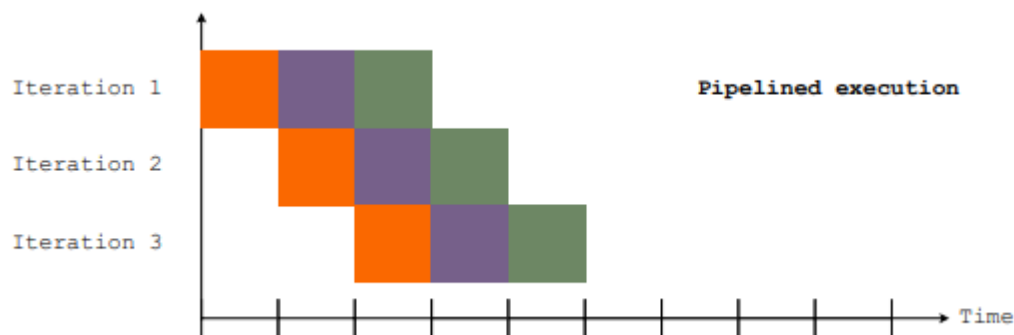


Figura 2.17. Lógica de una arquitectura pipeline.

2.2.2.2.3 *Diseño por Dataflow*

Este tipo de diseño de arquitecturas se enfoca en el manejo y flujo de los datos. La idea es que tomando cada uno de los colores como un módulo, si un paquete está listo en el módulo 1 y se puede ir trabajando en el módulo 2, este se envíe para empezar a procesarlo. El diseño por dataflow también se le conoce como paralelismo a nivel de tarea ya que se enfoca en la tarea a nivel de datos. Esto también hace que el manejo de la división del tiempo sea más difusa. Para esta arquitectura se utilizan interfaces FIFO entre los módulos para enviar datos cuando estos se hayan procesado.

2.2.3 Interfaces y Frameworks de Integración

Una FPGA puede ser empleada como un chip independiente conectado a dispositivos de entrada/salida (I/O) y utilizada como el núcleo de control y cómputo del sistema. Sin embargo, generalmente, en aplicaciones de aceleración de hardware, las FPGAs se utilizan como co-procesadores conectados a un sistema de procesamiento (SP). En una aplicación típica de aceleración de FPGA, esta toma los datos del SP, los procesa y envía los datos procesados nuevamente al SP. Toda esta comunicación requiere una interfaz para gestionar el flujo de datos entre el SP y la FPGA. [33]

El Protocolo de Interfaz Avanzada extensible (AXI) es actualmente adoptado por las FPGAs de Xilinx. El protocolo AXI ofrece una interfaz estándar para los desarrolladores. Existen tres tipos de protocolos AXI4 para diferentes aplicaciones:

AXI4: Este protocolo se prefiere cuando el diseño requiere una interfaz de mapeo de memoria.

AXI4-Lite: Este protocolo es un subconjunto del protocolo AXI4 y está diseñado para requerimientos de interfaz más simples. Algunas señales se eliminan de la interfaz AXI4 para reducir la lógica de diseño. AXI4-Lite es una interfaz de mapeo de memoria, pero no admite ráfagas de datos, está diseñada para transacciones individuales.

AXI4-Stream: Este protocolo está diseñado para admitir la transmisión de datos y es un subconjunto de la interfaz AXI4. Las señales de dirección se eliminan de la interfaz AXI4, ya que los datos se transmitirán y no se mapearán en la memoria. Este protocolo se prefiere para aplicaciones de transmisión de datos, como aplicaciones de procesamiento de imágenes.

Utilizar las interfaces AXI4 puede ser una tarea difícil si el diseño se desarrolla puramente con un lenguaje de descripción de hardware (HDL). Por otro lado, el uso de interfaces AXI4 es menos complejo con Vivado HLS. [34]

2.2.4 Aceleradores

Una de las aplicaciones más destacadas de las FPGAs en la actualidad es su utilización para el diseño de aceleradores de hardware. Estas arquitecturas están diseñadas para ejecutar de manera altamente optimizada todo o parte de un algoritmo de cómputo en una FPGA, aprovechando eficientemente los recursos disponibles [35].

El propósito principal de crear una arquitectura de acelerador es resolver un algoritmo específico de manera dedicada, adaptando el manejo de datos y variables de acuerdo con las características particulares de dicho algoritmo. Para mayor comprensión se puede contrastar con las CPU, que son de propósito general y pueden resolver diversos problemas. Un acelerador se enfoca en resolver un problema específico, reduciendo el hardware necesario al utilizar un diseño especializado para ejecutar el algoritmo de manera más eficiente.

Existen frameworks y herramientas que facilitan el desarrollo en FPGA, optimizando el uso de memoria y aprovechando el paralelismo de operaciones dentro de un bucle [36]. Son sumamente utilizados dentro del contexto del machine learning y la visión por computadora.

2.3 Flexible Accelerators Library

La *Flexible Accelerators Library* (FAL) es un marco de trabajo (framework) de código abierto desarrollado en colaboración entre varias universidades y colaboradores. Este tiene como objetivo generar aceleradores genéricos y personalizables que

permitan escalar el diseño de hardware en el campo del aprendizaje de máquina, visión por computadora y una serie de otras aplicaciones que pueden inclusive no estar relacionadas a la inteligencia artificial. El *framework* propone un mecanismo para realizar exploración del espacio de diseño para medir la capacidad y flexibilidad en la variación de los tipos de datos, el número de operandos entre otras variables.

FAL se concentra en una serie de retos que actualmente están presentes con las herramientas de desarrollo existentes tales como: [12]

- Aprovechar la Inteligencia Artificial (IA) en FPGAs de gama baja para una mejor explotación de sus capacidades de bajo consumo.
- Desarrollo de unidades de hardware genéricas y fáciles de personalizar para Deep Learning con flexibilidad para optimizaciones y modularidad para ajustar el consumo de recursos.
- Automatización de tareas de exploración de diseño.
- Análisis cuantitativo entre implementaciones para conjuntos de parámetros seleccionados.

Este hace uso de programación genérica en C++ estándar y Síntesis de Alto Nivel para implementar elementos de procesamiento para aceleradores modulares en combinación con cómputo aproximado (AxC) para reducir la huella de consumo de recursos a cambio de la precisión de los resultados.

2.3.1 Approximate Computing (AxC)

Algunas aplicaciones pueden ser aproximadas debido a su resistencia al error [37], sacrificando precisión de resultados para reducir la necesidad de unidades aritméticas complejas. Este paradigma se conoce como *Approximate Computing*.

Dentro del contexto de FAL, el *Approximate Computing* es de sumo interés. Dada su inherente imprecisión, las aplicaciones probabilísticas suelen ser buenos candidatos para la aproximación, como el aprendizaje automático basado en DNNs. Por lo general, las aplicaciones de AxC incluyen parámetros para ajustar el grado de aproximación, es decir, la precisión numérica de una etapa determinada en una aplicación base.

AxC, junto con la programación genérica, facilita la explotación de la resistencia a errores de las aplicaciones, intercambiando errores numéricos para obtener un menor consumo de energía y diseños más pequeños, aumentando la idoneidad de las FPGAs de gama baja para la aceleración.

2.3.2 AxC Executer

El AxC Executer es un framework en software que se encarga de realizar operaciones específicas relacionadas con redes neuronales convolucionales. Utiliza plantillas de C++ para describir operaciones de red neuronal, aprovecha la herencia para construir módulos de ejecución y permite la personalización de estos módulos para realizar cálculos específicos de manera eficiente.

Dado que cada módulo es reemplazable, es posible implementar módulos personalizados para probar optimizaciones y aproximaciones y transferir la computación a unidades de aceleración. Del mismo modo, es posible reemplazar las unidades aritméticas (sumadores y multiplicadores) para obtener un mayor grado de aproximación. [38]

A partir de esta librería se puede simular en software, los modelos desarrollados y configurarlos a partir de la estructura deseada.

2.3.3 Frameworks populares para DL

Esta sección presenta hls4ml y FINN, los marcos de aprendizaje automático más populares para realizar inferencias en dispositivos FPGA. El CERN respalda el primero, y el segundo es respaldado por la academia y AMD/Xilinx (uno de los principales proveedores de FPGA).

2.3.3.1 HLS4ML

El framework de hls4ml [39] implementa un flujo de trabajo que recibe el modelo en TensorFlow o PyTorch. Luego, realiza una conversión HLS y crea un proyecto que permite al usuario ajustar el diseño: estableciendo objetivos de optimización y ajustando la precisión numérica y el factor de reutilización. hls4ml proporciona ajuste de diseño en tiempo de compilación, lo que permite la síntesis del diseño gracias al uso de C++ en HLS.

El rendimiento óptimo en HLS4ML está determinado por:

- Tamaño y compresión del modelo: permitiendo reducir recursos.
- Precisión: la precisión numérica determinará el error del modelo y cuán precisa será la DLI.
- Flujo de datos y reutilización de recursos: el usuario puede ajustar el equilibrio entre la reutilización de recursos y el rendimiento.

2.3.3.2 *FINN*

FINN es un marco de aprendizaje automático para la implementación de modelos cuantificados en FPGAs, respaldado por AMD/Xilinx. Al igual que hls4ml, toma un modelo de Python y produce un diseño HLS capaz de ejecutarse en FPGAs Xilinx. Sin embargo, FINN, a diferencia de hls4ml, se centra en PyTorch como el principal marco de aprendizaje automático. Además, en lugar de generar un solo acelerador, FINN adopta un enfoque más granular, generando un acelerador único por capa. Cada capa puede estar más optimizada que otras, ya que el factor de reutilización (también conocido como factor de plegado) se aplica por capa.

FINN también admite la simulación de software de todo el data-path gracias a la simulación HLS. Esto es útil para la verificación del diseño y al calcular el factor de plegado de cada unidad desarrollada y ver cómo se comportan los resultados finales en cuanto a rendimiento y consumo de recursos. Además, la simulación puede realizarse a nivel de Python, código HLS y Verilog, lo que ofrece una verificación completa del diseño en sus diversas etapas.

2.3.3.3 *Oportunidades*

Ambas soluciones (HLS4ML y FINN) fueron concebidas para FPGAs de gama alta y utilización en la nube, lo que hace que carezcan de granularidad en el diseño y limite la posibilidad de implementar diseños en FPGAs de gama baja. Estas oportunidades corresponden a:

- Granularidad: ambos enfoques implementan las unidades del modelo completo en el FPGA. No hay posibilidad de distribuir el cálculo entre el FPGA y una CPU anfitriona para una ejecución híbrida.

- Computación aproximada: ambos marcos son precisos en sus cálculos. Abre la oportunidad de experimentar con técnicas de computación aproximada.
- Tamaño del modelo: los diseños producidos por ambos marcos aún pueden ser inadecuados para FPGAs de gama baja. La combinación de los dos primeros desafíos puede llevar a una mejora en este problema.
- Apertura a otras aplicaciones: ambos marcos están altamente adaptados a Deep Learning y restringen su uso en otras aplicaciones como Álgebra Lineal. Un enfoque más genérico podría satisfacer tanto al DL como a otros campos también.
- Mala documentación y portabilidad: ambos marcos se encuentran mal documentados y con instrucciones poco claras para su uso general.

2.4 Estado del Arte

Se realizó una investigación para determinar cuáles eran las alternativas para un modelo de clasificación de objetos según una red convolucional y alcanzar una posible solución.

2.4.1 Modelos en la academia

En la academia ha crecido el interés de desarrollar modelos pequeños y eficiente de redes neuronales. En [7] se expone como las aproximaciones actuales se han dividido en dos caminos; comprimir las redes pre-entrenadas o entrenar redes más pequeñas. Por medio de MobileNet, que introduce el desarrollo de un modelo ajustable en tamaño según los requerimientos de la aplicación, se apunta a no solo considerar tamaño sino también la velocidad (latencia).

La convolución separable en profundidad se introdujo en [23] y subsecuentemente se utilizó en Inception Models [18] para reducir la computación de las primeras capas. Este último introduce Batch Normalization para entrenar redes neuronales profundas. Una de las aproximaciones novedosas en cuanto a la forma en que se desarrolla el proceso de convolución en redes neuronales corresponde al uso de redes enflaquecidas (Flattened networks).

2.4.2 Métodos de convolución optimizados

Bastantes métodos se han explorado para reducir el número de parámetros y el número de operaciones enfocándose principalmente en la capa de convolución. Uno de los métodos es por medio de factorización del kernel. La factorización puede ser utilizada para descomponer una convolución de dimensiones superiores en una secuencia de convoluciones de dimensiones más bajas, lo que resulta en una complejidad computacional menor y aproximadamente el mismo resultado. Estas factorizaciones pueden ser reflejadas mediante una arquitectura de red neuronal modificada, al factorizar directamente los núcleos convolucionales ya entrenados, o a través de una combinación de ambos enfoques. Un ejemplo es justamente la convolución separable en profundidad descrita anteriormente, pero la forma en que se descomponen los kernels puede variar en dimensiones u otros factores.

Otro método para implementar el algoritmo de convolución resulta en convolución por la transformada rápida de Fourier (FFT). Al transformar las imágenes y los núcleos de redes neuronales convolucionales al espacio de frecuencia mediante la transformación rápida de Fourier un operador de convolución se convierte en una multiplicación elemento por elemento. Por lo tanto, FFT se utiliza ampliamente para la convolución de imágenes. Sin embargo, hay un costo adicional asociado a la realización de la FFT. Este tipo de transformación es eficiente para reducir el costo de convolución solo cuando la convolución es grande. Recientemente, las convoluciones de las redes neuronales convolucionales han sido optimizadas utilizando FFT y se están adaptando en sistemas móviles y embebidos. [40]

El último método que se discute es la convolución mediante el método de Winograd. El algoritmo de Winograd, basado en teoría de filtrado mínimo, se implementa en capas con núcleos pequeños. En comparación con las implementaciones más convencionales, el algoritmo de Winograd reduce el número de multiplicaciones utilizando los resultados de un filtrado intermedio. Calcula los recursos de manera más eficiente, ejerciendo una gran presión en el ancho de banda. La ventaja de la aceleración con FPGA consiste en paralelizar tanto como sea posible, generando

momentos en los que no todos los componentes se utilicen. Este algoritmo busca explotar al máximo el ancho de banda adicional creado por la paralelización. [25]

2.4.2.1 Line-buffer Convolution

La convolución utilizando line buffers es un método novedoso para ejecutar la operación de convolución en hardware. La solución con esta estructura consiste en asegurarse de que `data_out` y `data_in` se accedan la menor cantidad de veces posible. El caso ideal es, por supuesto, que cada posición se acceda solo una vez durante toda la ejecución [41].

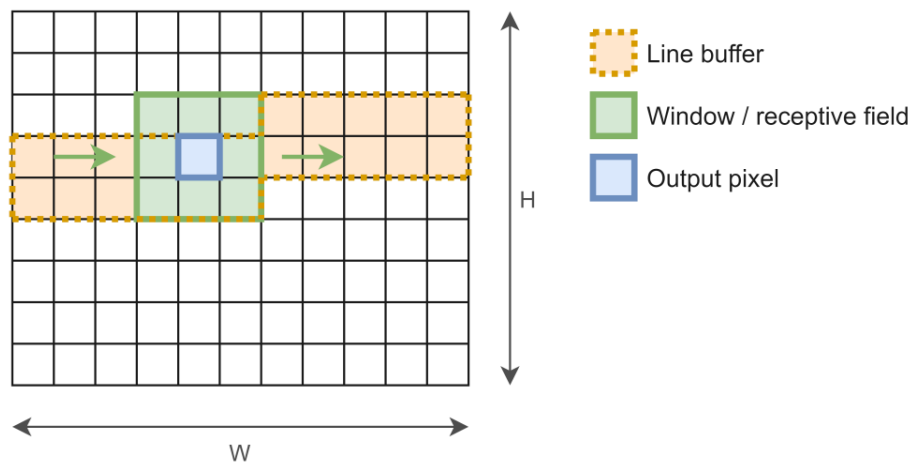


Figura 2.18. Operación de line buffer en convolución en 2D.

La figura de arriba muestra cómo se almacenan los píxeles de entrada en cualquier momento para una ventana de 3x3. El line buffer suele tener dimensiones según el tamaño del kernel y el número de columnas y sirve como una especie de caché que contiene exactamente los suficientes elementos para llenar la columna más a la derecha de la ventana cuando se desplaza en cada nueva iteración. Por lo tanto, solo un nuevo píxel de entrada debe leerse desde el puerto de I/O hacia la ventana.

2.4.3 MobileNet

La MobileNet es un tipo de red neuronal convolucional diseñada con el fin de desarrollar modelos fácilmente implementables en sistemas de visión embebidos [7]. La idea central de la MobileNet es la reducción del número de parámetros por medio de la convolución separable en profundidad. El número de parámetros dentro

de una red neuronal hace referencia al número de pesos y 'biases' que la red utiliza para aprender.

Estas fueron introducidas por Google y actualmente se encuentra en su tercera versión. Cada versión ha introducido distintos conceptos y avances novedosos a la convolución separable en profundidad, los cuales se abordarán más adelante. La red se planteó para sistemas de recursos limitados tales como los procesadores utilizados en celulares móviles. Se debe tomar en cuenta que el modelo estándar presentado en [7] [8] [24] fue entrenado y evaluado utilizando las imágenes del dataset de 'ImageNet'.

Los modelos MobileNet se han aplicado en diversos campos, incluidas tareas de visión por computadora como la clasificación de enfermedades de hojas [42], clasificación de enfermedades de la piel [43], detección y clasificación de cáncer de mama [44] y clasificación de enfermedades pulmonares [45]. Estas aplicaciones demuestran la versatilidad y efectividad de MobileNet en diferentes tareas médicas y análisis de imágenes.

La MobileNet, comparada con otros modelos, presenta ventajas en latencia y precisión y se debe analizar si resulta en general ser una mejor opción según sea el caso de aplicación. Su principal punto para considerar es que existen modelos más precisos pero que utilizan hasta 10 veces más recursos utilizados para ello.

2.4.3.1 MobileNetV1

Esta corresponde al primer modelo presentado. Como tal se presenta un sistema que realiza la convolución separable en profundidad a la vez que introduce dos hiper-parámetros globales que permiten decidir entre latencia y precisión. Estos hiper-parámetros permiten escoger el tamaño ideal según sea la aplicación y las limitaciones del problema.

En cuanto a las características de la red, esta consiste en una serie de capas conformadas por, capas para la convolución separable, la convolución puntual, normalización del batch de entrenamiento, funciones de activación dadas por ReLU. El down-sampling para reducir la dimensionalidad de la red y poder hacerla más

trabajable (menor tamaño – menor número de operaciones), se realiza por medio de convoluciones con un stride alto. Además, también se realiza un proceso de average pooling a la salida del modelo. Esto resulta en un total de 28 capas. El proceso en alto nivel se puede revisar en la Figura 2.19.

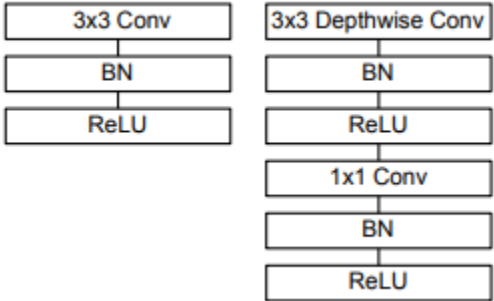


Figura 2.19. Capas de la convolución estándar y la MobileNet aplicando batch-normalization y ReLU. [7]

Una gran parte de las operaciones totales se realiza en la convolución puntual 1 x 1. Dentro del documento presentado se hace mención del uso de un algoritmo que lo ejecute tal como el de la multiplicación de matriz general (GEMM). Este algoritmo es uno de los más optimizados en álgebra lineal.

Cómo se mencionó previamente, este modelo introduce dos hiper-parámetros capaces de reducir el tamaño del modelo y por lo tanto mejorar la velocidad de ejecución. Esto puede llegar a necesitarse en distintos casos de aplicación. El primero de estos hiper-parámetros es un ‘multiplicador de ancho’ que busca enflaquecer uniformemente cada capa. Este se representa como α y para cada capa P esta tendrá un tamaño de αP . Generalmente α tiene un valor entre 0 y 1 con 1 como la base. El segundo hiper-parámetro es el de multiplicador de resolución dado por la letra ρ . Este reduce la resolución de las entradas y se define implícitamente a la hora de escoger el tamaño de la imagen que se procesa. Con estos se puede intercambiar entre precisión y baja latencia.

2.4.3.2 MobileNetV2

La idea con esta red consistía en buscar cómo reducir aún más el número de operaciones realizadas sin afectar la precisión del modelo. Su contribución

novedosa para alcanzar este objetivo corresponde a la introducción de un nuevo módulo de capa: un inversor residual con cuello de botella. Más allá del significado técnico de estas ideas, lo que significan se resume en dos puntos específicos.

- Antes de la convolución en profundidad, se agregó una capa de de convolución puntual 1×1 , que aumenta el número de canales para obtener más características.
- A lo último, no se usa ReLU y se cambia por una función de activación lineal que se usa para evitar que ReLU destruya funciones.

Estas ideas quedan más claras a partir de la comparación del proceso de convolución que ejecuta la MobileNetV1 en comparación con la MobileNetV2.

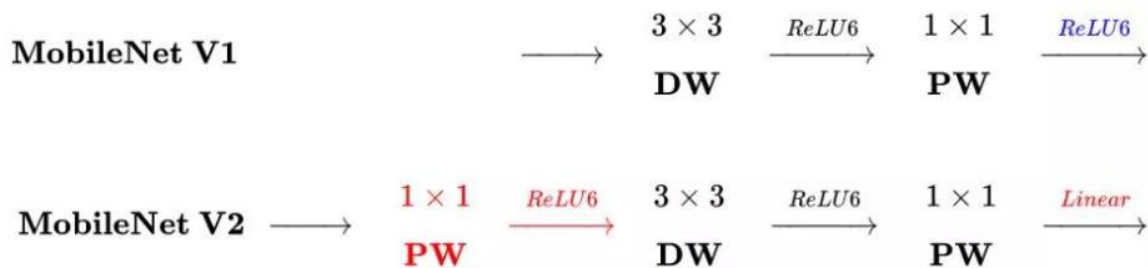


Figura 2.20. Estructura de convolución para la V1 y V2 del modelo MobileNet

La razón de esta aproximación recae en que para la MobileNetV1 con su convolución separable en profundidad las características extraídas por la capa de convolución en profundidad están limitadas por el número de canales de entrada. Esto significa que el bloque residual en ese caso, primero “comprime” y luego realiza convolución para extraer las características. De esta forma solo puede extraer muy pocas características, por lo que no es al principio “compresión”. MobileNetV2 hace lo contrario, e introduciendo una convolución puntual, “expande” la cantidad de características detectables y realiza una “compresión” al final. Todo esto es lo que se conoce como residuos invertidos.

Seguido de esto, al usar una “expansión” al inicio se encontrará un problema después de la “compresión” si se utiliza ReLU pues esta destruirá la función por su propia naturaleza. Para la entrada negativa, la salida de ReLU es cero; y las

características originales se han "comprimido", y luego, después de ReLU, algunas características se "perderán", por lo que ReLU no se usa aquí. Esto se llama cuellos de botella lineales.

2.4.3.3 MobileNetV3

La última iteración al momento de realizar esta documentación corresponde a la tercera versión. MobileNetV1 introdujo la convolución de profundidad para reducir la cantidad de parámetros. La segunda versión añadió una capa de expansión para obtener un sistema de expansión-filtrado-compresión utilizando tres capas. La última versión añade capas de "compresión y excitación".

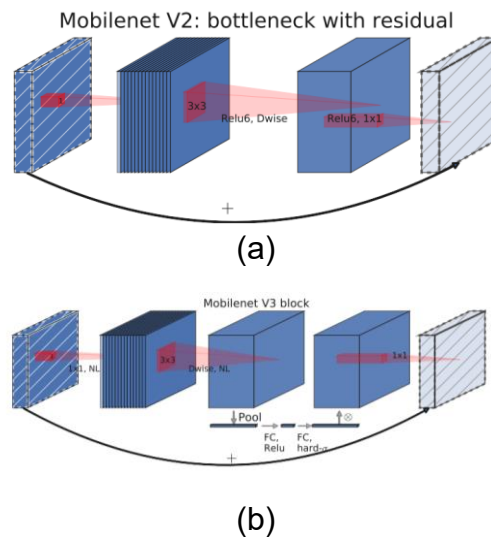


Figura 2.21. Estructura básica de MobileNetV2 y MobileNetV3

La capa de "compresión y excitación" en este modelo se añade de forma paralela y como se aprecia en la imagen consiste en un proceso de:

Pooling → *Capa densa* → *ReLU* → *Capa densa* → *h-swish* → *Escalar de vuelta*

En este flujo se introducen varios conceptos nuevos. El primero es "hard-swish" (h-swish) que corresponde a una función de activación no lineal que procede de su antecesor "swish" [46] que en la práctica se trata como una función conocida como SiLU presentada en la Figura 2.22.

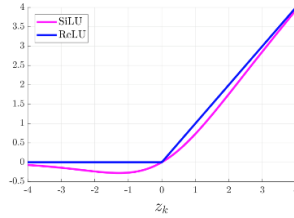


Figura 2.22. Función de activación “sigmoid-linear units” (SiLU)

La idea con “h-swish” es tener una función de activación que remplace lo computacionalmente costosa que es la función swish. Esta viene dada por la ecuación [12].

$$h - swish(x) = \begin{cases} 0 & x \leq -3 \\ x & x \geq +3 \\ x \cdot \frac{(x + 3)}{6} & otherwise \end{cases} \quad [12]$$

La función de activación y cómo se diferencia de su antecesora se visualiza en la Figura 2.23.

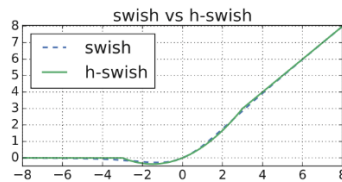


Figura 2.23. Función de activación “hard swish” (h-swish)

Esta adición del módulo de compresión y excitación ayuda al dar pesos desiguales a diferentes canales de entrada, al crear los mapas de características de salida, en contraste con el peso igual que proporciona una CNN normalmente. En el contexto algoritmo del modelo general, estos resultan ser los aportes más significativos que hay.

Por otro lado, dentro del estudio presentado en [24] se introduce una aproximación novedosa en lo que encontrar la arquitectura más óptima respecta, y es el uso de una herramienta de diseño para redes neuronal presentada en [47] y conocida como “Neural Architecture Search” (NAS) que se apoya en modelos de aprendizaje por

refuerzo para encontrar un modelo que dé la mejor precisión entre las combinaciones de la arquitectura posibles. Esto además se apoya en otra herramienta de diseño conocida como NetAdapt que consiste en un algoritmo que trabaja encontrando el número óptimo de filtros por convolución. Nótese que no se trata de encontrar nuevos valores de los filtros sino el número de filtros per se, considerando que si se requiere añadir nuevos pesos en los filtros el valor de estos se aleatoriza y se optimiza hasta encontrar el valor de precisión y latencia deseado.

Otra mejora en la red se basa en la reorganización de la arquitectura en la etapa final en la que se realiza la convolución, al llevar a cabo el proceso de agrupamiento (pooling) desde capas anteriores, en contraposición a lo que se había propuesto en la MobileNetV2. Al reorganizar este procedimiento, es posible eliminar capas que son costosas en términos de operaciones y recursos, sin aumentar significativamente la pérdida de precisión. De esta manera, el flujo del modelo se presenta de la siguiente manera:

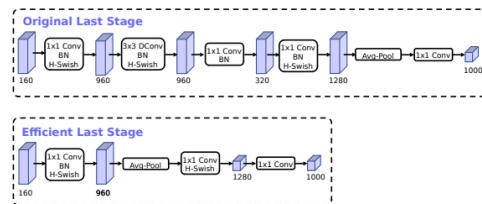


Figura 2.24. Comparación entre la última etapa de la arquitectura de la MobileNetV2 y la MobileNetV3.

Por último, la MobileNetV3 experimenta con el uso de 16 filtros a la entrada de la red en lugar de 32 que es lo común en modelos optimizados para uso en sistemas de bajos recursos. Los modelos tienden a usar 32 filtros en una convolución completa de 3x3 para construir bancos de filtros iniciales para la detección de bordes. Logra reducirlo a 16 filtros por medio del uso de la función de activación h-swish manteniendo la precisión y reduciendo millones de operaciones.

2.4.3.4 Implementación

A nivel de software

Librerías en Python como Tensorflow o Pytorch ofrecen modelos pre-entrenados y listos para utilizar de la red MobileNet en sus distintas versiones. Estos ofrecen distintos grados de parametrización, por ejemplo, el modelo de Tensorflow contempla aspectos como el tamaño de la figura a la entrada, los pesos a utilizar, el tipo de pooling o la activación, además de estar diseñada para ser un poco más fácil de integrar a un proyecto. En el caso de Pytorch se pueden configurar aspectos como los pesos y aspectos similares a Tensorflow, pero con un framework mucho más complejo y menos “user-friendly”. Por motivos de simplicidad para cualquier desarrollo necesario en Python en este trabajo se abordará el modelo de Tensorflow.

A nivel de hardware

Existen pocos trabajos que hayan abarcado la implementación de la MobileNet a nivel de hardware. Como tal se discute [48] pues corresponde justamente al desarrollo de una arquitectura escalable de la MobileNetV3 para implementación en FPGA. Esta se desarrolló utilizando Verilog y se describió en RTL para después ser sintetizada y evaluada en una FPGA.

Redes como la MobileNet ha sido modelada e implementada utilizando *high level synthesis* (HLS) [15]. Como tal se presenta una oportunidad en desarrollar un acelerador utilizando *HLS* considerando los objetivos de la biblioteca FAL para que esta pueda ser utilizada en el marco del proyecto. Se plantea que se puedan aprovechar sus características de eficiencia para su implementación en FPGAs de bajos recursos.

3. Metodología

La metodología para seguir es la conocida como “Ulrich Eppinger” (UE), que se presenta en el libro Diseño y Desarrollo de Productos [49]. En esta se sigue un proceso de diseño definido por múltiples etapas: planeación, identificación de las necesidades del cliente, establecimiento de especificaciones objetivo, generación conceptos de producto, selección de conceptos de producto, establecimiento de las especificaciones finales y planear el desarrollo descendente.

3.1 Planeación

Para la primera etapa se buscan los insumos del lado de las partes involucradas en el proyecto. De esta forma se puede sintetizar claramente el problema y así proceder con el resto de las etapas.

Dentro del contexto de un proyecto de investigación o de carácter académico que tiene como propósito generar conocimiento, los insumos resultan en la comprensión del estado de la investigación, el estado del arte, las tecnologías que se utilizan y las que se apunta a crear, cuáles son las limitaciones técnicas y económicas y una percepción del marco de tiempo.

Lo que se espera lograr con esta etapa es maximizar la efectividad del trabajo e identificar todas las oportunidades que puedan hacer del proyecto una contribución valiosa y significativa en su campo de estudio.

3.2 Identificación de necesidades

Esta etapa consiste en la identificación de las necesidades por parte del cliente, en este caso el asesor académico que dirige la investigación. Lo que se procura es que se identifique en el proyecto a desarrollar una serie de requerimientos para que se logre cumplir con los objetivos propios de la investigación.

Como tal, este proceso conlleva una serie de pasos que, dentro del contexto del proyecto, se adaptaron del proceso original propuesto por Ulrich-Eppinger para que se alcance a distinguir el objetivo de la investigación. Primeramente, como un paso cero, se debe tener entendimiento de la misión de la investigación y el camino que

se busca seguir para que esta sea una referencia destacada en el campo. Seguidamente, se procede con una recolección de datos del cliente, como tal una entrevista y un estudio de los trabajos previos para entender las oportunidades de nuevas implementaciones y mejoras.

Ahora bien, el siguiente paso justo después de identificar las necesidades, es asignar un grado de importancia dentro de una escala, la cual se predefinió del 1 al 3. Las necesidades estarán consolidadas en un documento que se entregará al cliente para futuras referencias dentro del marco de la investigación mientras que el grado de importancia quedará plasmado en una lista como la siguiente donde además se sintetice la necesidad para mayor efectividad.

Tabla 3.1. Interpretación de Necesidades

Tema	Necesidades Interpretadas	Importancia

3.3 Estado del arte

En este proyecto de investigación centrado en la creación de nuevo conocimiento, es fundamental reservar una fase inicial para llevar a cabo un análisis del estado actual en el campo. Dado que el propósito principal es generar este conocimiento, es importante señalar que actualmente no existen referencias sólidas que permitan discernir la superioridad de un modelo sobre otro en implementaciones en FPGA. Precisamente, la meta radica en desarrollar esta comprensión de modo que las personas puedan tomar decisiones informadas al elegir entre distintos modelos.

El estudio se basa principalmente en información teórica, y su validez se buscará demostrar mediante resultados experimentales. La elección de un modelo sobre otro se fundamentará en la teoría respaldada por investigaciones anteriores, tal como se presenta en papers, estudios previos y documentación de las herramientas

y librerías. En disciplinas como el aprendizaje automático y la visión por computadora, donde el progreso es constante, esta fase de revisión se vuelve crucial debido a la gran cantidad de información disponible que influye en el desarrollo en este campo.

Para que el trabajo presentado logre ser innovador y relevante en el campo es importante poder compararlo con los trabajos actualmente desarrollados. De esta forma también es posible apoyarse de los descubrimientos para generar mejores aportes. Se puede hacer la revisión de este estudio en la Sección 2.4. Estado del Arte.

3.4 Selección de Modelos

Durante esta etapa, se llevará a cabo la selección de conceptos que mejor se ajusten y aborden los criterios previamente establecidos, al centrarse en los detalles definidos en las secciones anteriores. Esto implica la creación de un marco comparativo que facilite la evaluación relativa de cada concepto y permita identificar oportunidades de mejora.

Los conceptos seleccionados se someterán a una comparación. Estos corresponden a sistemas previamente desarrollados y presentados en diversos trabajos académicos y papers. Este enfoque busca simplificar el proceso de discriminación y selección, ya que se puede recurrir a esta información como punto de partida para tomar decisiones al respecto. La referencia resulta importante para tener un buen criterio de evaluación.

En la Tabla 3.2 se presenta la propuesta para la evaluación de cada uno de los conceptos según los criterios definidos. Se realizará una clasificación de los conceptos a partir de los criterios de selección, pero con un método que proporcione un puntaje que dicte un tentativo ganador. Los pesos de cada criterio se determinarán a partir de la importancia relativa de las necesidades.

Tabla 3.2. Filtrado de los conceptos que se tomaron en cuenta.

Criterios de selección	Peso	Paradigmas			
		<u>Referencia</u>	Propuesta 1	Propuesta 2	Propuesta 3
1.	%	0			
2.	%	0			
3.	%	0			
4.	%	0			
5.	%	0			
Evaluación Ponderada					
Evaluación neta Lugar ¿Continuar?					

Una vez seleccionada una de las propuestas, se procede a desarrollar el modelo en base a los algoritmos presentes en el mismo, los avances expuestos y el estado del arte.

3.5 Establecimiento de Especificaciones

Una vez seleccionado una de las propuestas, se procede a abordar este desde un punto de vista más técnico para realizar una evaluación en términos de su aporte académico. Se realiza una traducción de las necesidades a un lenguaje de diseño con detalles precisos y medibles de lo que el sistema debe realizar es lo que se busca con esta sección. Ahora bien, que las especificaciones sean medibles es un aspecto sumamente importante, pues el sistema, en base a las necesidades, se evaluará a partir de las métricas. Para poder traducir desde lo que expresa el asesor (cliente) a esto, se debe tomar en cuenta un estudio del estado de la investigación, los avances que se han logrado dentro del proyecto y también avances fuera de este.

Ahora bien, sintetizando el proceso hasta ahora, se propone la tabla a continuación.

Tabla 3.3. Métricas definidas para cumplir con las necesidades encontradas.

Número	Número de Necesidad	Métrica	Importancia	Unidades	Valor Marginal	Valor Ideal
1						
2						
3						
4						

La importancia y la definición del valor ideal y marginal es también parte de lo que se logre interpretar de las necesidades, del grado de importancia establecido, y de las expectativas del proyecto en base al estudio realizado.

Esta es la metodología sobre la cual se evaluará el modelo, sin embargo, este se desarrolla como una etapa aparte que consiste en todo el diseño e implementación de este para posteriormente evaluarlo con esta tabla presentada.

3.6 Generación de conceptos

Dentro del marco del proyecto y el campo del desarrollo de hardware por FPGAs utilizando HLS, la generación de conceptos se basará en el uso de pragmas/ directivas/ optimizaciones. Tal como se expuso en la Sección 2.2.2.2, por medio de directivas, para un mismo modelo se pueden alcanzar una gran cantidad de propuestas con distintos alcances y objetivos que se tratarán como conceptos. Se considerará que para un mismo modelo se pueden hacer pequeñas modificaciones al algoritmo en caso de ser necesario, pero considerando que las optimizaciones apuntan al tratamiento de los datos, los ciclos de operación y establecimiento de protocolos.

A partir de la investigación del estado del arte, la interpretación de las necesidades, y el establecimiento de especificaciones, se plantean múltiples alternativas para lograr el desarrollo del sistema. Para esto es necesario hacer una nueva revisión a la definición del problema considerando todas las reflexiones de los procesos anteriores y realizar una subdivisión o sintetización en subproblemas.

Para resumir las aproximaciones dependiendo del subproblema que den origen a los conceptos se utilizará la tabla que se presenta a continuación.

Tabla 3.4. Definición de las características del concepto.

Subproblema	Solución Seleccionada

La tabla se empleará cuando sea pertinente en el proceso de diseño. Es relevante señalar que, debido a la naturaleza del problema, algunas decisiones serán más directas y no requerirán necesariamente esta tabla. Su presentación se propone como una opción adicional a considerar durante las distintas iteraciones del proceso.

Cada decisión tomada en el proceso de diseño puede dar lugar a nuevas alternativas que requieran iteraciones adicionales en el procedimiento. Se documentará este flujo de decisiones desde los conceptos más fundamentales hasta los subproblemas más específicos que puedan surgir a partir de ellos.

3.7 Desarrollo final

Por último, es fundamental realizar una selección basada en los criterios previamente establecidos, pero también planificar el desarrollo descendente y definir las especificaciones finales. Con esto se busca sintetizar todos los objetivos que el proyecto logró y su posición dentro del área de estudio.

Además de esta planificación, se llevará a cabo un estudio de las oportunidades de mejora que el proyecto puede ofrecer para futuros trabajos, el cual quedará consolidado en un documento detallado. Este análisis permitirá identificar áreas en las que se pueden realizar mejoras o expansiones, estableciendo así las bases para investigaciones posteriores. Paralelamente, se prestará especial atención a la documentación necesaria para el uso del proyecto.

Por último, se buscará activamente la publicación de los resultados y hallazgos obtenidos a lo largo del proyecto. Esto contribuirá a la difusión del conocimiento y permitirá que otros miembros de la comunidad científica se beneficien de los avances y conclusiones alcanzados.

3.7.1 Gestión de Costo del Proyecto

Este proyecto involucra procesos relacionados con planificar, estimar, presupuestar, financiar, gestionar y controlar los costos de modo que el proyecto pueda ser valorizado monetariamente. Se presenta como dos soluciones distintas, aquella como el costo del proyecto desarrollado de existir la necesidad de replicarse y aquella como la estimación de ahorros que se generaría de partir desde la existencia de este. Además, se considera una tercera referencia en base al uso de soluciones con otras tecnologías.

Se considera este como un proyecto de carácter académico por lo que existen detalles propios de proyectos de esta naturaleza que no se alinean a un proyecto industrial o empresarial. La mayor parte de del proyecto se llevó a cabo en horas de investigación y desarrollo, lo que se conoce como costo de ingeniería.

El desglose de costos y análisis financiero se presentan como parte de la sección de resultados.

4. Desarrollo de la solución

En el presente capítulo se detalla cómo se ejecutó el proyecto a partir de la metodología seleccionada. Según las etapas previamente descritas, este capítulo busca profundizar en los principales factores que se tomaron en cuenta, así como evidenciar el proceso de decisión para los distintos problemas que este trabajo presentó.

Es importante mencionar que lo aquí presentado se plantea a partir de la determinación del problema dentro de la fase de interpretación de necesidades, para lo cual se utilizaron varios insumos principalmente por parte de declaraciones del asesor técnico MHPC. Luis León Vega. A partir de esta etapa se concluyó en la síntesis del problema descrita en la sección 1.3 de este documento que identifica la necesidad de optimizar la operación de convolución en el marco del trabajo de investigación realizado por el laboratorio.

Una vez que el problema a resolver se definió con claridad a partir de las necesidades, se procedió con las demás etapas de la metodología establecida.

4.1 Interpretación de las necesidades

Se parte del hecho de que el proyecto busca el desarrollo de aceleradores para realizar tareas de visión por computador. Esto a partir de la oportunidad que existía con el área de Mecatrónica y los aportes que esta puede brindar en conocimiento.

A partir de esto, se determinó que respecto a las distintas operaciones que requerían optimizaciones, la convolución, que es propia de las redes neuronales convolucionales y la visión por computadora, requería desarrollarse. Para esto después de un proceso de entrevista, las necesidades interpretadas se resumen en los siguientes puntos:

- Se requiere un acelerador que ejecute la operación de convolución con tiempo menor que la convolución estándar.
- Se requiere evaluar que el acelerador ejecute la operación de convolución con un menor consumo de recursos respecto a la convolución estándar.

- Se requiere que el acelerador sea integrable a FAL (Sección 4.1.2) y compatible con la estructura del sistema de construcción. Se requiere que cumpla con un check-list en cuanto a los requerimientos propios de integración a FAL.
- Se requiere que el acelerador permita la integración con distintos sistemas para el procesamiento de imágenes por visión por computador.
- Se requiere que la precisión del modelo se encuentre al nivel de los modelos de la academia.

A partir de esta lista en crudo, se procedió a esquematizarla según la Tabla 3.1, y seguido de una reunión se le asignó una importancia en un rango del 1 al 3 tal como se describió anteriormente.

Tabla 4.1. Necesidades interpretadas para la realización del acelerador de convolución.

	Tema	Necesidades Interpretadas	Importancia
1	Tiempo de Ejecución	El valor de latencia debe ser menor al de la convolución estándar.	3
2	Cantidad de Recursos	Número de LUTs, DSPs, Flip-flops y BRAM debe ser menor a la convolución estándar	3
3	Desarrollo en HLS	El ambiente de desarrollo debe de realizarse a partir del paradigma de descripción de hardware de High Level Synthesis	3
4	Precisión del Modelo	El tradeoff de la precisión vs eficiencia del modelo no debe ser significativa para tareas de clasificación en distintos contextos.	2
5	Flexibilidad del Modelo	El modelo debe soportar distintas modificaciones sobre el manejo de los datos a utilizar que permitan	2

		hacer pruebas en distintas aplicaciones.	
6	Capacidad de Integración	El acelerador permite una integración fluida con FAL para su escalabilidad en distintas aplicaciones.	3

En relación con la necesidad 4, es importante tomar en cuenta que esta define uno de los indicadores sobre los cuales se evaluará el proyecto, para el cual después del estudio respectivo se determinó el valor específico que se debe alcanzar.

Los trabajos presentados para redes como la MobileNet [7], Xception [50], y la EfficientNet [51] presentan resultados alrededor del 80% para sets de datos de ImageNet. Estos se tratan de resultados que se trabajan en ambientes sumamente controlados los cuales resultan de una gran cantidad de pruebas y ajustes para lograr alcanzar. Tomando el criterio del asesor técnico Luis León se tomó la decisión de apuntar a resultados menores a estos por motivos de tiempos en el proyecto. De esta manera se definió que el valor a apuntar para el tipo de convolución que se desarrolle y la red que se implemente corresponda a algo mayor a 75%.

4.1.1 Requisitos mínimos de hardware y software

Previo a comenzar el desarrollo es importante mencionar los requerimientos básicos que se debe poseer a nivel de hardware y software para poder desarrollar el proyecto de este informe.

Computadora con los siguientes requisitos mínimos:

Procesador: Se recomienda un procesador Intel o AMD de múltiples núcleos con una velocidad de reloj de al menos 2 GHz.

Memoria: Se recomienda al menos 8 GB de memoria RAM.

Espacio en disco: Se recomienda un espacio de disco duro de al menos 20 GB para la instalación completa de Vivado HLS y sus componentes adicionales.

Tarjeta gráfica: Se recomienda una tarjeta gráfica con capacidad para resoluciones de pantalla de al menos 1024x768 píxeles.

Estos requisitos responden al uso del software de Vivado HLS 2018.2 [52]. Mayor detalle en recomendaciones se pueden consultar en las fuentes de Xilinx.

Es importante adicionar que, aunque el sistema operativo requerido para Vivado HLS resulte flexible entre Windows y Linux, es imperativo el uso de Linux, dado que el sistema de FAL fue construido en este y actualmente añade un alto grado de complejidad llevar este al ambiente de desarrollo a Windows.

4.1.2 Estudio de los requisitos para integración en FAL

Para esto se procedió a realizar una reunión con el asesor técnico del proyecto acerca de los requerimientos para que este proyecto fuera de utilidad dentro de su estudio y trabajo en FAL. De esta forma el trabajo realizado podría integrarse fluidamente.

Los principales requerimientos tomando como referencia tanto los objetivos de la librería a nivel de investigación, criterio creado a partir del estudio del estado de la investigación y la retroalimentación del MHPC. Luis León se resumen a continuación en donde la unidad de procesamiento debe poseer las siguientes características:

1. Código Sintetizable

- Que el código pueda ser convertido o sintetizado en circuitos electrónicos reales.
 - Código en C++ Sintetizable:
Esto hace referencia a que no todo el código de C++ se puede convertir o traducir hardware. Es decir que se debe utilizar una versión o subconjunto de C++ que no utilice características complejas que no se puedan utilizar en HLS.
 - Evitar Operaciones No Sintetizables:
Algunas operaciones y bibliotecas de C++ no son sintetizables, como el manejo dinámico de memoria (por ejemplo, malloc),

operaciones de punteros complejas, operaciones de archivos, y funciones no admitidas por la herramienta de HLS.

- **Limitaciones de Recursos:**

Las herramientas de HLS tienen limitaciones en términos de recursos disponibles, como registros y unidades de procesamiento. Se deben tener en cuenta estas limitaciones y ajustar el código en consecuencia. Parte de los objetivos de FAL es trabajar en FPGAs de bajo perfil, las cuales por definición tienen menos recursos.

2. Compatible con la estructura del sistema de construcción de FAL.

- Que las características, funcionalidades o requisitos del proyecto pueden ser implementados utilizando las capacidades y componentes proporcionados por el framework de FAL a partir de su sistema de construcción en el lenguaje “meson”.
- Además, que el proyecto pueda ser utilizado dentro del framework para futuras implementaciones.
- El framework de FAL proporciona una serie de abstracciones y soluciones listas para usar que permiten desarrollar más rápidamente, por lo que en caso de requerirse que estas puedan apoyar el desarrollo.
- Que la implementación tenga una estructura flexible que permita modificar los parámetros para realizar pruebas e implementaciones varias.

3. Simulable con el AxC Executer para simular redes neuronales profundas

- Se busca que el diseño sea integrable con el repositorio del AxC Executer de FAL para la simulación de la unidad dentro de una estructura en software.

- La idea es que sea posible construir módulos de ejecución y permitir la personalización de estos módulos para realizar cálculos específicos de manera eficiente. De esta manera que sea posible evaluar las combinaciones posibles para el mejor funcionamiento del acelerador.

4. Estructura flexible: resolución numérica, reemplazabilidad de operadores, ancho de bus personalizable.

- Se está haciendo referencia a la capacidad del código para adaptarse y configurarse de acuerdo con diferentes requisitos y características del hardware objetivo. Esto es especialmente importante en el contexto de HLS, donde el mismo código de C++ puede ser sintetizado en hardware personalizado para diferentes plataformas y configuraciones.
 - Resolución numérica:
Esto podría incluir la posibilidad de cambiar el número de bits utilizados para representar valores enteros o fraccionales.
 - Reemplazabilidad de operadores:
Que el código esté diseñado de manera modular, de modo que los operadores y funciones utilizados pueden ser reemplazados o intercambiados fácilmente sin afectar significativamente el funcionamiento global.
 - Ancho de bus personalizable:
Refleja la capacidad de ajustar el ancho del bus de datos utilizado para la comunicación entre diferentes módulos o componentes en el hardware sintetizado.

5. Compatible con PYNQ para uso en FPGAs con MPSoC

- PYNQ es un proyecto de código abierto que proporciona una plataforma y una metodología para el diseño de sistemas embebidos basados en FPGA en computación adaptativa.

- PYNQ permite a los desarrolladores diseñar hardware personalizado en lenguajes de descripción de hardware como VHDL o Verilog y luego integrarlo con facilidad en aplicaciones basadas en Python. Esto facilita la creación de aceleradores de hardware específicos para aplicaciones. La idea es que el hardware desarrollado se pueda ver beneficiado por esta característica.

4.2 Definición de Especificaciones

Para poder iniciar con esta etapa, se deben traducir las necesidades a métricas que permitan concluir si se cumple o no lo solicitado. Estas definen de forma directa el grado al cual el producto satisface las necesidades. Por la naturaleza del problema, muchas de estas métricas se cuantifican a partir de la misma plataforma de desarrollo, sin embargo, todas aquellas con un grado de nivel técnico alto se explican en la sección 2 del Marco Teórico. Además, a partir de la interpretación de las necesidades y la continua comunicación con el asesor Luis León se establecieron rangos y valores metas sobre los cuales calificar si se alcanzaban las necesidades encontradas. El principal insumo para definir las que se tomó en consideración con el asesor, fueron los rendimientos y valores alcanzados por las aproximaciones y trabajos anteriores en FAL. De esta forma un caso marginal resulta en obtener algo similar a lo que ya se tenía y lo ideal sería obtener algo superior a ello.

Las métricas también consideran investigación que se realizó previamente. Otro detalle importante que tomar en cuenta es que para cualquier concepto se debe procurar tener condiciones similares a evaluar.

Tabla 4.2. Métricas de evaluación para conceptos.

	Número de Necesidad	Métrica	Importancia	Unidades	Valor Marginal	Valor Ideal
1	1	Latencia	3	ciclos/píxel	[25,32]	<25
2	2	Consumo de LUTs	3	% utilización	[15,20]	<15
3	2	Consumo de DSP48E	2	% utilización	[3, 4]	<3
4	2	Consumo de BRAM_18K	2	% utilización	[15, 30]	<15
5	2	Consumo de Flip-flops	3	% utilización	[11,20]	<11
6	3	Compilable en C++	3	Binario	True	True
7	3	Sintetizable a RTL	3	Binario	True	True
8	3	Cosimulable	3	Binario	True	True
9	3	Empaquetable a FPGA	3	Binario	True	True
10	4	Precisión del Modelo	2	% de precisión	[75,85]	[85,100]
11	5	Tamaño de Bus	2	bits [rango]	[4, 64]	[4, 512]
12	5	Tamaño de Dato	3	bits [rango]	[4, 8]	[4, 16]
13	5	Tamaño de imagen	2	píxel	512x512	1024x1024
14	5	Tamaño Máximo de Kernel	2	valor	3x3 dw 1x1x3 pw	5x5 dw 1x1x16 pw
15	5	Máximo Stride	2	valor	1	2

4.3 Selección de conceptos

Esta sección se basa en la investigación interna y externa realizada y consolidada en la Sección 2 del Marco Teórico. En esta se presentaron los distintos modelos de convolución que se trabajan a nivel académico y de la industria de tecnología para la optimización de esta operación.

La forma en la que el proceso de diseño se llevó a cabo correspondió en primer lugar a la selección del modelo de convolución óptimo que según el estudio del estado del arte y las necesidades del proyecto presentara la mayor oportunidad de

ser implementado y útil en el marco de la investigación. Los 4 conceptos que se evaluaron se resumen en la siguiente tabla:

Tabla 4.3. Algoritmos de convolución evaluados para su desarrollo en FAL

Criterios de selección	Peso	Paradigmas			
		Referencia Conv. Estándar	Convolución Separable	Convolución FFT	Convolución Winograd
Eficiencia computacional	25%	3	5	5	5
Flexibilidad y adaptabilidad.	15%	5	4	3	2
Necesidad en FAL	40%	1	5	2	4
Precisión	10%	5	3	5	4
Uso de Memoria	10%	3	5	4	4
Evaluación Ponderada		0,15 + 0,15 + 0,08 + 0,10 + 0,06	0,25 + 0,12 + 0,40 + 0,06 + 0,10	0,25 + 0,09 + 0,16 + 0,1 + 0,08	0,25 + 0,06 + 0,32 + 0,08 + 0,08
Evaluación neta Lugar ¿Continuar?		0,54 4 No	0,93 1 Sí	0,68 3 No	0,79 2 No

Los 4 algoritmos evaluados se abordaron conceptualmente en el marco teórico. Como tal la que presentó una mejor calificación corresponde a la convolución separable. En esta destaca la necesidad que existe de implementar este algoritmo en FAL, criterio que tomó el mayor peso. Esta resulta en un área inexplorada y fácilmente evaluable debido a su característica principal en la disminución de las operaciones respecto a la convolución estándar. Es un tipo de convolución bastante novedosa que, además de su eficiencia computacional, puede actuar como un tipo de regularización, lo que previene overfitting en modelos más profundos.

En cuanto a su implementación, se tomaron en cuenta distintas referencias en donde se utilizó este tipo de convolución para armar redes neuronales complejas que manejaran este tipo de convolución y se aprovecharan de sus características. De esta forma se encontraron distintos modelos de redes para los cuales se podrían realizar pruebas a nivel de investigación a futuro. Para cuestiones del proyecto en

curso y realizar las pruebas necesarias, se debe seleccionar una para la realización de pruebas. De esta manera, dentro de los requerimientos de FAL, y una vez seleccionado el tipo de convolución a desarrollar, fue necesario agregar una última viñeta a las necesidades de FAL descrita a continuación:

- **Compatibilidad con redes que usen convolución separable**
 - La convolución separable en profundidad es un modelo de convolución utilizado en varios modelos de redes. El más común es MobileNet sin embargo su aplicación va más allá. Este debe funcionar de forma tal que sea útil para la construcción de distintas redes tales como:
 - MobileNet
 - Xception
 - EfficientNet
 - ShuffleNet
 - Inception-v4
 - Estas últimas son redes que juegan con este algoritmo dentro de sus capas para alcanzar mejores resultados.

Nótese que modelos de convolución como la convolución por transformada rápida de Fourier (FFT) o winograd, presentan características sólidas que evaluadas bajo el mismo criterio resultan en opciones viables para abordar el problema. Sin embargo, después del estudio presentado al asesor, se decidió que estas no representaban una necesidad mayor dentro de FAL pues existían ya trabajos previos en la biblioteca que aproximaban los resultados que se esperaban obtener. Incluso la misma convolución estándar, utilizada como referencia, presenta características positivas, sin embargo, es la necesidad en FAL la que predomina como criterio.

4.4 Codificación del algoritmo de convolución separable en software.

Siguiendo el proceso de diseño, lo primero que se hizo una vez realizado la investigación respectiva del algoritmo fue realizar un modelo y prototipo a nivel de

software. Esto consistió en una prueba inicial de concepto para un mejor entendimiento del funcionamiento del algoritmo y una inicial familiarización con el ambiente de desarrollo. Para esto se consideraron distintas opciones que pudieran agregar valor a nivel de la fase de validación o comprensión de los modelos.

4.4.1 Prototipo de algoritmo de convolución en C++

Como se mencionó previamente, el proyecto se desarrollará a partir del paradigma de descripción de hardware de HLS tal como se describe dentro de las necesidades y requerimientos. Este resulta en una opción con facilidad de programación, mayor velocidad en el proceso de desarrollo, reutilización del código y alta portabilidad hacia distintas arquitecturas. Estas características ventajosas se deben principalmente a que este se programa en C++ por lo que la traducción de software a hardware resulta más fluida. Aunque dicha traducción no es un mapeo 1 a 1, sino que se deben ajustar algunos detalles propios de la plataforma de desarrollo, resulta en una excelente aproximación.

Lo que se realizó según este proceso fue el trabajo en los algoritmos de convolución separable específicamente. Se realizó un código que fuera capaz de tomar una imagen de varios canales, ejecutar una convolución depthwise a cada canal y una vez esto se realizara, se ejecutara una convolución pointwise.

Esta prueba consistió en trabajos sobre la conocida imagen de “Lenna”, donde se ejecutaron operaciones de padding, se consideró stride y un manejo de las dimensiones en base a este último y tamaños de kernel que se quisiera utilizar.

A partir de este desarrollo se evidenció la necesidad de una librería en C++ especializada en el manejo y procesamiento de imágenes por lo que se trabajó con la librería de opencv. Fue posible además entender la importancia del manejo de los tipos de datos que se utilizaban para hacer más eficiente el algoritmo a nivel de implementación de hardware. Esto condujo a una aproximación inicial para el hardware del tamaño que se iba a abordar.

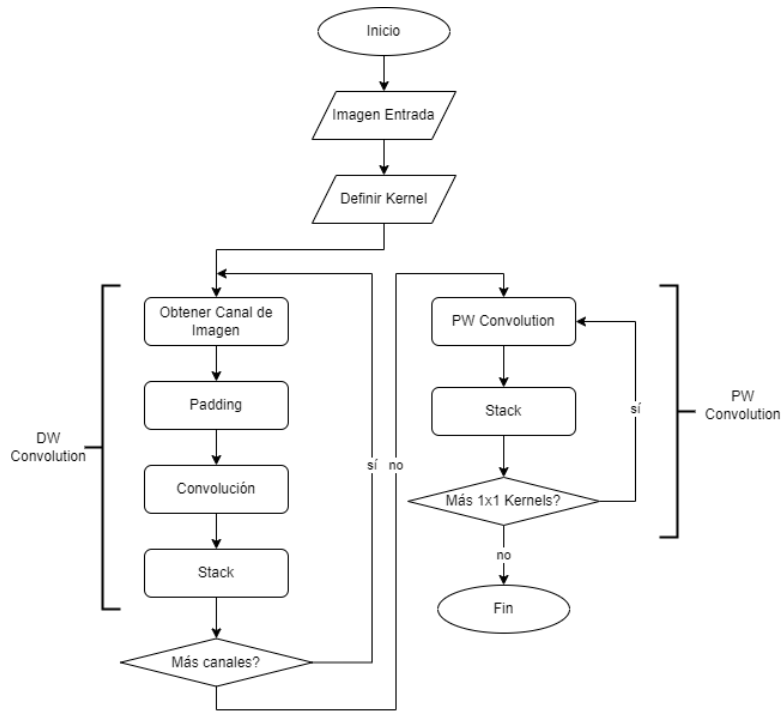


Figura 4.1. Diagrama de flujo de algoritmo de convolución separable en C++.

Durante esta etapa, se logra comprender el flujo general del algoritmo y se identifican las variables necesarias para el sistema. Es fundamental considerar el padding de la imagen original, de acuerdo con el tamaño del kernel, para garantizar la coincidencia de las dimensiones de salida. En caso de que haya stride o no-padding, se debe calcular con precisión las dimensiones de salida para evitar la pérdida de datos.

Inicialmente se creó un script en Python a muy alto nivel que facilita la comprensión del algoritmo y agiliza su posterior traducción a C++. También se evaluó el código que ejecuta los distintos módulos de la convolución separable en profundidad y produce una imagen final. Esta imagen se comparó con el mismo proceso realizado mediante librerías como OpenCV, las cuales más adelante se utilizarán para evaluar el algoritmo en hardware. A continuación, se presentan los resultados de esta primera fase en Python.



Figura 4.2. Comparación de resultados entre procesamiento de imagen mediante algoritmo personalizado vs librerías estándar.

El script utilizado para este se puede consultar en la sección de Anexos, y como tal realiza una evaluación donde compara las salidas de ambas implementaciones. Como se observa por los resultados, en el caso de esta primera aproximación en Python, las imágenes procesadas resultan exactamente iguales.

Seguido de la comprensión de los algoritmos en alto nivel con Python se procede a hacer el mismo ejercicio con C++. El resultado de este código se trasladó al mismo ambiente de Python donde la imagen resultante se comparase nuevamente con la aproximación de los algoritmos propios de las librerías Python. Los resultados nuevamente se obtuvieron tal como se obtuvieron en la Figura 4.2. Estos algoritmos en software resultaron ser el insumo sobre el cual se comparó el AxC Executer en la sección 5.5 con respecto al hardware desarrollado.

4.4.2 Prototipo de red neuronal convolucional en Python

Python es un lenguaje de alto nivel ampliamente empleado en el campo de la inteligencia artificial y el aprendizaje automático, así como para la creación de pequeños scripts de validación y la exploración de diversos conceptos de programación. Por esta razón, se considera una opción válida dentro del proyecto para llevar a cabo pruebas y experimentaciones.

Python cuenta con varias librerías que hacen el proceso de desarrollo más eficiente. Dentro del ámbito del machine learning destacan las librerías de Tensorflow y Pytorch que cuentan con una serie de modelos pre-programados y documentaciones para implementarlos. Modelos como los previamente

presentados de MobileNet, Xception, EfficientNet se encuentran dentro de estas librerías para utilizarse en distintas aplicaciones.

Una primera aproximación de desarrollo fue la familiarización con estos modelos, para a nivel práctico entender los resultados que estos podían retornar. Para esto, se generó un código que fuera capaz de tomar una serie de datos, entrenarlos a partir de la estructura del modelo y retornar valores de pérdidas y precisión.

La forma en que se programó dicho software consistió en la construcción de la estructura; que se define según las librerías, y después a partir de un set de datos se entrenó y validó el modelo. El modelo consiste en un backbone generado a partir de alguno de los modelos previamente mencionados y una red neuronal densa.

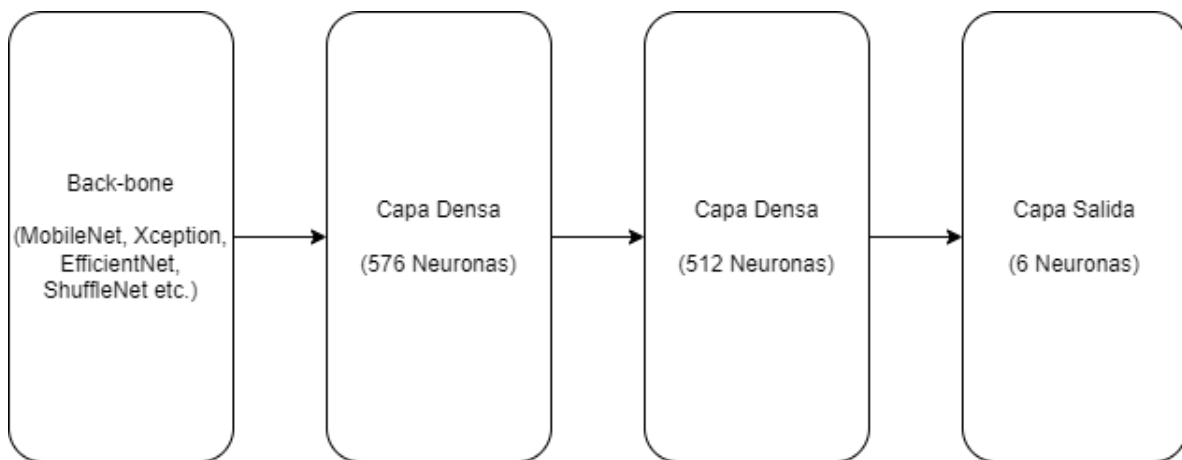


Figura 4.3. Estructura base del prototipo de Python de la Red Neuronal Convolutiva.

Como tal, se entrenó y probó el modelo en base a un set de datos que reuniera las características que la librería considera favorables en términos de dimensiones. Para esto se decidió utilizar el set de datos de Intel Image Classification proveído por la plataforma Kaggle, que consiste en imágenes de escenas naturales alrededor del mundo. Se debe considerar que al tratarse de un prototipo a este no se le realizó ninguna tarea de ajuste fino ni ajuste en los pesos del back-bone por lo que resulta en un software sumamente básico para pruebas.

Ahora bien, se debe seleccionar un backbone para tener un marco comparativo. De esta forma desde el AxC Executer se procede a desarrollar dicha estructura de red

y se realizan pruebas de esa misma red utilizando las librerías de Python. Para esto, y siguiendo con la metodología propuesta, la decisión se basa en la Tabla 4.4 que favorece la comparación de conceptos, modelos, algoritmos etc.

Tabla 4.4. Modelos de convolución que emplean convolución separable evaluados para su desarrollo en software y hardware.

Criterios de selección	Peso	Paradigmas			
		Referencia Red Convolución Estándar	MobileNet	Xception	EfficientNet
Eficiencia computacional	30%	0	5	4	3
Flexibilidad y adaptabilidad.	15%	5	4	4	3
Innovación	10%	0	5	3	5
Precisión	25%	4	3	4	5
Facilidad de programación	20%	5	3	3	3
Evaluación Ponderada		0,00+ 0,15 + 0,00 + 0,20 + 0,20	0,30 + 0,12 + 0,10 + 0,15 + 0,12	0,24 + 0,12 + 0,06 + 0,20 + 0,12	0,18 + 0,09 + 0,10 + 0,25 + 0,12
Evaluación neta		0,55	0,79	0,74	0,74
Lugar		4	1	2	2
¿Continuar?		No	Sí	No	No

Se procede a continuar con la red MobileNet, para la cual un estudio de esta a detalle se realiza y se incluye en el Marco Teórico. Esta se presenta como una mejor alternativa sobre todo por la eficiencia computacional. El modelo de MobileNet más allá de utilizar convolución separable, que es el común denominador entre los modelos evaluados, reduce la cantidad de operaciones y memoria necesarias al utilizar las capas de cuello de botella residuales invertidas, lo que permite una implementación particularmente eficiente en memoria. Está diseñada para ambientes con menos recursos por lo que además en el contexto del actual proyecto que busca trabajar con FPGAs de bajo perfil, resulta útil.

El modelo de la MobileNet, así como muchos otros, se encuentran programados en las librerías de ML de Python como Keras y Pytorch. De esta forma por medio de una función sencilla se invoca este modelo para correr pruebas y obtener un valor de precisión inicial sobre el que partir para un set de datos. Evaluando para ImageNet se alcanza en Python una precisión de entre 88% y 91% con 3000 imágenes de entrenamiento y 64 batches. Resultados gráficos se presentan en la siguiente figura.

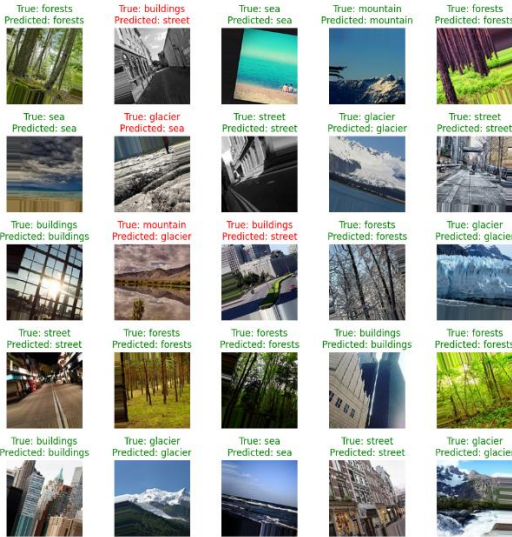


Figura 4.4. Predicciones realizadas para el modelo de la MobileNet con el set de datos de Intel Image Classification.

4.5 Planteamiento y Diseño del Modelo a nivel modular

La siguiente etapa del desarrollo consistió en el diseño modular de cómo se implementarían en hardware los algoritmos previamente prototipados en software. Una vez entendidos los requisitos y las especificaciones proporcionadas por el framework de FAL, es necesario establecer detalles adicionales propios de la implementación que se desea desarrollar. Estos detalles abarcan las entradas, salidas, protocolos, la FPGA a utilizar, el manejo de datos, las expectativas de rendimiento y consumo energético, detalles de escalabilidad y flexibilidad, y en general, todo lo que se pueda considerar para que la unidad soporte distintas aplicaciones.

Ahora bien, antes de entrar en la definición de estas variables, es importante considerar un análisis e investigación preliminar para ver distintos enfoques y soluciones utilizadas previamente. Con el objetivo de evaluar la viabilidad técnica y la idoneidad de un diseño sobre otro considerando el marco de tiempo y conocimiento sobre el cual se trabaja. Esta investigación se trató principalmente a partir del marco teórico previamente expuesto y una serie de conceptos adicionales que se tratarán más adelante. Esta investigación se hace también con el fin de obtener insumos para desarrollar el criterio de diseño para definir las especificaciones previamente mencionadas.

En la tabla a continuación se resumen algunas de las consideraciones que se tuvieron para el diseño de la unidad.

Tabla 4.5. Consideraciones base para el desarrollo de la arquitectura.

	Justificación
Entradas	Considera los datos que la unidad de hardware recibirá y procesará. Se debe definir qué datos de entrada, la frecuencia de las entradas y la fuente de los datos, junto con el formato de datos compatibles.
Salidas	Considera qué resultados se esperan de la unidad de hardware y cómo se utilizarán estos resultados en el sistema más amplio. Se debe definir los formatos de salida de datos, la frecuencia de las salidas y la compatibilidad con otros componentes del sistema.
Protocolos	Considera la comunicación y la transferencia de datos, la facilidad de implementación y la interoperabilidad.
FPGA	En este caso, la selección se basa en la disponibilidad y particularidad del proyecto, que apunta a sistemas de bajo perfil.

En base a este diseño y considerando la operación de convolución se procede a desarrollar un modelo general a nivel de bloques para el acelerador. En este se muestran entradas, salidas y protocolos mientras que la FPGA a utilizar consiste en

el modelo xc7a50tcs325-2 que corresponde a una FPGA Artix-7 de disponibilidad en el laboratorio.

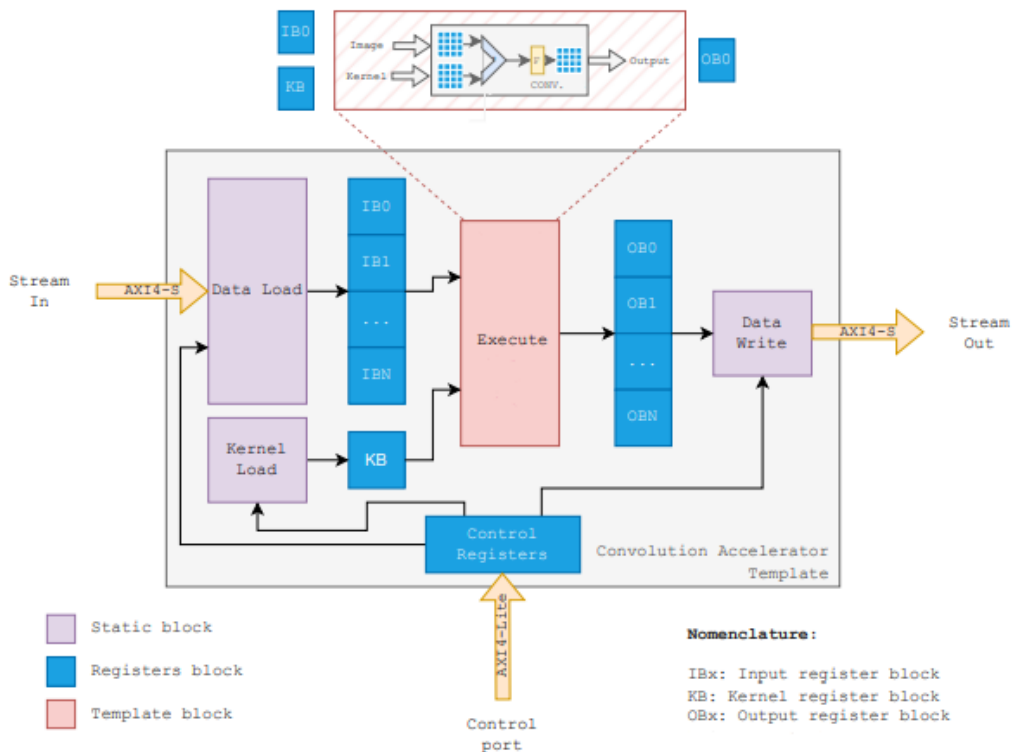


Figura 4.5. Diseño modular del acelerador de convolución.

El diseño asume una convolución separable, que ejecuta dos procesos de convolución, tanto depthwise como pointwise. Se contemplaron dos casos: uno en el que se tratase un único acelerador con la capacidad de ejecutar la convolución depthwise y la convolución pointwise desde una misma arquitectura única, o bien dos aceleradores distintos que ejecuten cada uno un tipo de convolución, es decir dos arquitecturas. La primera opción reduce la flexibilidad ya que, en varios modelos, existen operaciones entre una convolución y otra, además resulta más complejo a nivel de las herramientas de desarrollo hacer una única arquitectura, por lo que se decidió crear dos aceleradores basados en la misma lógica pero que trabajan cada uno en su caso específico.

La arquitectura planteada en la Figura 4.5 es un caso general. La forma que ambos aceleradores se van a desarrollar en hardware es igual en cuanto al flujo de datos.

Para ambos, se consideraron aspectos iguales, por lo que en el presente informe se hará la distinción según se requiera. Primeramente y tal como se mostró en la figura mencionada, se consideraron 4 módulos principales para ambos aceleradores:

Data Load: Recibe el flujo de entrada y almacena la imagen en un registro de respaldo.

Kernel Load: Recibe el kernel y lo almacena en un banco de registros.

Execute: Cuya implementación está personalizada y basada en un modo de operación particular. Este corresponde a la ejecución del algoritmo de convolución según corresponda.

Data Write: Lee el banco de registros de salida para enviar.

Los módulos de Data Load, Kernel Load, y Data Write representan bloques programados de igual forma en HLS con su propia lógica y desarrollo. Estos se profundizan en pseudocódigo en los anexos. El módulo de Execute se abarca a detalle en esta sección pues representa la lógica central del acelerador.

Existen distintas aproximaciones a nivel de descripción de hardware por high-level synthesis para abordar la convolución (e incluso los otros bloques). Esto supone considerar nuevamente ciertos parámetros para el diseño del sistema los cuales se resumen tal como en el caso anterior en la siguiente tabla:

Tabla 4.6. Consideraciones adicionales para el desarrollo de la arquitectura.

	Justificación
Manejo de datos	Considera cómo se almacenan, procesan y manipulan los datos dentro de la unidad de hardware. Esto permite revisar técnicas y estrategias específicas utilizadas para optimizar el rendimiento y la eficiencia del manejo de datos.
Escalabilidad y Flexibilidad	Considera cómo puede adaptarse y crecer para manejar mayores cargas de trabajo, volúmenes de datos más grandes y requisitos más complejos para

	distintas aplicaciones. Considera cómo la unidad de hardware puede ampliarse sin comprometer su rendimiento y funcionalidad básicos.
--	--

Se consideran dos distintas opciones para desarrollar el algoritmo de convolución en HLS a partir de las formas en las que se pueden manejar los datos, primero trabajarlo a nivel de line buffers y windows o bien trabajarlo como una convolución directa. Estas dos se comparan a continuación utilizando el mismo marco comparativo expuesto en la sección 3.

Tabla 4.7. Algoritmos de convolución evaluados para su desarrollo en FAL.

Criterios de selección	Peso	Paradigmas	
		Convolución Directa [arrays]	Convolución con Line Buffers
Uso de Memoria	20%	3	5
Uso de Recursos en FPGA	25%	3	4
Rendimiento	25%	3	4
Complejidad	30%	5	4
Evaluación Ponderada		0,12 + 0,15 + 0,15 + 0,30	0,20 + 0,20 + 0,20 + 0,24
Evaluación neta ¿Continuar?		0,72 No	0,84 Sí

La implementación directa de convolución [53] se destaca por su simplicidad en el proceso, no requiriendo buffers adicionales o técnicas avanzadas de almacenamiento. Sin embargo, su desventaja principal es la menor eficiencia en el uso de recursos, ya que puede demandar más recursos de hardware y memoria para almacenar los datos de entrada, el kernel y los resultados intermedios de la convolución. Además, su rendimiento suele ser más lento en comparación con otras técnicas debido a la falta de técnicas avanzadas de almacenamiento en búfer y procesamiento optimizado.

Por otro lado, al profundizar en la convolución con buffers de línea [41] que es la técnica de programación que mediante la aplicación de la metodología resultó ganadora, se destaca que esta presenta:

- Uso eficiente de recursos: La implementación con line buffers y window buffers utiliza estos buffers especializados para almacenar y procesar de manera eficiente los datos de entrada, lo que reduce el consumo de recursos y la complejidad del diseño.
- Alto rendimiento: Al aprovechar técnicas de almacenamiento en búfer y procesamiento optimizado, esta implementación puede lograr un rendimiento significativamente más rápido en comparación con la implementación directa de convolución.
- Mayor complejidad en el diseño: La utilización de buffers de línea y ventana agrega complejidad al diseño y puede requerir un mayor esfuerzo de programación y depuración en comparación con la implementación directa de convolución.

Finalmente, esta decisión se traduce en el tiempo dedicado al desarrollo de la práctica más eficiente. Dadas las circunstancias del proyecto, se procede al desarrollo de la convolución utilizando line buffers.

Acorde a los requerimientos establecidos en la sección 4.1.2, el número 4 corresponde a la necesidad de que el modelo pudiera tener una estructura flexible, con el fin de desarrollar Design Space Exploration. Este apunta a identificar la configuración óptima de síntesis para un diseño dado [54]. De esta forma una de las consideraciones a tomar en cuenta es que el diseño deba ser modificable en cuanto a resolución numérica, reemplazabilidad de operadores, ancho de bus y datos personalizable, lo cual limita el diseño de la unidad para cumplir con esta condición.

A nivel de pseudo-código, y de la mano con los desarrollos realizados en software, se busca traducir a hardware, de la manera más eficiente tomando en cuenta las limitaciones de recursos, energía y tiempo, los siguientes algoritmos:

Algoritmo 1 Convolución Depthwise

```
1: input: image, kernel
2: output: convolved image
3:
4:   for each row in the output matrix:
5:     for each column in the output matrix:
6:       sum = 0
7:       for each kernel row:
8:         for each kernel column:
9:           sum += input[row+kernel_row][col+kernel_col] * kernel[kernel_row][kernel_col]
10:
11:        output[row][col] = sum
12:   return output
13: End Function
```

Algoritmo 2 Convolución Pointwise

```
1: input: image, kernel ▷ mismo número canales
2: output: convolved image
3:
4:   for each row in the output matrix:
5:     for each column in the output matrix:
6:       sum = 0
7:       for each channel in the input matrix:
8:         sum += input_image[row][col][ch] * kernel[ch]
9:       output[row][col] = sum
10:  return output
11: End Function
```

Es importante dejar claro que, a nivel de codificación y desarrollo, el proyecto abarca la implementación únicamente de la unidad de procesamiento que realiza la operación de convolución dentro de la red neuronal convolucional.

4.6 Diseño en HLS

Esta sección busca sintetizar e ilustrar la forma en que se llevó a cabo el proceso de desarrollo en la herramienta de Vivado HLS 2018.2.

4.6.1 Codificación base

La idea central de esta codificación, como se ha mencionado previamente, es trabajar en un código basado en el lenguaje de programación de C o variantes de este como C++ o SystemC para poder obtener un RTL que posteriormente pueda ser sintetizado en una FPGA a través de las herramientas de Xilinx adecuadas.

En HLS, cada uno de los módulos propuestos se trabaja como una función separada en C++. Como tal se debe desarrollar un módulo de lectura de datos y uno de lectura del kernel, uno de ejecución de la convolución, es decir que trabaje los datos y ejecute el algoritmo, y por último uno que reciba estos datos y los escriba de vuelta.

Como se mencionó previamente, la forma en la que se diseñó el algoritmo para un uso más eficiente de la memoria es a través de line buffers, de esta forma el algoritmo de convolución, que si bien conserva su estructura base se trabaja de la siguiente forma.

Algoritmo 3 Convolución con Line Buffers y Windows

```
1: input: image, kernel
2: output: convolved image
3:
4: line_buffer = array of kernel size × image columns
5: window = kernel_size
6: for each row in the output matrix:
7:     for each column in the output matrix:
8:         line_buffer -> insert [row][column]
9:         if line_buffer is filled:
10:            sum = 0
11:            for each kernel row:
12:                for each kernel column:
13:                    result=line_buffer[row+kernel_row][col+kernel_col] * kernel[kernel_row][kernel_col]
14:                    window -> insert result
15:                    sum(elements in window)
16:                    output[row][col] = sum
17:            return output
18: End Function
```

Ahora bien, ambos casos ejecutan la convolución con un grado de diferencia a denotar. Esto se ilustra con los siguientes dos gráficos que muestran cómo se realiza el manejo de datos dentro del algoritmo para depthwise y para pointwise.

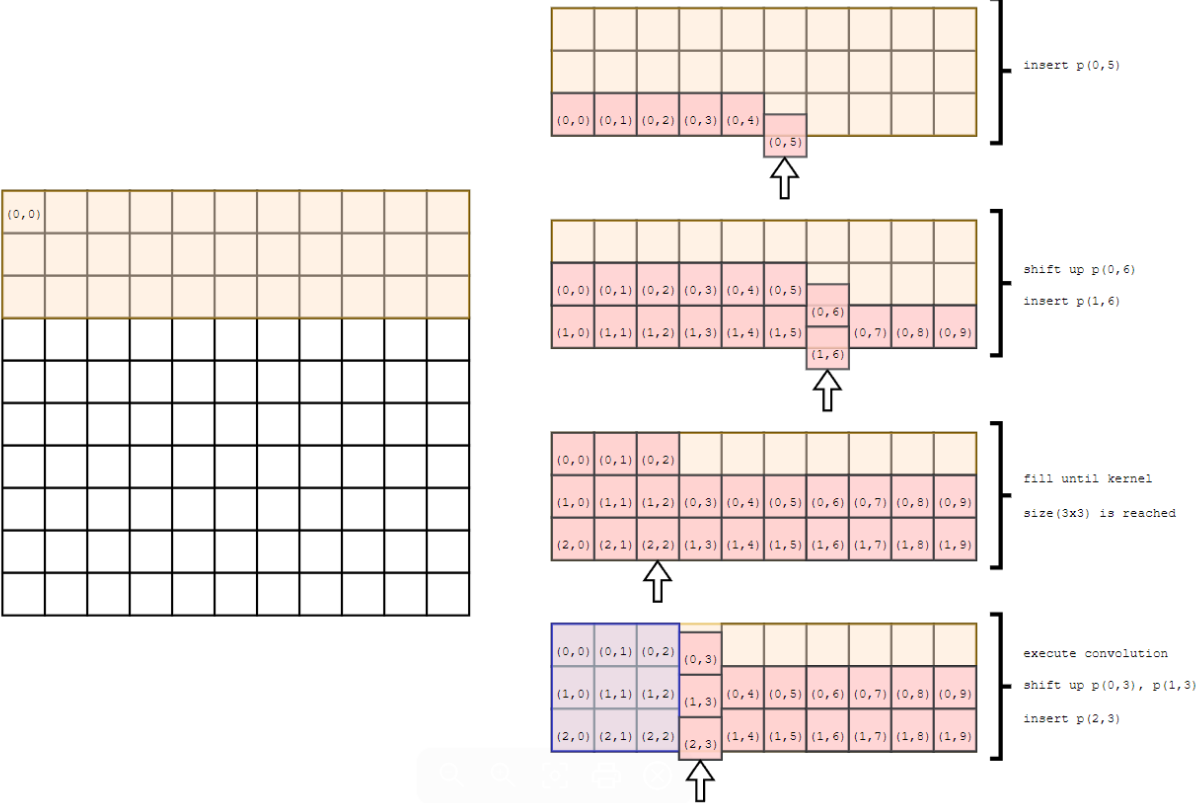


Figura 4.6. Ejecución de Line Buffers para caso de acelerador de convolución depthwise.

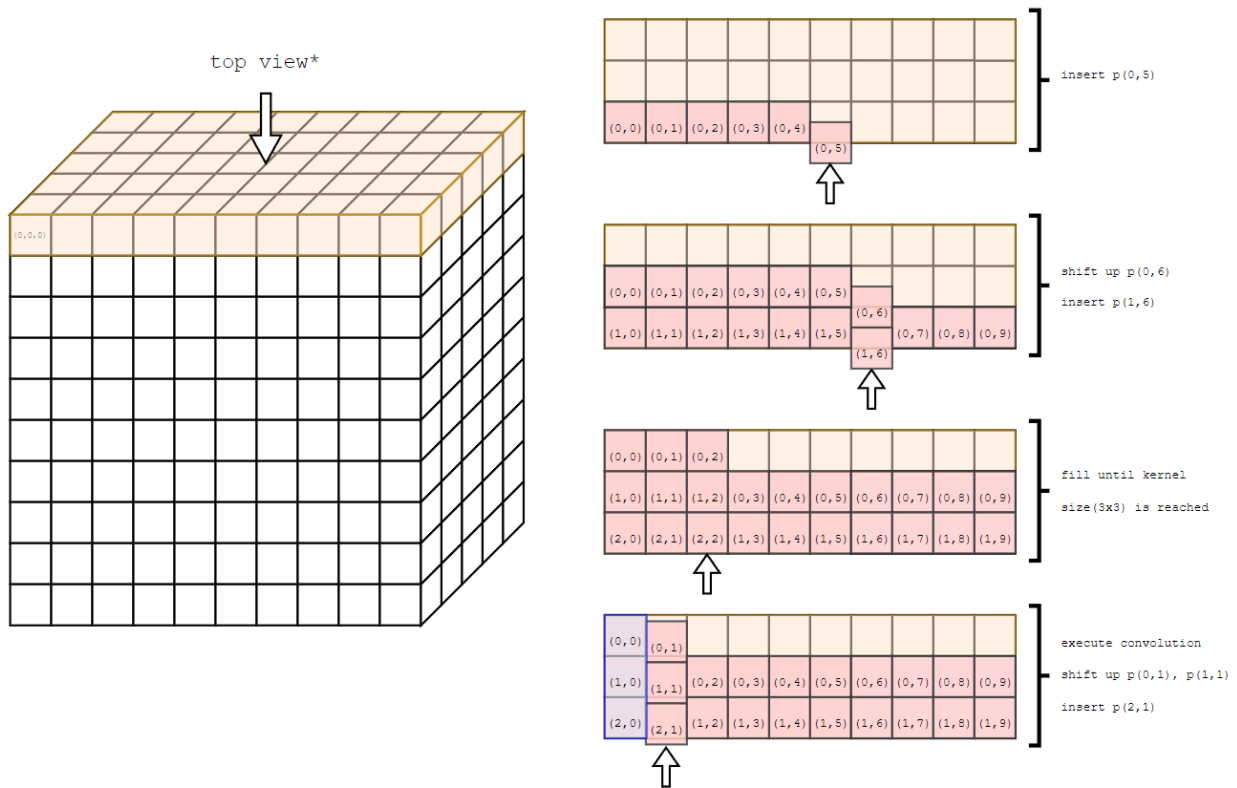


Figura 4.7. Ejecución de Line Buffers para caso de acelerador de convolución pointwise.

De esta forma se alcanza un uso más eficiente de la memoria en donde se almacena temporalmente datos de píxeles que se han leído recientemente, lo que evita la necesidad de acceder a la memoria principal repetidamente y facilita el acceso rápido a datos requeridos para el cálculo de la convolución.

Este algoritmo es de carácter general para los dos casos de convolución a tratar sin embargo es importante denotar algunas de las diferencias que se deben tomar en cuenta para un acelerador y el otro. Aunque ambas lógicas incluyen el uso de una ventana que se arrastra y realiza la multiplicación de los píxeles y el kernel, las dimensiones son distintas. La operación de la convolución depthwise llena el line buffer en base al número de filas dado por el tamaño del kernel, por lo que el acelerador inicia el desplazamiento una vez que este está cargado de este número de filas. La operación de la convolución pointwise actúa sobre una imagen con una dimensión adicional. Este también se llena en base al tamaño del kernel, pero dicho

kernel está dado por el número de canales de dicha imagen. El algoritmo para pointwise debe esperar a que toda la capa de canales se llene, y después opera.

El flujo de trabajo en Vivado HLS abarca tres principales etapas una vez desarrollado el código de los algoritmos. Primeramente, la validación/ simulación en C++, la cual corresponde a la comprobación del correcto funcionamiento del código y la compilación de este. Esta permite realizar pruebas para garantizar el funcionamiento según los requisitos establecidos y el correcto uso de las librerías y demás funcionalidades utilizadas.

Seguidamente se realiza la síntesis en donde el código en lenguaje de alto nivel se transforma en una descripción de hardware lógico, generalmente en forma de puertas lógicas y bloques funcionales. Esta etapa incluye la asignación de recursos y la optimización del diseño para cumplir con las restricciones de recursos y de rendimiento. Esta etapa es sumamente importante dentro del desarrollo del proyecto pues arroja una primera aproximación de por donde se encuentra el estado de las primeras métricas en cuanto a consumo de recursos y de tiempo. Esto permite de forma iterativa ir revisando posibles fallas en la lógica que permitan mejorar el sistema.

Por último, se debe comprobar que el código sea cosimulable. Se realiza una simulación que incluye tanto el modelo de alto nivel como la descripción de bajo nivel generada por la síntesis. Se simulan juntas para validar e internamente comparar el comportamiento y la funcionalidad del diseño en el contexto del hardware de destino. Para este se toma en consideración el testbench diseñado y se sintetiza.

Este flujo permitió constantemente estar evaluando la lógica del sistema diseñado para finalmente alcanzar el punto donde este lograra desarrollar la tarea según las especificaciones dadas.

4.6.2 Implementación de aproximaciones

Dentro de los objetivos de desarrollo de la librería de FAL está el uso de técnicas de computación aproximada para tener un menor consumo de recursos. Se espera que

al reducir el tamaño de dato o palabra que se opera de manera exacta, se reduzca el uso de recursos en la síntesis en FPGA, y se reduzca las latencias con respecto a la versión exacta de los operadores.

Se trabajaron los datos a partir de un nivel de abstracción, haciendo uso de un tipo de dato personalizado que permita al usuario extender los operadores para usar tamaños personalizados, para hacerlo solamente necesitan cambiar el tipo de dato al que se quiera usar.

```
using DataType = ap_fixed <Q_BW, Q_INT>;
```

Como se puede observar, se define un tipo de dato para el sistema según la biblioteca `ap_fixed`, la cual trabaja con este tipo de dato proporcionado por Xilinx. Este tipo de dato está diseñado para representar números fijos con un número específico de bits (`Q_BW`) y una cierta cantidad de bits fraccionales (`Q_BW – Q_INT`). Este nivel de flexibilidad precisión permite hacer distintas pruebas que se alinean a la naturaleza académica del proyecto. Además, se definieron aspectos en el código tales como el tamaño del bus el cual también resulta flexible para el manejo de los paquetes.

4.6.3 Funcionalidades añadidas

Ambos aceleradores se diseñaron para soportar diversas aplicaciones que no se limitan al Aprendizaje Profundo. Muchos de los desarrollos en hardware existentes no ofrecen la flexibilidad de trabajar con parámetros de operación como el stride, el uso de kernels de múltiples dimensiones o diferentes tamaños de imagen. Por lo tanto, ambos aceleradores se crearon para abarcar varios casos de operación de acuerdo con las limitaciones establecidas.

La implementación trabaja con varios contadores auxiliares para llevar el control del número de columnas, filas y canales y poder hacer los correctos direccionamientos a los pixeles de interés. En el caso de un stride de orden superior, si estos contadores se incrementan proporcionalmente al valor de stride definido, la ventana de operación del kernel se desplaza según ese valor y puede implementarse con un

número mayor de desplazamientos. En el caso del sistema diseñado, este se limitó a trabajar stride con un valor máximo de 2.

El tamaño del kernel es un factor importante para distinguir entre ambos aceleradores pues es su diferencia principal. La forma en que se debe trabajar es a partir de un padding de ceros al kernel para cualquiera de los dos casos, sin embargo, de igual forma se debe considerar un valor máximo según las capacidades que se apunta satisfacer. Para el caso de la convolución depthwise se definió la capacidad máxima de un kernel de tamaño 5 sin embargo, la forma en que este trabaja se realiza como se muestra a continuación.

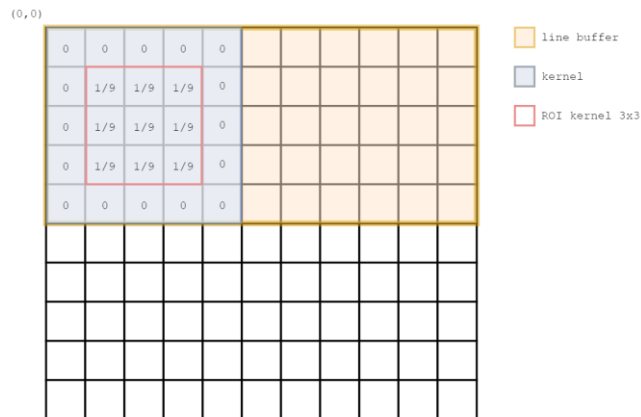


Figura 4.8. Padding a kernel para uso general en convolución depthwise.

En este caso, se está utilizando un kernel de 5x5 para todas las aplicaciones, pero si se necesita trabajar con un kernel más pequeño, como uno de 3x3, se agrega un padding de cero. Esto permite que, aunque se realice la multiplicación considerando toda la matriz de 5 filas y 5 columnas, solo se tomen en cuenta los valores relevantes dentro de la región de interés de 3x3. De esta manera, se ajusta el proceso para adaptarse a diferentes tamaños de kernel sin afectar el resultado final.

Nótese que a partir de esto el line buffer que se definió también debe tener un tamaño de filas igual al máximo del tamaño del kernel, es decir en este caso 5.

En el caso de pointwise de igual forma se ejecuta un padding para satisfacer las dimensiones definidas. Las dimensiones máximas de imagen o capas corresponden

a 16 por lo que las dimensiones del kernel son acordes a este. Así se visualiza esta operación:

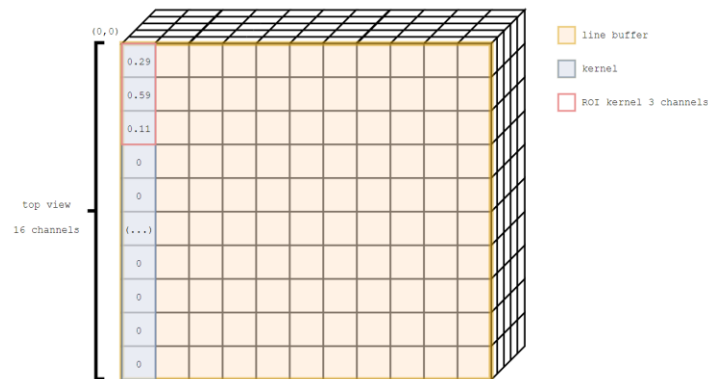


Figura 4.9. Padding a kernel para uso general en convolución pointwise.

En este caso se utiliza un kernel de 1x1x16, lo cual resulta útil para aplicaciones de Deep Learning donde los canales entre capas de convolución van aumentando en grandes órdenes, pero paralelo al caso de depthwise, si se requiere trabajar una aplicación más convencional, con imágenes de 3 canales, se aplica un padding de ceros a los canales del 4to en adelante para solo tomar en cuenta los valores relevantes. El tamaño del line buffer se toma también según el número máximo de canales que soporta el acelerador.

Ahora bien, en aplicaciones de Deep Learning, los órdenes de capas resultan en ocasiones de hasta más de 1000, por lo que un acelerador que solo soporta 16 debe ajustarse acorde. La estrategia es que, si se realiza la operación sobre 16 canales, pero se requiere abarcar un mayor número de canales, el resultado de la convolución anterior vaya siendo tomada en cuenta y sumada para la siguiente convolución.

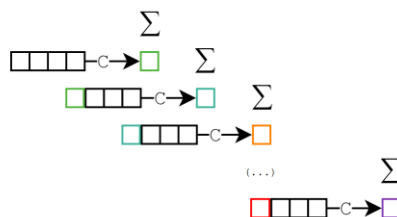


Figura 4.10. Convolución pointwise realizada sobre capas de +16 canales.

Ahora bien, aplicando esa lógica, la cantidad de canales debe tener un valor tal que se abarquen todos los canales. Por ejemplo, si se tienen 64 canales, se toman los primeros 16, pero después se debe reservar un canal en la próxima convolución para arrastrar el resultado anterior por lo que solo se considerarían 15 canales, por lo que no basta con dividir los 64 canales entre 16 y realizar la operación 4 veces. Si se realiza 4 veces, se abarcarían únicamente 16+15+15+15 canales, es decir que 3 canales quedarían sin convolucionar. Para poder abarcar estos canales se debe ingresar una imagen con un número específico de canales Y dado por la siguiente fórmula:

$$Y = (X - 1) - (N \% (X - 1)) \quad [13]$$

X = Canales que soporta el acelerador

N = Canales en la capa

El acelerador también presenta limitaciones en el número de columnas y filas que se definen a partir de las capacidades de memoria y recursos de la FPGA. Aunque se pudieran definir límites para soportar imágenes en los órdenes que hoy en día trabajan las computadoras modernas, para el área de visión que se apunta con el proyecto, basta con definir casos de operación reducidos. El valor de estos parámetros también define la otra dimensión del line buffer (recordando que la primera dimensión está dada por el valor del kernel). En el caso de depthwise, el line buffer tiene dimensiones del número de columnas máximo. En el caso de pointwise, está dado por el número de canales.

Es importante destacar que los aceleradores son para uso general. Las capacidades máximas y mínimas son límites de lo que puede llegar a soportar, pero valores en medio de este rango se pueden trabajar. Como se ha comentado, el diseño está hecho para que, si se trabajan rangos menores, se realicen los paddings respectivos y se soporten las operaciones.

Dentro de las limitantes eso sí, se encuentra que el número de columnas debe ser un valor múltiplo de la cantidad de datos por paquete. Si en un paquete de datos caben 8 datos (que en este caso corresponden a pixeles), el número de columnas debe ser múltiplo de 8.

En las siguientes tablas se sintetizan las capacidades de los aceleradores con la justificación en caso de ser necesaria para el valor escogido:

Tabla 4.8. Límites operativos del acelerador depthwise.

Parámetro		Valor	Justificación
Kernel	MAX	5	Modelos de redes en la academia no suelen ser mayores a 5.
	MIN	3	Valor mínimo para operación según los estándares.
Stride	MAX	2	Las aplicaciones con un stride mayor a 2 no suelen ser comunes. Facilidad de cómputo.
	MIN	1	Mínimo valor lógico posible.
Filas	MAX	1024	Capacidades de memoria en FPGA
	MIN	5	Dado por el valor máximo de Kernel.
Columnas	MAX	1024	Capacidades de memoria en FPGA. Datos por paquete definidos.
	MIN	Datos por paquete	Mínimo múltiplo posible según limitaciones de envío de paquetes definida.

Tabla 4.9. Límites operativos del acelerador pointwise.

Parámetro		Valor	Justificación
Filas	MAX	1024	Capacidades de memoria en FPGA.
	MIN	1	Dado por el valor máximo de Kernel.
Columnas	MAX	1024	Datos por paquete definidos.
	MIN	Datos por paquete	Mínimo múltiplo posible según limitaciones de envío de paquetes definida.
Canales	MAX	16	Según capacidades de recursos en FPGA de bajo perfil.
	MIN	3	Según canales de imagen RGB o estándares populares.
Kernel	SIZE	16	Kernel siempre será igual a la cantidad de canales máximo. Si son menos canales aplica como previamente se explicó.

4.6.4 Directivas de Optimización

El proceso de optimización es sumamente utilizado en High Level Synthesis ya que permite que a partir de un mismo desarrollo se alcancen varias soluciones que puedan tener distintos enfoques. Este proceso se lleva a cabo a partir del uso de *pragmas* los cuales son proveídos y soportados por la herramienta de Vivado HLS.

Cada función (load, write, etc.), está planteada para que se pueda evaluar individualmente con distintas optimizaciones. En la Tabla 4.10 se exponen las distintas soluciones a partir de las posibles arquitecturas. Todas estas soluciones se evalúan según una arquitectura tipo dataflow. Esta se plantea de la siguiente forma en base a lo expuesto en el Marco Teórico.

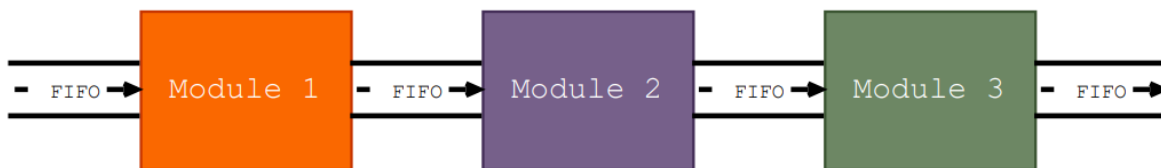


Figura 4.11. El diseño por dataflow involucra añadir interfaces FIFO para comunicar cada módulo.

El proceso de desarrollo para optimizaciones en la herramienta de Vivado HLS se puede visualizar en la Figura 4.12. En este se puede observar como la herramienta ubica cada uno de los módulos según el orden de ejecución apropiado y las instrucciones de optimización asignadas.

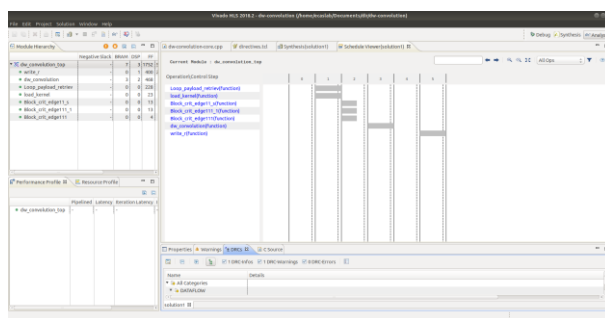
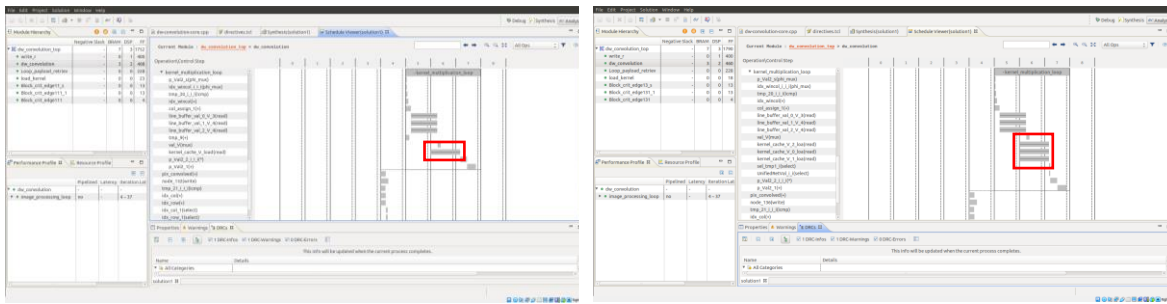


Figura 4.12. Ambiente de análisis para evaluación del flujo del sistema.

Por ejemplo, en la Figura 4.13 se aprecia el cambio en el flujo de operación cuando se realiza una directiva de partición en el arreglo del kernel. De esta forma la lectura

de las dimensiones de un kernel se realiza en paralelo lo que para cada momento en que es invocado vuelve más eficiente su lectura pues resulta de más fácil acceso. Esto genera una disminución a nivel de latencia en el diseño. Nuevamente, esto representa un mayor consumo de recursos por lo que es un aspecto que considerar según el objetivo de diseño que se desee, menos recursos o menos latencia.



(a)

(b)

Figura 4.13. Cambio en el flujo de operación a partir de directiva de array partition
 Como se comentó previamente, la lógica utilizada para el acelerador de la convolución depthwise y pointwise es muy similar por lo que, bajo esta premisa, las optimizaciones se realizan a manera de espejo entre las funciones de una y otra.

Tabla 4.10. Tabla Resumen para Optimizaciones

Concepto	Pipeline	Unroll	Array Partition	Inline	Loop Flatten	Arquitectura
A		Load Data Write Data	<u>Line Buffer</u>	Load Data		Dataflow
B		Load Data Load Kernel* Write Data Convolution	<u>Line Buffer</u> <u>Kernel</u>	Load Data	Load Kernel*	Dataflow
C	Load Kernel* Convolution	Load Data Load Kernel Write Data	<u>Line Buffer</u>	Load Data		Dataflow
D	Convolution	Load Data Load Kernel* Write Data	<u>Line Buffer</u> <u>Kernel</u>	Load Data	Load Kernel*	Dataflow
E	Convolution	Load Data Load Kernel* Write Data	<u>Line Buffer</u> <u>Kernel</u>	Load Data Load Kernel*	Load Kernel*	Dataflow

* en el caso del módulo Load Kernel para el acelerador de pointwise convolution, este no se manejó con un UNROLL ni LOOP FLATTEN por la forma de carga de un kernel de 1D.

** En **Negrita** las optimizaciones a loops. En Subrayado las optimizaciones a estructuras de datos.

Las optimizaciones presentadas en el eje horizontal de la tabla pueden ser consultadas en la sección 2.2.2.2 del presente informe. Cada una de estas es capaz

de ejecutar un módulo específico según ciertas reglas. Cada uno de estos conceptos se evaluó según su consumo de recursos y latencia según el testbench, lo cual se abordará más adelante para tener una solución definida para las etapas posteriores.

Otro detalle fundamental dentro del proceso de optimización correspondió a la depuración del código para evitar el uso de operaciones costosas para el hardware, como es el caso de la división. Se buscó que las operaciones se basaran en operaciones básicas como desplazamientos y lógica binaria para manipular los paquetes de datos a un nivel bajo aún desde este paradigma de HLS.

4.7 Testbench y Reporte de Vivado HLS

Para evaluar el sistema durante el proceso de diseño se trabajó con un testbench que fuera capaz de evaluar el correcto funcionamiento de cada una de las etapas y módulos de los aceleradores. Este consistió en un código en C++ capaz de enviar los datos de una imagen de dimensiones flexibles para probar distintos casos y tamaños. Básicamente, este testbench recibe una imagen y la envía al acelerador, para luego recibirla de vuelta y guardarla como un archivo .png. Se utiliza la librería de OpenCV con la cual se logra la lectura de imágenes a un tipo de objeto propio de la librería y se programó un algoritmo que convirtiera este objeto en matrices de datos en C++.

Las pruebas se llevaron a cabo con imágenes de dimensiones reducidas por motivos de practicidad. La imagen de prueba corresponde a una llamada 'frog', perteneciente al popular conjunto de datos para DL de CIFAR10. Esta imagen presenta los siguientes parámetros:

```
height = 32;  
  
width = 32;
```

Ahora bien, la imagen se evalúa con parámetros adaptables según cada uno de los casos.

Específico de acelerador Depthwise:

```

kernel_size = 3;

stride_value = 1;

kernel[MaxKernelSize * MaxKernelSize] =
{0, 0, 0, 0, 0,
0, 0, 0, 0, 0,
0, 0, 1, 0, 0,
0, 0, 0, 0, 0,
0, 0, 0, 0, 0};
//MaxKernelSize = 5 como se mencionó en las capacidades.

```

Específico de acelerador Pointwise:

```

kernel[KernelSize] =
{0.29, 0.59, 0.11, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
//KernelSize = 16 como se mencionó en las capacidades.

```

Este testbench se evaluó para cada una de las arquitecturas según las distintas configuraciones de optimización que se probaron. De esta forma fue posible evaluar el consumo de recursos preliminar y la latencia entre una solución y otra. Estos datos se presentan en la siguiente tabla que hace referencia a los conceptos de la Tabla 4.10. Como se mencionó, la FPGA corresponde a una FPGA Artix 7 por lo que los valores arrojados son según las especificaciones de esta.

Tabla 4.11. Tabla de consumo de recursos y latencia para optimizaciones propuestas en convolución depthwise.

Concepto	LUTs (%)	FF (%)	DSP48E (%)	BRAM (%)	Latency
no optimization	18	3	2	6	34036
A	13	3	2	6	34036
B	14	3	2	6	34000
C	15	3	2	6	14210
D	15	3	2	6	14200
E	14	3	2	6	14200

La latencia en este caso se define como los ciclos de reloj que dura el acelerador para ejecutar el sistema de inicio a fin. En el caso de la convolución pointwise, los resultados de los conceptos evaluados se presentan a continuación.

Tabla 4.12 Tabla de consumo de recursos y latencia para optimizaciones propuestas en convolución pointwise.

Concepto	LUTs (%)	FF (%)	DSP48E (%)	BRAM (%)	Latency
no optimization	13	3	3	13	21669
A	8	3	3	13	21669
B	10	3	3	13	21669
C	8	3	2	12	17574
D	10	3	2	12	17574

** por la lógica de implementación del acelerador pointwise, el concepto E no se puede implementar.*

4.7.1 Design Space Exploration

Una vez que se obtuvo una solución óptima según las distintas directivas que se aplicaron, este concepto se tomó para realizar un estudio de sus capacidades computacionales. Este estudio se denomina Design Space Exploration y consiste en probar distintas combinaciones de operación según los distintos parámetros flexibles del acelerador modificando la precisión arbitraria de los datos, el tamaño del kernel y otros parámetros que pueden considerarse, pero no son tomados en cuenta pues no resultan de interés para los objetivos de la investigación en el laboratorio.

Las pruebas también implicaron la comparación de métricas entre los resultados de la convolución generados por el acelerador de hardware y los resultados obtenidos mediante funciones de software de librerías comunes en el campo. Es importante tener en cuenta que estas librerías son principalmente diseñadas para su uso en entornos de software y no son traducibles a hardware de la misma forma con HLS en contextos de hardware.

Las pruebas en sí implicarían revisar el consumo de los recursos de la FPGA en diferentes modos de operación. Además, se analizan diversas métricas que se

detallan a continuación para comparar la salida de la imagen tanto en el software como en el hardware en varias configuraciones.

4.7.1.1 Error Cuadrático Medio

El cálculo del error cuadrático medio es una métrica comúnmente utilizada para comparar dos imágenes. Esta calcula la diferencia cuadrática media entre los valores de cada píxel de una imagen con otra. Suele utilizarse como una medida de la calidad de la imagen o para evaluar la precisión de un algoritmo de procesamiento de imágenes [55]. La fórmula con la que se realiza el cálculo corresponde a:

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i,j) - K(i,j)]^2 \quad [14]$$

m y n son el ancho y altura de la imagen

$I(i,j)$ es el valor del píxel en la imagen de referencia

$K(i,j)$ es el valor del píxel en la imagen procesada

En el caso del presente trabajo un MSE bajo, sería señal de que el acelerador ejecuta la función de convolución de forma adecuada en comparación con un procesamiento en software.

4.7.1.2 Peak Signal to Noise Ratio (PSNR)

Peak Signal-to-Noise Ratio (PSNR) es de igual forma una métrica comúnmente utilizada para evaluar la calidad de una imagen respecto a una referencia. La PSNR se calcula como la relación entre la potencia máxima de una señal y el error cuadrático medio (MSE) entre dos imágenes. Se utiliza para medir la calidad de una imagen en términos de ruido y distorsión, proporcionando una medida cuantitativa de la fidelidad de la imagen en comparación con la original. El PSNR se evalúa en decibelios, y un valor más alto de PSNR generalmente indica una mejor calidad de imagen y una menor distorsión. [56]

$$PSNR = 10 \log \left(\frac{\max(\hat{X}^2)}{MSE} \right) \quad [15]$$

\hat{X} es el valor máximo de píxel

4.7.1.3 Índice de Similitud Estructural (SSIM)

El Índice de Similitud Estructural (SSIM) tiene en cuenta la similitud estructural, la luminancia y el contraste entre las imágenes para proporcionar una estimación más precisa de la calidad de la imagen en comparación con la métrica del Error Cuadrático Medio (MSE) o el PSNR.

El sistema visual humano está muy adaptado para extraer información estructural de las imágenes, de tal forma que una medida de la información estructural puede dar una buena aproximación de la calidad de imagen percibida. [57]

Este cálculo da como resultado una medida global de similitud adimensional. El valor de SSIM oscila entre -1 y 1, donde 1 indica una perfecta similitud entre las imágenes y -1 indica una completa discrepancia [58]. El SSIM es utilizado para medir la calidad de la imagen de una manera más perceptual y realista que las métricas tradicionales basadas únicamente en la intensidad de los píxeles.

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \quad [16]$$

x y y son las imágenes a comparar

μ_x y μ_y representan las media de x y y

σ_x y σ_y son las desviaciones estándar de x y y

σ_{xy} es la covarianza de x y y

C_1 y C_2 son constantes de estabilización para evitar la inestabilidad numérica

Si bien el MSE computa por el promedio de las diferencias al cuadrado de la imagen operada y la imagen de referencia píxel por píxel, el SSIM utiliza un método más apegado a la forma en que los seres humanos perciben las imágenes por medio de un sistema HVS (human visual system).

4.7.1.4 Configuraciones para pruebas en los aceleradores.

A continuación, se detallan las permutaciones que fueron evaluadas y sobre las cuales se obtuvieron los resultados para los aceleradores. Estas se introducen a partir de una estructura de datos de tipo json.

Convolucionador Depthwise:

```
{
  "ACCELERATOR": ["depthwise-convolution"],
  "Q_KS": [3, 5],
  "Q_BW": [4,8,10,12,16],
  "Q_INT": [1,2],
  "Q_O": [1024],
  "Q_CORE": ["None"],
  "Q_PES": [1],
  "CLOCK_PERIOD": [10],
  "TB_ARGV": ["examples/depthwise-convolution/misc/lenna.png", "examples/depthwise-convolution/misc/baboon.png", "examples/depthwise-convolution/misc/barbara.png"],
  "Q_FIXED_RATIO": 0.5,
  "Q_INPUTS_TB": [0],
  "Q_OUTPUTS_TB": [0],
  "Q_SEED": [0]
}
```

Convolucionador Pointwise:

```
{
  "ACCELERATOR": ["pointwise-convolution"],
  "Q_KS": [3],
  "Q_BW": [4,8,10,12,16],
  "Q_INT": [1,2],
  "Q_O": [1024],
  "Q_CORE": ["None"],
  "Q_PES": [1],
  "CLOCK_PERIOD": [10],
  "TB_ARGV": ["examples/pointwise-convolution/misc/lenna.png", "examples/pointwise-convolution/misc/baboon.png", "examples/pointwise-convolution/misc/barbara.png"],
  "Q_FIXED_RATIO": 0.5,
  "Q_INPUTS_TB": [0],
  "Q_OUTPUTS_TB": [0],
  "Q_SEED": [0]
}
```

Las variables aquí incluidas se resumen en el cuadro a continuación pues según estas el consumo de recursos y el tiempo de procesamiento varía.

Q_KS:	Q_BW:	Q_INT:	Q_O:	** El resto de las variables se definen por default ya que no resultan importantes para estas unidades.
Tamaño Kernel	Tamaño de Dato	Tamaño de valor entero en dato	Tamaño máximo de dimensiones	
TB_ARGV:				
Imagen de prueba (512p x 512p)				
Lenna:	Baboon:	Barbara:		
				

Importante mencionar que, para la validación, los kernels utilizados fueron los siguientes: en primer lugar, para la convolución depthwise se empleó un filtro gaussiano que provoca un efecto de desenfoque en la imagen (haciéndola más borrosa). Este filtro se aplicó únicamente al primer canal de la imagen, lo que la convierte en una imagen puramente en 2D y sus valores de píxeles no pueden representar los 3 canales de color, por lo que se mapea directamente a una escala de grises. En el caso de la convolución pointwise, se realizó una conversión directa a escala de grises a través de un cálculo ponderado. Matemáticamente, al multiplicar cada canal por una constante específica y sumar los resultados, se obtiene el valor en escala de grises que mejor representa la imagen. Esta operación es la que comúnmente se utiliza para obtener una imagen en blanco y negro en aplicaciones convencionales.

4.8 AxC Executer y Simulación de red en FPGA

Una vez desarrollados los aceleradores, se procedió a la integración de estos dentro de una red neuronal convolucional a nivel de FPGA. Si bien se explicó que una red neuronal está compuesta por una serie de capas que realizan distintas operaciones matemáticas en donde la convolución pointwise y depthwise son parte de estas, existen otras operaciones que se deben tomar en cuenta para poder realizar un proceso de clasificación. Casos como realizar batch normalization, una función de activación, o sumas y multiplicaciones son parte de lo que la red debe cubrir.

Para poder realizar la integración y probar una aplicación de clasificación, se trabaja en un framework conocido como AxC Executer, que permite tomar distintos modelos de red para realizar distintas operaciones de Machine Learning a nivel de FPGA en conjunto con una simulación en software. Este framework está programado en su mayoría en C++, y permite para las operaciones que no presentan un acelerador, ser simuladas en software mientras que, para ejecutar operaciones de convolución, estas se ejecutan simuladas en FPGA.

La red por simular consiste en la MobileNetV2, la cual contiene 206 capas distribuidas entre distintas operaciones. Las capas de convolución se reemplazan en software por su versión en hardware y se corre la red para determinar la precisión con la que se realiza una clasificación. La Figura 4.14. Estructura en capas de la red MobileNetV2. Denotado en rojo las capas de convolución. ilustra a modo de bloques la estructura de la red y denota encerrado en color rojo los dos bloques de interés relacionados a la operación de convolución tanto depthwise como pointwise. La red toma ciertos parámetros de entrada y llama al acelerador haciendo uso de la librería de FAL. Se debe crear un código para hacer el llamado del acelerador y manejar el stream de datos y la red se apoya de este código para realizar la clasificación. En lo que respecta al flujo de trabajo del framework, este se basa en utilizar el acelerador por medio de un código muy similar al testbench donde en el ambiente de Vivado HLS se realizaron las pruebas. Básicamente se hace un llamado a la función para que procese la imagen de interés.

Cada una de las capas que se presentan en la Figura 4.14 fuera de las de convolución se programó en software y se hace un llamado de estas por medio de la estructura de la red, donde para cada capa se definen todos los parámetros de entrada y se espera una salida acorde que la siguiente capa debe procesar. En futuros desarrollos se puede realizar una integración en cada capa utilizando aceleradores para sumas y multiplicaciones de FAL, con aproximaciones novedosas con menor consumo de recursos, pues de reducir estas, la red podría resultar más eficiente.

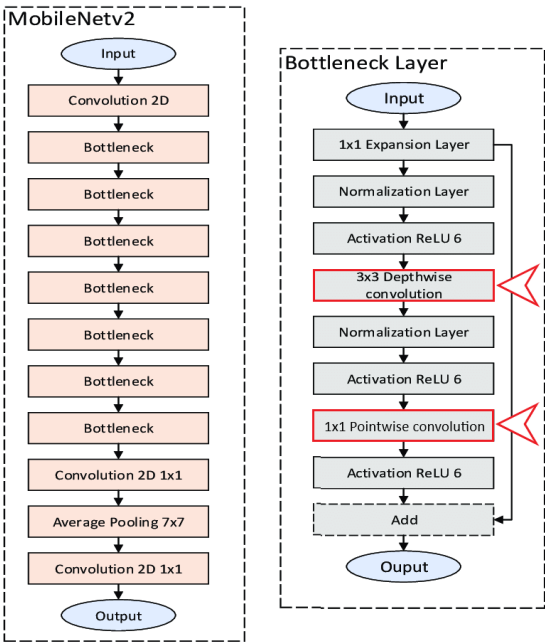


Figura 4.14. Estructura en capas de la red MobileNetV2. Denotado en rojo las capas de convolución.

5. Resultados y Análisis

El propósito de este capítulo es presentar y analizar detalladamente los resultados alcanzados durante el desarrollo del proyecto. Se lleva a cabo una comparación entre los resultados obtenidos y las referencias establecidas, con el fin de evaluar la efectividad y la validez de los logros alcanzados hasta la fecha. Se explorarán las métricas y enfoques para identificar áreas de mejora potencial en el diseño, tal como se plantearon anteriormente con el objetivo de perfeccionar y refinar la implementación. Asimismo, se destacarán las explicaciones clave derivadas de estos análisis, ofreciendo una perspectiva clara y coherente en relación con el desarrollo y los objetivos del proyecto.

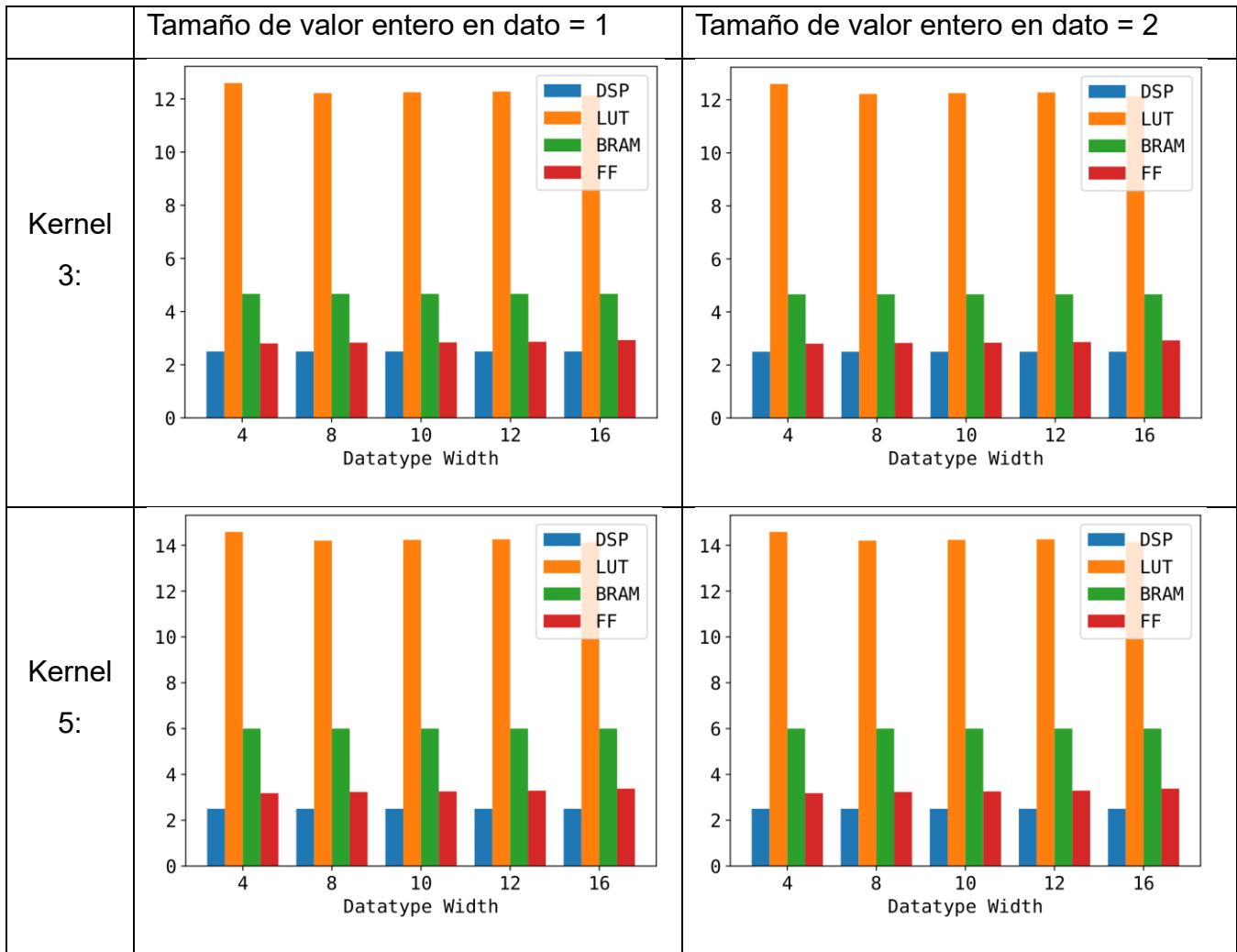
5.1 Pruebas de consumo recursos

Guiado por el Design Space Exploration, se realizaron pruebas sobre ambos aceleradores para determinar el consumo de recursos que estos generaban. Como se observa en la Sección 4.7.1.4, ambos aceleradores se probaron bajo distintas configuraciones.

5.1.1 Consumo de recursos en Convolución Depthwise

El objetivo de estas pruebas es evaluar como ante distintos tamaños de dato, la operación de convolución depthwise variaba. Bajo una lógica según los comportamientos típicos, entre mayor sea el tamaño del dato, mayor resulta ser el uso de recursos.

Tabla 5.1. Consumo del porcentaje total de recursos en FPGA Artix7 para acelerador de convolución depthwise.



Como se observa, el cambio en el tamaño de dato de 4 a 16 y entre el valor del dato entero entre 1 y 2 no cambia significativamente. Únicamente se visualiza un cambio cuando se manejan distintos tamaños de kernel en el consumo de LUTs y BRAM. Generalmente un mayor tamaño de dato se asocia a un mayor consumo de memoria, recursos de lógica, y velocidad de cálculo que se traduce en tiempo. Sin embargo, varios factores pueden explicar por qué el consumo tiene variaciones mínimas.

Optimización automática del compilador HLS: El compilador de HLS puede optimizar el código para utilizar de manera más eficiente los recursos de la FPGA

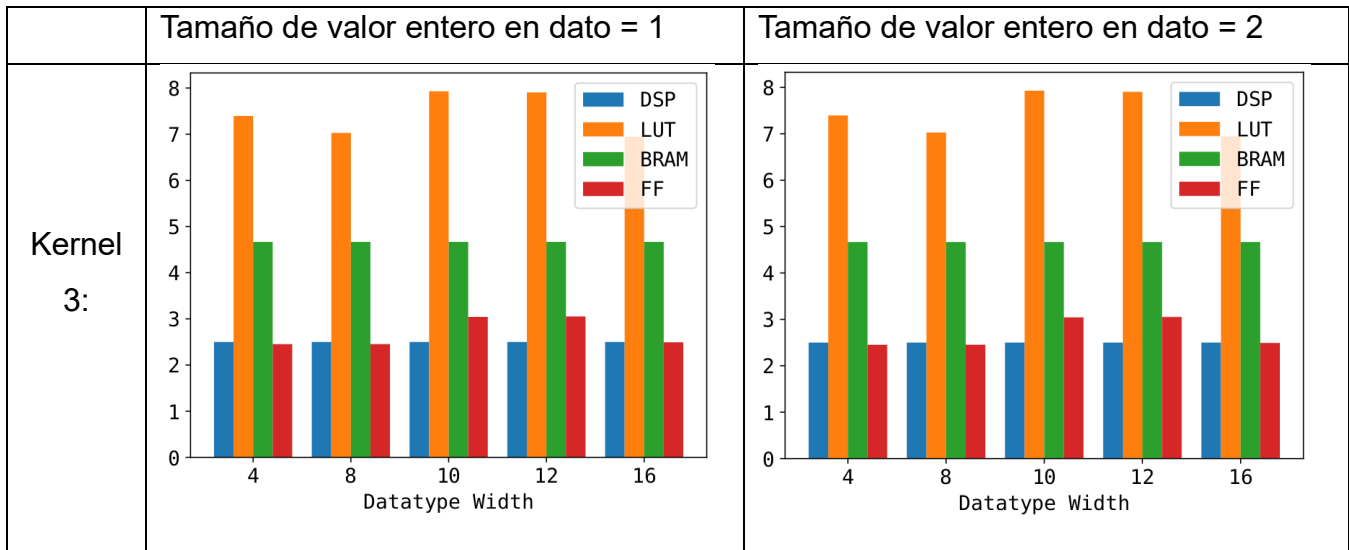
[59]. Esto puede incluir la optimización de variables y operaciones, lo que resulta en un consumo de recursos similar para diferentes tamaños de datos.

Estructuras de control y flujo de datos: El diseño general de la lógica de control y el flujo de datos puede estar optimizado de tal manera que el tamaño del dato no tenga un impacto considerable en el uso de recursos. Es decir que la misma lógica del sistema y la forma que este se programó influye en gran medida en cómo se asignan los recursos para la operación. El uso de una estructura de datos como la de Line Buffers proveída por las librerías estándar de HLS o bien técnicas de corrimientos de bits contribuyen a que a mayor tamaño de dato se tengan resultados similares. Algo que apoya esta hipótesis es que la variable del Line Buffer se define de forma flexible a partir del tamaño de Kernel que se escoja, al escoger un tamaño de 3 o de 5, el Line Buffer crece o decrece su tamaño resultando en necesidad de mayor memoria (BRAM) y mayor cantidad de operaciones por ciclo de reloj (LUTs).

5.1.2 Consumo de recursos en Convolución Pointwise

También se evalúa la convolución pointwise, que sigue un procedimiento similar. No obstante, se analizó un único tamaño de kernel debido a las particularidades de la operación, en la cual el cambio de tamaño no producía una variación en la imagen resultante de la convolución. Anteriormente se mencionó que el tamaño máximo del kernel era de 16 debido a una decisión de diseño. No obstante, en este caso, al validar los resultados del acelerador en una imagen de 3 canales, y no necesariamente en una aplicación de DL, donde los canales podrían ser superiores a 3 según la capa, se llevaron a cabo pruebas definiendo dicho valor en 3.

Tabla 5.2. Consumo del porcentaje total de recursos en FPGA Artix7 para acelerador de convolución pointwise.

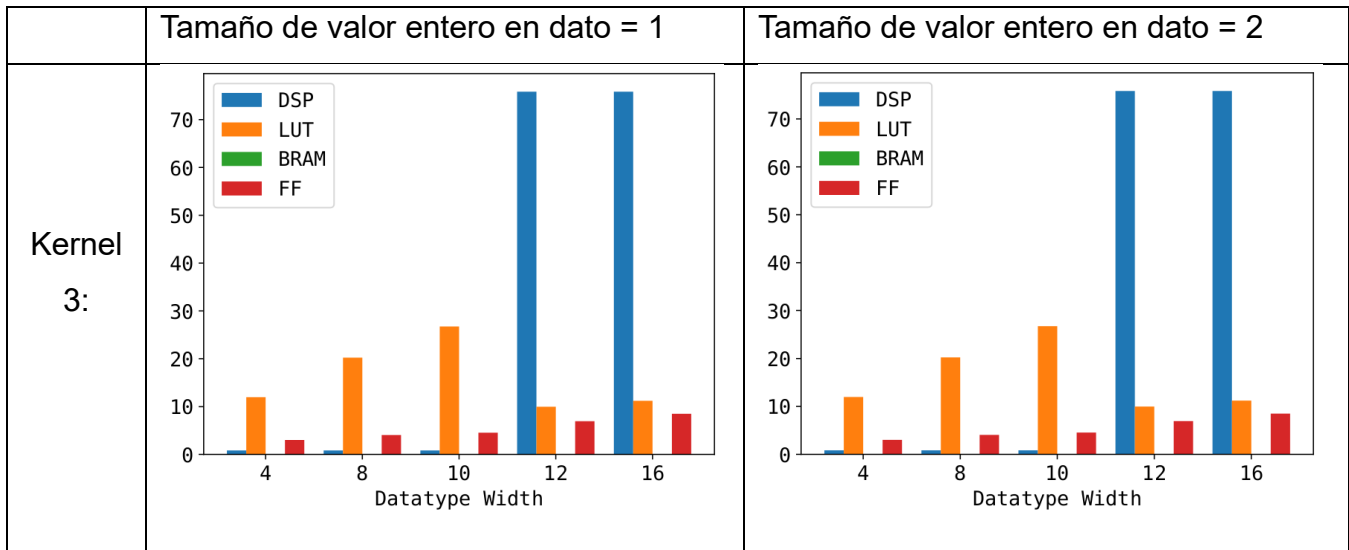


Nuevamente, en esta operación, el tamaño de dato y la parte entera no resultan ser un factor de mayor peso sobre los recursos, lo cual era esperado dado que la lógica de esta operación es idéntica a la utilizada para la convolución depthwise. Entre las diferencias a resaltar entre esta implementación y los resultados anteriores, se encuentra el manejo de tamaños de datos que no se ajustan correctamente a un bus de datos de 64 bits, como los casos de 10 y 12. Para estos casos, se requiere un tratamiento especial de los datos para enviar un número proporcional de paquetes, lo cual se logra mediante el agregado de ceros como columnas adicionales sin inherencia en el resultado de la imagen pero que sí se expresan en un mayor consumo de LUTs y FFs.

5.1.3 Consumo de recursos en aproximaciones anteriores

Dentro de las aproximaciones previamente realizadas en FAL para implementar unidades de convolución se compara bajo las mismas condiciones de operación para comparar el consumo de recursos. La técnica de convolución actualmente presente corresponde a stream-convolution la cual presenta cierto nivel de paralelismo en la ejecución. Esta aplica convolución sobre una imagen, y resulta en la aproximación más cercana que existe dentro del framework de FAL en comparación con las convoluciones aquí tratadas.

Tabla 5.3. Consumo del porcentaje total de recursos en FPGA Artix7 para acelerador de convolución por stream.



Respecto a este acelerador, presenta varias particularidades y diferencias, ya que no envía todos los datos de la imagen; en cambio, funciona como un operador de convolución invocado desde el TB. Básicamente, se invoca esta operación acelerada por cada desplazamiento definido en el TB. Se destaca su nulo consumo de memoria BRAM, a expensas de un alto consumo de otros recursos. Así, se evidencia cómo la aproximación tratada en este informe se convierte en una opción cuando la memoria no es un factor crítico. Por el contrario, para preservar el resto de los recursos, una aproximación de convolución con line buffers puede resultar más útil. Además, es importante considerar que esta aproximación presentada en la tabla anterior varía con el tamaño de dato, y a partir de 12 para un bus de datos de 64, dispara el consumo de DSPs, lo que también debe tenerse en cuenta al elegir la aproximación de convolución. Esta aproximación además solo considera un tamaño de kernel.

Esta convolución presente en FAL no resulta una comparación fiel a la convolución pointwise, pero de igual manera evidencia como la operación de convolución con otras técnicas de programación pueden tener consumos de recursos muy superiores en proporción a los obtenidos.

5.2 Pruebas de Latencia

Estas pruebas buscan evidenciar el tiempo que toma para que una operación específica se complete durante la simulación a nivel de sistema. Estas se realizan durante la co-simulación (proceso del flujo de trabajo de HLS). Según sea el caso del acelerador, va a constatar todo el proceso desde que entra la imagen hasta que esta sale convolucionada. De igual forma, como parte de las pruebas del DSE se obtuvieron los resultados según las configuraciones establecidas.

5.2.1 Latencia en Convolución Depthwise

Estas pruebas muestran para distintos tamaños de dato, de valor entero y de kernel el valor de latencia. Cada uno de estos es una unidad de hardware individual implementable en FPGA que puede operar bajo esas capacidades. En la leyenda, se encuentran valores representados por una *E* y una *C* en referencia al error y el consumo relativo dentro de la FPGA respectivamente.

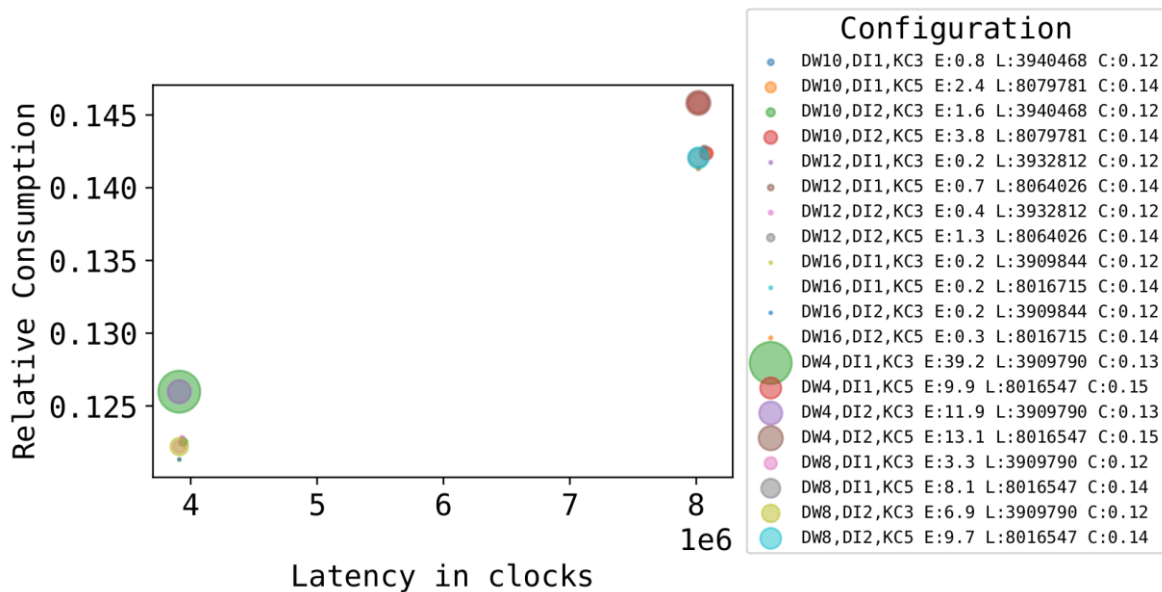


Figura 5.1. Latencia por unidad de hardware testeada durante el DSE para convolución depthwise.

Como se observa, existen dos agrupaciones en cuanto a la latencia, aquellas unidades con kernel de 5 y aquellas con kernel de 3. Esto es un comportamiento esperado debido a que mientras que un kernel de 5 ejecuta 25 multiplicaciones, un kernel de 3 únicamente ejecuta 9.

La media en latencia para el kernel de 3 es de 3920541 ciclos de reloj con un periodo de 10ns. Un kernel de 5, que es el máximo soportado por el acelerador tiene una latencia casi del doble con un valor promedio entre las unidades de hardware de 8038723 ciclos de reloj.

5.2.2 Latencia en Convolución Pointwise

En el caso de la convolución pointwise, los resultados se presentan en la Figura 5.2 siguiendo la misma lógica de resultados que se presentó anteriormente.

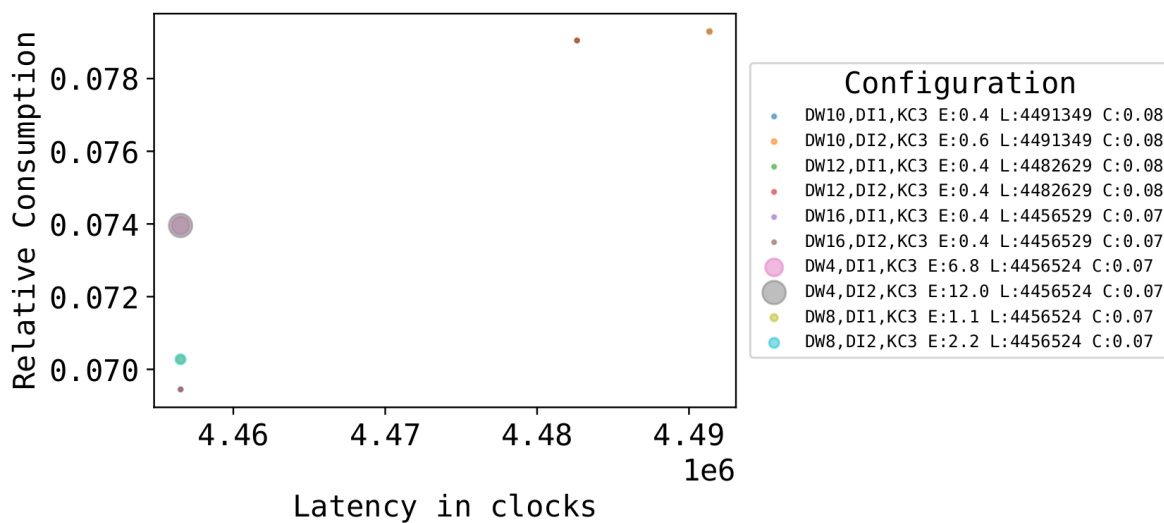


Figura 5.2. Latencia por unidad de hardware testada durante el DSE para convolución pointwise.

Como tal, se aprecia como cada una de las unidades evaluadas tiene un consumo mucho menor de recursos de aproximadamente solo un 7~8% del total y que el procesamiento total de todos los píxeles tiene en promedio una latencia de 4468711 ciclos de reloj. Considerando que este acelerador toma 3 veces el número de píxeles (dado el número de canales), se revalida que esta operación es muy eficiente en la duración de ejecución.

5.2.3 Latencia en aproximaciones anteriores

La aproximación de convolución presente actualmente en FAL se presenta a continuación donde como se explicó anteriormente consiste en un acelerador de la operación de convolución que es invocado por cada multiplicación entre un kernel y

la ventana de interés en la imagen. Esto quiere decir que, para evaluar cada unidad, es necesario multiplicar por el número de veces que se invoca. Para esto en la siguiente tabla se muestran dichos valores que son constantes a lo largo de las unidades.

Tabla 5.4. Latencia por unidad de hardware testeada durante el DSE para convolución por stream.

Configuración	Consumo (%)	Latencia (ciclos)	Contador de ejecuciones
DW10,DI1,KC3	0.26746932515337424	512.0	16384
DW10,DI2,KC3	0.26746932515337424	512.0	16384
DW12,DI1,KC3	0.7583333333333333	512.0	16384
DW12,DI2,KC3	0.7583333333333333	512.0	16384
DW16,DI1,KC3	0.7583333333333333	512.0	16384
DW16,DI2,KC3	0.7583333333333333	512.0	16384
DW4,DI1,KC3	0.119739263803681	512.0	16384
DW4,DI2,KC3	0.119739263803681	512.0	16384
DW8,DI1,KC3	0.20237730061349693	512.0	16384
DW8,DI2,KC3	0.20237730061349693	512.0	16384

Si se realiza una multiplicación básica de la latencia para todas las ejecuciones, el resultado es de:

$$latencia\ total = 512 \times 16384 = 8388608\ ciclos$$

Esto indica que las aproximaciones anteriores realizan la operación con un mayor consumo y en un mayor tiempo.

Ahora bien, por medio de una operación básica, en una imagen de 512x512 píxeles como las probadas en este estudio, se puede obtener la cantidad de ciclos de reloj por píxel.

$$ciclos\ por\ píxel = \frac{latencia\ en\ co - sim}{total\ de\ píxeles} \quad [17]$$

Tabla 5.5. Latencia en ciclos por píxel para los aceleradores presentes en FAL.

	Kernel 3	Kernel 5
Convolución Depthwise	$\frac{3920541}{512 \times 512} = 14,9 \text{ ciclos/píxel}$	$\frac{8038723}{512 \times 512} = 30,6 \text{ ciclos/píxel}$
Convolución Pointwise	$\frac{4468711}{512 \times 512 \times 3} = 5,7 \text{ ciclos/píxel}$	-
Convolución Referencia	$\frac{8388608}{512 \times 512} = 32 \text{ ciclos/píxel}$	-

Por lo tanto, se ofrece una unidad que por medio del uso de una técnica de line buffers, logra un procesamiento mayor de píxeles por ciclo de reloj, que además incluye la funcionalidad de realizar operaciones con un kernel de 5.

5.3 Pruebas de sobre la calidad de las imágenes obtenidas

Se realizaron estudios en base a métricas de error cuadrático medio (MSE), Peak Signal to Noise Ratio (PSNR) y del índice de similitud estructural (SSIM) las cuales se encuentran en la sección 4.7.1 descritas. Estas comparan la imagen obtenida con una referencia en software.

5.3.1 Calidad de imagen obtenida en Convolución Depthwise

En la convolución depthwise, se trabajó en un canal de la imagen. La biblioteca de OpenCV en C++, que se incorpora en el desarrollo de este trabajo, incluye un método de filtrado que aplica la operación de convolución y produce una imagen de referencia, igual a la salida deseada del sistema. En este caso ambas imágenes se les aplicó un kernel de tipo filtro gaussiano que aplica un desenfoque a la imagen.

En la siguiente figura se ejemplifica los distintos resultados de la convolución aplicado a la imagen de Lenna para la convolución depthwise. La referencia corresponde al resultado de la operación en software.



Figura 5.3. Resultados de la convolución depthwise según el tamaño de dato para un tamaño de parte entera de 2.

Aunque estas tres métricas son útiles para evaluar la calidad de la operación y del hardware diseñado, el propósito de este análisis es demostrar la capacidad del hardware y las opciones que ofrece, de manera que, en comparación con una operación de referencia, demuestre estar a la altura. Esto incluso podría implicar la exploración de diversas configuraciones que podrían resultar en una mayor precisión y calidad. Decisiones sobre un hardware y otro deben apoyarse además en las métricas de recursos y latencia presentadas.

5.3.1.1 MSE

A partir de esta métrica se logra evidenciar que las diferencias entre unidades de aceleración según el nivel de precisión que se desee alcanzar van a ir disminuyendo y que, a mayor tamaño de dato, con mayor espacio para parte fraccionaria el error va a ser menor.

Tabla 5.6. Error cuadrático medio en convolución depthwise para cada unidad según el tamaño de dato.

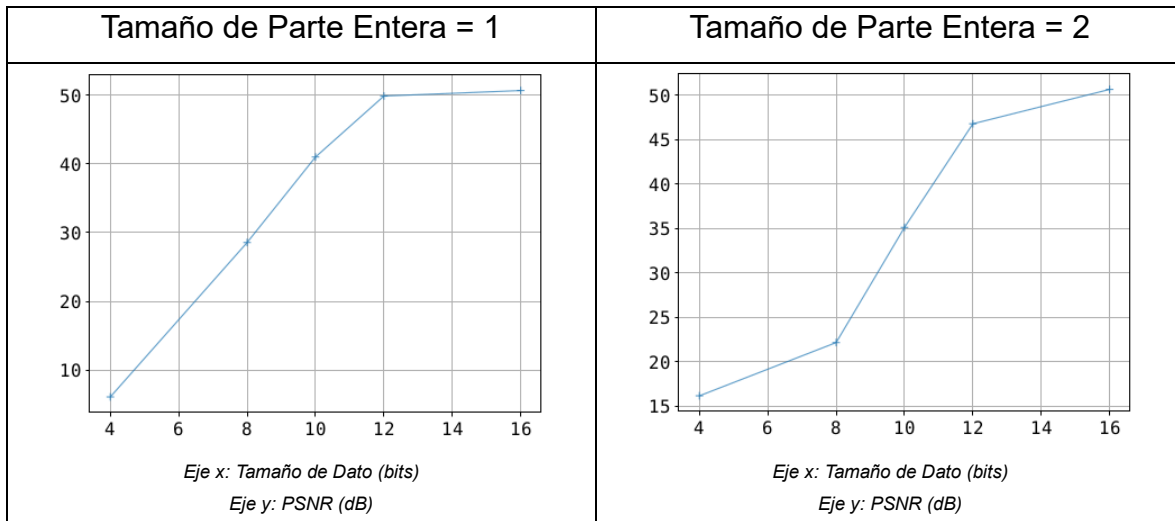
Datatype Width (bits)	Datatype Integer (bits)	MSE
4	1	13087.6
4	2	1299.02
8	1	76.8054
8	2	342.464
10	1	4.30078
10	2	16.7549
12	1	0.559113
12	2	1.13658
16	1	0.46508
16	2	0.46508

El comportamiento mostrado aquí es el esperado. A medida que aumenta la cantidad de bits para cada dato, también lo hace la precisión píxel a píxel en la comparación. El error acumulado entre los datos para un dato de 16 bits es más cercano a cero. Como se observa, cuando la parte entera es 2, hay menos precisión, pero para un tamaño de dato de 16, la diferencia es nula, lo que sugiere que, a partir de ese punto, las diferencias píxel a píxel con arquitecturas de mayor tamaño de dato serán mínimas.

5.3.1.2 PSNR

Generalmente, al evaluar el PSNR, se debe tener en cuenta el contexto de la aplicación, ya que un valor puede ser considerado aceptable o inaceptable según diferentes situaciones. En las aplicaciones abordadas aquí, donde el proyecto se centra en gran medida en la computación aproximada, se toma como referencia un estándar académico que considera un PSNR menor a 20 dB como inaceptable y mayor como adecuado o bueno, dependiendo del contexto [60]. Estos valores fueron consultados además a partir de la asesoría de Luis León que revalidó esta pesquisa.

Tabla 5.7. Peak Signal to Noise Ratio para convolución depthwise

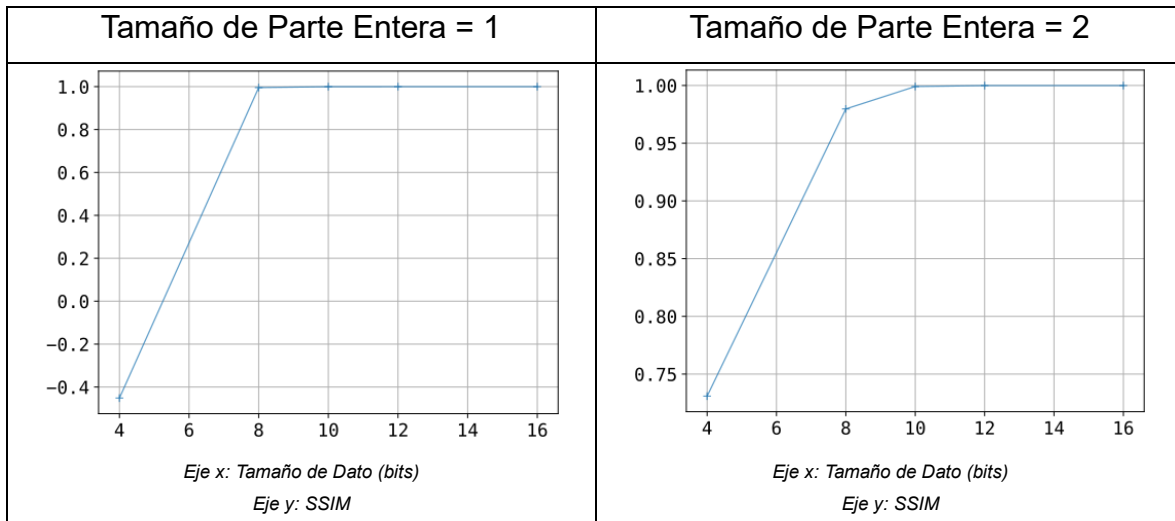


Como se observa en la imagen, se aprecia que las imágenes obtenidas mediante la convolución depthwise a partir de 8 bits presentan resultados aceptables. Ambos casos, con diferentes tamaños de la parte entera, exhiben diferencias en el valor de PSNR para cada tamaño de dato. Aunque en ambos casos se obtienen resultados aceptables según lo establecido previamente, destaca el hecho de que, para un tamaño de dato que ofrece más espacio para la precisión en su parte fraccionaria, se alcanzan mejores resultados, y, además, consumen prácticamente la misma cantidad de recursos.

5.3.1.3 SSIM

Ahora bien, para el SSIM, donde se busca evaluar la calidad de la imagen obtenida con base en un sistema que evalúe la calidad en términos visuales, se pueden revisar los resultados presentes en la Figura 5.3. Resultados de la convolución depthwise según el tamaño de dato para un tamaño de parte entera de 2. En esta figura, se expresa de forma visual cómo entre imagen e imagen se observan diferencias respecto a la referencia, sobre todo en 4 y 8 bits.

Tabla 5.8. Índice de Similitud Estructural para convolución depthwise



Como se observa, para ambos casos, a partir de 10 bits es prácticamente indiscutible tanto a nivel visual como gráfico que la imagen resulta ser una copia de la referencia en software. En el caso de la imagen con la parte entera del dato igual a 1, cuando el tamaño de dato es 4, la imagen muestra una métrica muy baja de SSIM, indicando que la imagen es prácticamente inutilizable. Esto puede deberse a diversos factores, pero principalmente a problemas de implementación. Debido a que utilizar 4 bits es, de por sí, sumamente inusual, se tomará como una recomendación trabajar en futuros proyectos con el acelerador. Nuevamente, debido a que con un tamaño de la parte entera del dato igual a 2, se disponen de menos cifras para representar la parte fraccionaria y, por lo tanto, existe menos precisión, la imagen no es 100% idéntica a la referencia. Se observan pequeños detalles en la imagen que no representan de manera fiel las texturas y los cambios de color.

5.3.2 Calidad de imagen obtenida en Convolución Pointwise

En el caso de la convolución pointwise, se trabajó con las imágenes presentadas en la sección 4.7.1.4 que corresponden a imágenes en 3 canales RGB que se les aplicó una conversión a 1 canal blanco y negro. Se validó su calidad a partir de un método de transformación presente en la librería de opencv que toma como entrada un vector con los valores de conversión. Este aplica la misma operación que si se

tratase de un kernel y una convolución pointwise, pero está programado sobre esta librería a nivel de software. En este caso toma los valores estándar para una conversión a blanco y negro, igual al kernel que se está aplicando a nivel de convolución.

En la siguiente figura se ejemplifica los distintos resultados de la convolución aplicado a la imagen de Lenna para la convolución pointwise. La referencia corresponde al resultado de la operación en software.

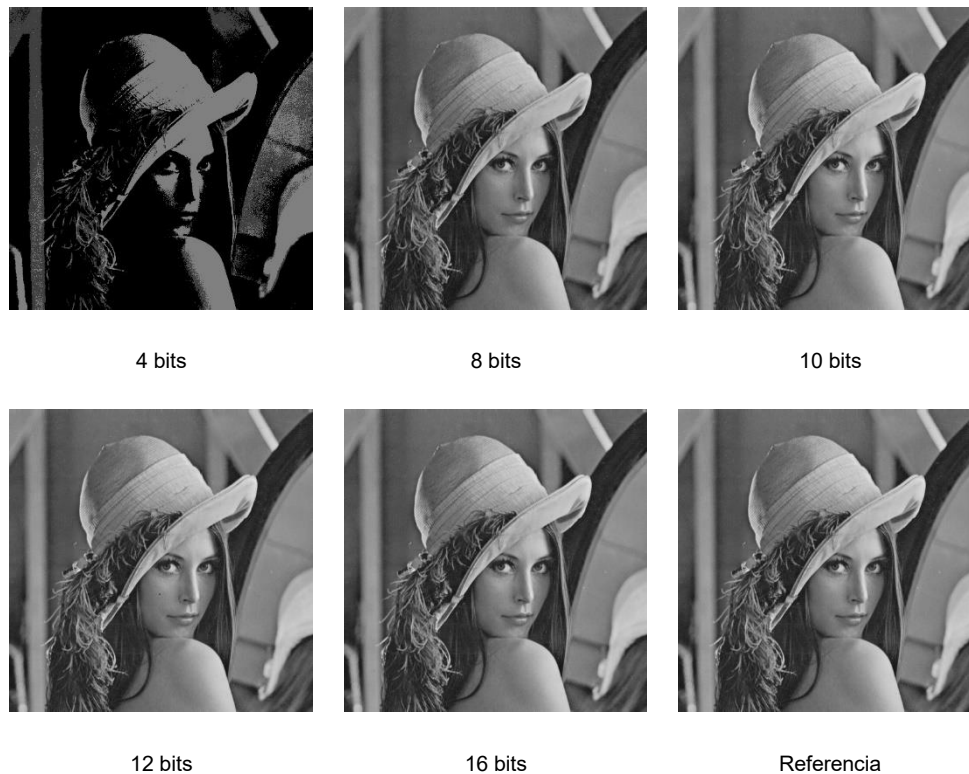


Figura 5.4. Resultados de la convolución pointwise en distintos tamaños de dato para un tamaño de parte entera de 2.

5.3.2.1 MSE

En el caso de la convolución pointwise, el proceso de evaluación sigue el mismo esquema que la convolución depthwise, realizando una comparación píxel a píxel de la diferencia media.

Tabla 5.9. Error cuadrático medio en convolución pointwise para cada unidad según el tamaño de dato.

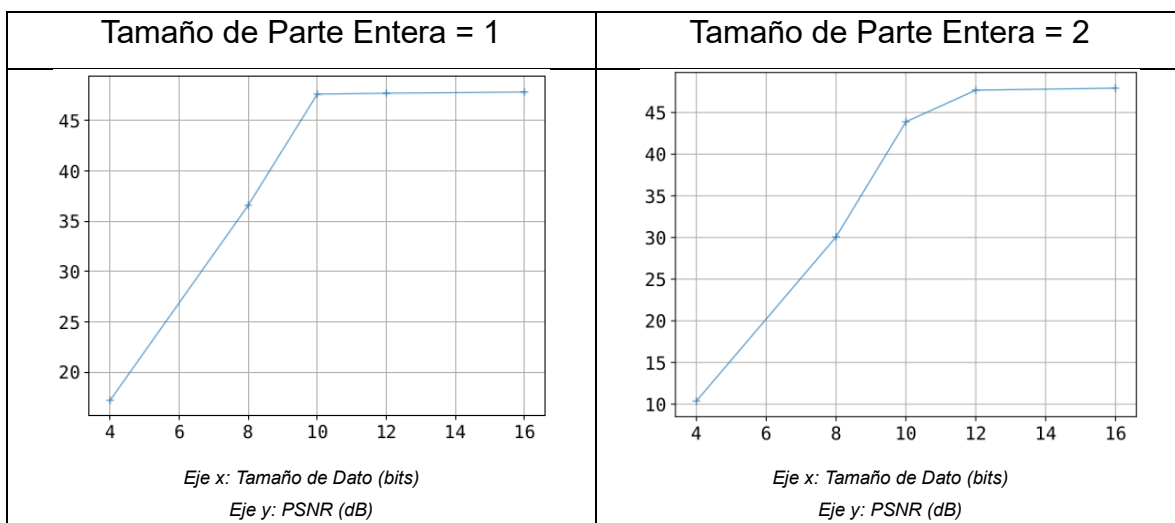
Datatype Width	Datatype Integer	MSE
4	1	1426.49
4	2	5291.06
8	1	13.3283
8	2	58.6285
10	1	0.98959
10	2	2.4345
12	1	0.959583
12	2	0.965179
16	1	0.933445
16	2	0.913139

Se observa un comportamiento similar, donde a mayor precisión en los datos, menor es el error. Se obtienen resultados óptimos y menores diferencias entre los valores teóricos y los obtenidos con el hardware al proporcionar mayor espacio para la parte fraccionaria del dato. Estas conclusiones son visualmente evidentes a través de los resultados presentados a nivel de imagen anteriormente.

5.3.2.2 PSNR

El comportamiento del PSNR en el acelerador pointwise es bastante similar al del acelerador depthwise. De la misma forma se evalúa cada tamaño de dato según la parte entera de estos.

Tabla 5.10. Peak Signal to Noise Ratio para convolución pointwise

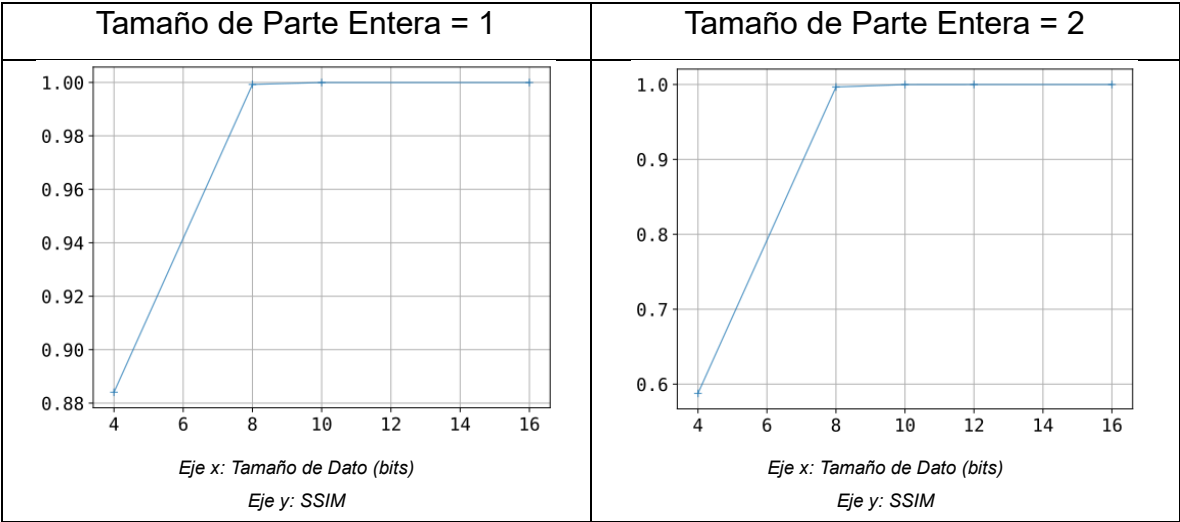


El tamaño del dato a partir de 8 bits presenta resultados considerados aceptables u óptimos según el contexto general. A partir de 12 bits, en ambos casos, el PSNR se estabiliza alrededor de 47 dB. Esta métrica se respalda en el MSE, que, como se observa, tiene un comportamiento similar al presentado en este caso. Un PSNR en ese rango se considera alto; sin embargo, existen estándares para los cuales esto puede no ser suficiente, y se debe evaluar este aspecto al elegir una unidad para DL. Se recomienda considerar aspectos como el ruido en la imagen, los colores presentes e incluso otras técnicas de aproximación de los datos. Aunque se realizaron pruebas con varias imágenes, es importante tener en cuenta, por ejemplo, que para las imágenes "baboon" y "barbara", el color no está presente, por lo que las pruebas con otras imágenes más coloridas podrían proporcionar resultados más fidedignos para cualquier contexto.

5.3.2.3 SSIM

Para evaluar la calidad visual de las imágenes que se procesan por medio de este acelerador nuevamente se utilizó el índice de similitud estructural.

Tabla 5.11. Índice de Similitud Estructural para convolución pointwise



En este los resultados resultan prácticamente iguales y al ser una métrica que busca cuantificar la percepción visual humana, se confirma a partir de las imágenes presentadas al inicio de esta sección. No se denotan mayores cambios entre la

referencia y las imágenes a partir de 8 bits y el comportamiento de 4 bits continua con el mismo patrón en todas las pruebas que se realizaron sobre los aceleradores, aunque bien, este no resulta de mayor interés por su poca aplicabilidad en el contexto de DL.

5.4 Pruebas de los aceleradores en FPGA

Una vez que se realizaron las distintas pruebas para determinar la capacidad operativa de los aceleradores, se procedió a realizar pruebas directamente sobre una FPGA. Para este caso se utilizó la Kria KV260 que es una FPGA de mayor gama que la Artix7, debido a que las pruebas propuestas corresponden a una comparación directa con el acelerador de las librerías de Xilinx el cual por su diseño no se puede correr en la Artix7 que es de menor gama.

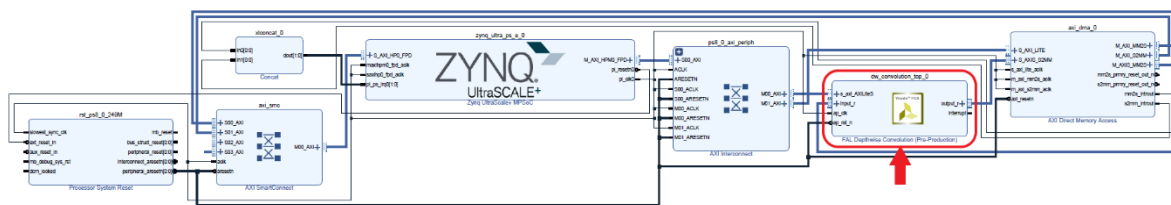


Figura 5.5. Configuración de la FPGA para el procesamiento de imágenes.

El segundo bloque de derecha a izquierda corresponde al acelerador depthwise que se desarrolló e incluyó en el sistema. Este es sustituible con el acelerador de Xilinx. Las pruebas se realizaron para tener un marco de comparación a nivel de recursos y latencia respecto al acelerador de Xilinx además del consumo energético asociado a la lógica asociada. El tamaño de dato para las pruebas fue de 8bits, con 1 bit entero.

Tabla 5.12. Latencia en ms para los distintos tamaños de imagen.

Resolución de imagen	Xilinx Filter 2D - 8 bit	FAL - 8 bit
32x32	1.36ms - 9.92us	1.35ms - 17.7us
64x64	1.35ms - 12.6us	1.43ms - 15.8us
96x96	1.35ms - 13.2us	1.70ms - 10.8us
128x128	1.35ms - 13.3us	2.06ms - 8.89us

160x160	1.35ms - 12.0us	2.54ms - 13.1us
256x256	1.40ms - 13.8us	4.60ms - 12.5us
320x320	1.44ms - 9.23us	6.50ms - 10.1us
512x512	1.61ms - 7.30us	14.8ms - 2.91us
768x768	1.93ms - 12.0us	31.8ms - 8.72us
1024x1024	2.40ms - 7.60us	55.6ms - 4.71us

Primeramente, es visible que la latencia asociada al filtro de Xilinx resulta menor. Si bien esto representa una ventaja no deja de ser importante mencionar que esta es únicamente implementable en FPGAs de alta gama. Además, es también considerando el consumo de recursos, que se puede concluir ventajas de uno sobre el otro.

Tabla 5.13. Consumo de Recursos para distintas

Recursos	BRAM	DSP	FF	LUT	Frecuencia Máxima
FAL - 8 bit, k = 3, s = 1	10	3	1675	4118	228.78 MHz
Xilinx - 8 bit, k = 3, s = 1	6	0	5739	8725	100.44 MHz

El acelerador implementado, representa una opción con un consumo mucho menor de FFs y LUTs que en ocasiones puede resultar necesario. Además, se debe considerar que añadir la capacidad de tomar kernels de distintos tamaños al igual que el valor de stride, predefine aspectos de la lógica programable que aumentan la latencia asociada y el consumo mayor en BRAM y DSPs. El valor de frecuencia para este contexto de los aceleradores de hardware se refiere a la velocidad del reloj en la que el circuito del acelerador puede funcionar de manera confiable y sin errores. Una frecuencia más alta significa que el acelerador puede realizar más operaciones en un período de tiempo determinado, lo que generalmente se traduce en un rendimiento más rápido de las aplicaciones o algoritmos que utilizan el acelerador.

Como parte de las pruebas realizadas en la FPGA, se evaluó el consumo energético de realizar estas operaciones en el procesamiento de imágenes de distintos tamaños. Primeramente, el consumo energético estático, es decir cuando no se está realizando activamente ninguna operación, (la alimentación necesaria para mantener la configuración de los elementos lógicos) correspondió a 0.301W. Por otro lado, el consumo energético dinámico, producto de la actividad de la FPGA realizando operaciones (la energía al ejecutar algoritmos o realizar cálculos que consume energía adicional debido al cambio dinámico de estados lógicos, la conmutación de transistores y la transferencia de datos) tuvo un valor de 0.140W. Esto da como resultado de 0.401W de consumo energético total de la lógica programable mientras que el consumo total de la FPGA y todos sus sistemas interconectados da como resultado un consumo de 2.864W. Este resultado, al compararse con el consumo energético de otros sistemas, como por ejemplo una GPU, puede ser hasta 15 veces menor [61] para una misma tarea. Esto demuestra la complejidad de implementar esta tecnología en una variedad de aplicaciones críticas, en contraste con sistemas que requieren una cantidad considerablemente menor de energía.

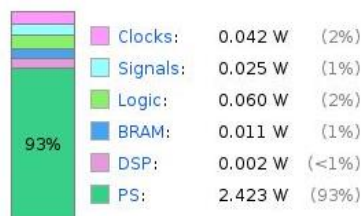


Figura 5.6. Consumo energético de la FPGA en operación de convolución de una imagen.

5.5 Pruebas de la red en simulación

Se realizaron pruebas para determinar el grado de precisión con el que se realiza una clasificación. En este punto es importante definir algunas características del resultado y la forma en que este se analiza. Al tratarse de una clasificación la precisión se mide de la siguiente forma en donde predicciones totales corresponden a aquellas correctas e incorrectas.

$$\text{precisión} = \frac{\text{predicciones correctas}}{\text{predicciones totales}} \quad [18]$$

El framework y la programación de esta red neuronal han sido diseñados de manera que puedan obtenerse los resultados de manera óptima. Esto se refleja en la capacidad de generar un valor de precisión según la cantidad de clasificaciones correctas realizadas, a partir de un archivo binario que se utiliza posteriormente para evaluar la precisión del modelo, como se explicó previamente. No se generaron resultados visuales, como una visualización de imágenes y sus etiquetas, ya que esto no estaba dentro del interés del análisis. Aunque técnicamente es posible realizar esta tarea, el enfoque principal estaba en evaluar otras métricas, como, nuevamente, la precisión de las predicciones. Los resultados obtenidos fueron suficientes para cumplir con los objetivos y evaluar la calidad de las predicciones de manera efectiva.

El diseño de las pruebas se realizó de la siguiente manera. Realizado un previo análisis en Python, se determinó los rangos en los cuales los datos dentro de la red operaban.

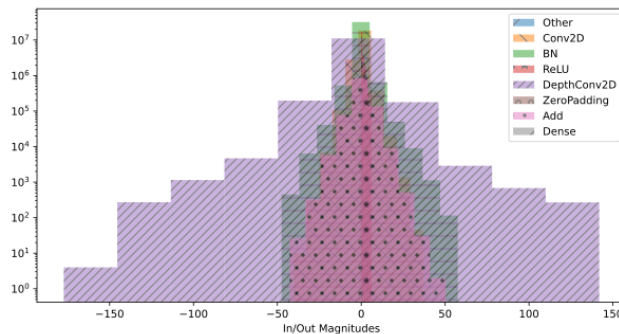


Figura 5.7. Histograma de actividad de la magnitud de los datos presentes en la red.

De esta forma es posible determinar el tamaño de dato al que pueden operar las distintas capas, lo cual si bien, a nivel de recursos no va a representar un mayor cambio como se analizó previamente en los aceleradores diseñados, si lo hará en latencia, por lo que a un tamaño de dato menor la red será más rápida y las pruebas

se buscaron hacer para buscar el menor tipo de dato que todavía fuera lo suficientemente preciso.

Se diseñó la red convolucional para que todas las capas pudieran someterse a estas 'pruebas de cuantización', sin embargo, las capas de interés son aquellas que realizan la convolución separable por lo que se iteró sobre estas capas para encontrar esa mejor combinación con un valor de precisión mayor a 75%, tomado como criterio para considerar un sistema capaz de realizar clasificaciones adecuadas. Este valor fue indicado por León, y definido según su criterio experto.

Inicialmente, se sustituyeron las capas correspondientes con los aceleradores, y se obtuvo un valor de precisión que se comparó con una versión sin los aceleradores diseñada completamente en software según algoritmos de C++. Esto arrojó un valor de **86%** de precisión, para un total de 50 muestras, según la base de datos de ImageNet reducida, conocida como *Imagenette*. Este valor para software correspondió igualmente a 86%. Las pruebas se realizaron con un tamaño de dato de 24bits con 8bits de su parte entera, que corresponde al límite superior para el cual los resultados siempre van a ser iguales con un tamaño de dato superior.

Seguidamente se buscó obtener la mejor combinación que, con el menor tamaño de dato arrojara un valor de precisión mayor a 75%. Los resultados de estas pruebas se presentan a continuación.

Tabla 5.14. Resultados de pruebas realizadas para determinar la combinación óptima del tamaño de dato.

POINTWISE		DEPTHWISE		Precisión
Tamaño de Dato	Parte Entera	Tamaño de Dato	Parte Entera	
18	5	14	7	57%
18	5	16	7	54%
18	5	18	7	57%
18	5	20	7	60%
18	5	22	7	57%
20	5	14	7	54%

22	5	14	7	51%
22	5	22	7	54%
22	6	22	7	91%
18	6	14	7	54%
20	6	14	7	85%
18	6	16	7	63%
20	6	16	7	88%

A menor tamaño de dato, se pueden cargar más bits dentro de un mismo bus, lo que se traduce en un menor volumen de procesamiento y, por ende, en la menor latencia mencionada. Las pruebas se llevaron a cabo de manera iterativa, probando combinaciones una a una hasta encontrar aquella que arrojara los mejores resultados. A partir de la Figura 5.7. Histograma de actividad de la magnitud de los datos presentes en la red., se observó que, para representar adecuadamente los datos y evitar pérdida de información en el proceso, cada una de las capas de convolución requería un tamaño de entero apropiado. Gracias al histograma, es posible obtener una aproximación inicial para comenzar las pruebas, sin embargo, a partir de ese punto, es necesario revisar el tamaño de la parte fraccionaria, que constituye la parte central de las pruebas, aumentándola gradualmente. Se determinó que con un tamaño de dato de 20bits con 6bits de su parte entera para el acelerador de convolución pointwise y un tamaño de dato de 14bits con 7bits de su parte entera para el acelerador de convolución depthwise, se alcanzaba una precisión del modelo de 85%, sumamente cercana a la precisión de 86% que se determinó en el rango de operación superior.

Esto demuestra que, dentro de la red, al realizar la sustitución en las capas correspondientes, los aceleradores son capaces de responder a un caso real de clasificación, dando pie a la utilización de otros aceleradores para una implementación total en una FPGA en un sistema aislado y funcional.

5.6 Requerimientos establecidos vs requerimientos cumplidos

Como parte fundamental de la metodología de diseño implementada, se parte de una serie de métricas necesarias que se deben cumplir, las cuales se establecieron

en la Tabla 4.2. Con apoyo en esta tabla se realiza la siguiente síntesis en base a los resultados obtenidos para las distintas pruebas a las que se sometieron los aceleradores.

Tabla 5.15. Valor obtenido vs el valor meta para las métricas de los aceleradores.

	Número de Necesidad	Métrica	Importancia	Unidades	Valor Marginal	Valor Ideal	Valor Obtenido
1	1	Latencia**	3	ciclos/píxel	[25,32]	<25	[5.7,32]
2	2	Consumo de LUTs**	3	% utilización	[15,20]	<15	[8,12]
3	2	Consumo de DSP48E**	2	% utilización	[3, 4]	<3	2
4	2	Consumo de BRAM_18K**	2	% utilización	[15, 30]	<15	[4,5]
5	2	Consumo de Flip-flops**	3	% utilización	[11,20]	<11	[2,3]
6	3	Compilable en C++	3	Binario	True	True	True
7	3	Sintetizable a RTL	3	Binario	True	True	True
8	3	Cosimulable	3	Binario	True	True	True
9	3	Empaquetable a FPGA	3	Binario	True	True	True
10	4	Precisión del Modelo	2	% de precisión	[75,85]	[85,100]	86%
11	5	Tamaño de Bus	2	bits [rango]	[4, 64]	[4, 512]	[4, 512]
12	5	Tamaño de Dato	3	bits [rango]	[4, 8]	[4, 32]	[4, 32]
13	5	Tamaño de imagen	2	píxel	512x512	1024x1024	1024x1024
14	5	Tamaño Máximo de Kernel	2	valor	3x3 dw 1x1x3 pw	5x5 dw 1x1x16 pw	5x5 dw 1x1x16 pw
15	5	Máximo Stride	2	valor	1	2	2

**Pruebas realizadas en un marco de referencia de un kernel de tamaño 3.

En verde se encuentran los resultados que se obtuvieron cumpliendo las metas establecidas para el mismo. En amarillo como bien se aprecia, los resultados también fueron satisfactorios, pero en este caso, se abre la posibilidad de realizar pruebas adicionales para poder establecer rangos de operación incluso mejores a los reportados para el presente documento.

Ahora bien, para cada una de las métricas considérese lo siguiente:

1. Latencia: En lo que respecta a latencia, las aproximaciones anteriores alcanzaron un procesamiento 30 ciclos por píxel cuando se trabajó condiciones con un kernel de 3x3. Para este caso, en el convolucionador desarrollado se redujo hasta la mitad este valor mientras que para un kernel de 5x5 se obtuvo un procesamiento de 32 ciclos por píxel. Este resulta superior, pero para un caso donde se añade la capacidad de manejar kernels de mayor tamaño y flexibilizar la unidad de procesamiento.
2. Consumo de LUTs: En lo que respecta este rubro, a diferencia de los resultado con la solución propuesta, la aproximación existente en FAL presentaba un consumo de LUTs dentro de un rango que se definió como el marginal para la evaluación. Los resultados que se obtuvieron arrojaron valores dentro de un rango sumamente reducido, pero aun así menor debido al tipo de algoritmo utilizado.
3. Consumo de DSP48E: Similar al caso de LUTs, los DSPs presentan el mismo comportamiento.
4. Consumo de BRAM_18K: Este sí resulta un caso particular dentro del análisis de los resultados obtenidos en lo que respecta a recursos. La aproximación con la que se tomaron ambos casos difería en el uso de memoria. Mientras que una aproximación toma toda la imagen y la almacena para procesarla, la otra toma pequeñas ventanas de la imagen, obtiene lo que necesita y si esa ventana ya no la requiere la desecha para pasar con la siguiente. De esta forma el consumo en el primer caso descrito de memoria va a ser superior.

Considerando esto se definieron los márgenes en base a un consumo no mayor al 30% de la placa, para el cual en la realidad se está gastando menos de un 5%.

5. Consumo de Flip-flops: Similar al caso de LUTs y DSPs, los Flip-flops presentan el mismo comportamiento.
6. Compilable en C++: La unidad descrita en hardware por medio de high-level synthesis compila de forma exitosa sin presentar errores en el proceso
7. Sintetizable a RTL: La aplicación de desarrollo es capaz de sintetizar el hardware de forma exitosa y así obtener un modelo en RTL de la misma para implementar en FPGA, sin presentar errores en el proceso
8. Cosimulable: Es posible verificar y se comprueba que a nivel de software y los resultados de simulación en hardware coinciden y se ejecutan de la misma manera.
9. Empaquetable a FPGA: El diseño puede ser implementado y ejecutado en una FPGA utilizando los recursos disponibles en la FPGA y respetando las limitaciones de hardware y la arquitectura específica de la FPGA objetivo.
10. Precisión del Modelo: El modelo realizado en base a la red MobileNetV2 presenta una precisión de 86% queriendo decir que para el 86% de las veces, puede clasificar correctamente la imagen que se le señala.
11. Tamaño de Bus: Se realizaron pruebas para verificar el tamaño de bus que se podría soportar. Dado que las pruebas todas se realizaron en imágenes estándar como la de 'lenna', una prueba para un tamaño de bus mayor a este no fue realizada. Siguiendo el mismo patrón y en base a los bajos consumos obtenidos la

expectativa es que este valor pueda ser mucho mayor.

12. Tamaño de Dato: El tamaño de dato superó las expectativas definidas, y los datos se definieron de hasta 32bits dentro de las pruebas realizadas porque resultó ser lo necesario ya que las imágenes no presentaban mayores mejoras hasta ese punto. Sin embargo, para aplicaciones particulares, donde esto sí sea de interés un estudio con datos de mayor tamaño y por ende mayor precisión, son recomendables para confirmar que se pueden usar valores aún mayores.
13. Tamaño de imagen: Se realizaron pruebas muy puntuales durante el desarrollo en donde se introdujeron imágenes de hasta 1024x1024. Nuevamente, por motivos de la investigación, no resultaba de interés validar tamaños mayores por lo que se insta a realizar estas pruebas en trabajos futuros.
14. Tamaño Máximo de Kernel: Dado que no existían referencias de kernels de un tamaño superior a 3x3, se limitaron las pruebas a validar si el acelerador propuesto podía realizar trabajos con un kernels de 5x5. Este valor se alcanzó, por lo que la meta fue cumplida superando las aproximaciones tanto de desarrollos anteriores como las opciones brindadas por las librerías internas de Xilinx.
15. Máximo Stride: Se alcanza un stride de hasta 2 dentro de la convolución lo cual resulta en un parámetro que no se había alcanzado en ninguna aproximación anterior que conformase alguna de las librerías.

5.7 Análisis Financiero

El análisis presentado se deriva de la evaluación de varios componentes considerados durante el proyecto. Es importante señalar que no se abordaron en su totalidad muchos detalles altamente específicos. El objetivo principal de este análisis es proporcionar al lector una visión general de lo que puede representar económicamente este proyecto. Aunque se tomaron en cuenta varios aspectos, no se profundizó exhaustivamente en todos los detalles, ya que la intención es ofrecer una perspectiva general. En la sección 3.7.1 se mencionó que este análisis se realizaba además desde tres perspectivas importantes que podrían considerarse a la hora de valorar su aplicabilidad futura en un proyecto.

El proyecto se valora en un escenario base de 5 años que plantea que en ese horizonte de tiempo la tecnología va a ser superada. Se incluyen factores varios propios de un análisis financiero de tipo operativo, aunque muchos de estos por la particularidad del proyecto no se consideraron, tales como ingresos y costo de ventas. Además, se subdividieron los gastos en OPEX y CAPEX alineados con el tipo de análisis. Algunas suposiciones fueron tomadas, las cuales se especifican para cada caso.

El análisis se realiza dentro del marco general del proyecto, y no de la unidad que durante este informe se trabaja. Es decir que se debe asumir un proyecto visto desde la integración total de todas las aristas que se trabajan para desarrollar el sistema de visión completo y no únicamente la parte de la unidad encargada de la operación de convolución del proyecto.

5.7.1 Caso 1: Costo del proyecto de ejecutarse desde cero

La primera aproximación para el análisis es el costo del proyecto a ejecutarse desde cero, es decir sin una existencia previa de alguna aproximación que logre hacer lo que en este informe se documenta. Como se aprecia en la tabla a continuación, los gastos operativos consisten en la mayor parte de la inversión en el proyecto.

Estos gastos se distribuyen de la siguiente manera. Se asume como un proyecto que debería ser tomado por un desarrollador senior preferiblemente. Esto debido a

que, en caso de tomarse por parte de un desarrollador junior, el nivel de apoyo que se requeriría por parte de un senior elevaría el costo a básicamente tener dos ingenieros. Si bien, durante los 6 meses del proyecto, este se asumió por parte de un desarrollador junior, no fue sin el apoyo constante de un desarrollador senior que se logró llevar a cabo. Se consultó al respecto a Luis León, asesor técnico, y se concluyó que, estableciendo este proyecto como prioridad dentro de la carga de trabajo de una persona en su nivel académico, se puede llevar a cabo en 3 meses, donde se asume un salario de 2 millones de colones neto. A esto se le hace una suma de las cargas sociales por parte del patrono. Seguidamente la FPGA a utilizar para el proyecto que fue la Artix7 de Xilinx, cuyo valor se obtiene de una búsqueda en el mercado costarricense. El gasto de electricidad, aunque claramente se debe tomar como una estimación es a través de una consulta a la página de Repsol que indica que un uso continuo de un computador ronda los 50kWh mensuales que al valor por kWh del mercado costarricense representa aproximadamente 70 colones por kWh.

Ahora bien, en base al horizonte de tiempo de 5 años explicado anteriormente, se valora que cualquier tarea de mantenimiento se puede realizar de forma más breve y por medio de mano de obra no tan calificada como la de un desarrollador senior, por lo que el gasto en salario se reduce en ese rango de tiempo. Asumiendo un mantenimiento del sistema cada 6 meses en el cual se trabaje durante 15 días, los gastos quedan reflejados según lo que se observa en la Tabla 5.16.

Tabla 5.16. Costos del Sistema para Escenario Base

	Año 0	Año 1	Año 2	Año 3	Año 4	Año 5
INGRESOS	-	-	-	-	-	-
COSTO DE VENTAS	-	-	-	-	-	-
GASTOS						
<u>Gastos Operativos</u>						
Salario de Desarrollador Senior	- 6 000 000.00					
Cargas Sociales	- 2 520 000.00					
FPGA	- 94 050.00					
Mantenimiento (Serv. Prof Desarrollador)		- 800 000.00	- 800 000.00	- 800 000.00	- 800 000.00	- 800 000.00
Gasto de Electricidad	- 9 000.00	- 3 000.00	- 3 000.00	- 3 000.00	- 3 000.00	- 3 000.00
Gasto de Internet	- 72 810.00	- 24 270.00	- 24 270.00	- 24 270.00	- 24 270.00	- 24 270.00
Utilidad Operativa	- 8 695 860.00	- 827 270.00	- 827 270.00	- 827 270.00	- 827 270.00	- 827 270.00
<u>Gastos Financieros</u>						
Impuestos	- 2 173 965.00	- 206 817.50	- 206 817.50	- 206 817.50	- 206 817.50	- 206 817.50
Utilidad Neta	- 10 869 825.00	-1 034 087.50	-1 034 087.50	-1 034 087.50	-1 034 087.50	-1 034 087.50
Cambios en el CT						
Depreciación		- 108 000.00	- 108 000.00	- 108 000.00		
Inversiones						
Equipo de Computación	- 324 000.00					
FLUJO	- 11 193 825.00	-1 142 087.50	-1 142 087.50	-1 142 087.50	-1 034 087.50	-1 034 087.50

5.7.2 Caso 2: Costo del proyecto implementando el sistema desarrollado

Este considera que el sistema en FAL puede ser utilizado por cualquier grupo, empresa o persona con necesidad de una tecnología de este tipo. Dicho interesado hace uso del sistema en FAL para la aplicación que requiera.

El segundo caso parte de un supuesto muy importante respecto al proyecto, y es la correcta documentación de este. Si bien, este se puede replicar sin tener que realizar un desarrollo desde cero, es importante partir de una base sólida que permita elevar el sistema para que diversos perfiles técnicos puedan trabajar en

esto. El perfil de un desarrollador junior dentro del área de sistemas embebidos contempla la capacidad de leer y comprender distintos sistemas siempre y cuando exista una buena documentación. Si esto es así, el mayor cambio dentro de la primera aproximación es el reemplazo del desarrollador senior con uno junior, el cual representa un salario mucho menor y un desarrollo en un menor rango de tiempo de hasta dos meses. Si bien, el gasto a lo largo de la ventana de tiempo se mantiene constante en mantenimiento, internet, electricidad, etc. El ahorro en el grado académico que se debe dirigir al proyecto puede resultar representativo.

Tabla 5.17. Costos del Sistema para Escenario con existencia de implementación en FAL.

	Año 0	Año 1	Año 2	Año 3	Año 4	Año 5
INGRESOS	-	-	-	-	-	-
COSTO DE VENTAS	-	-	-	-	-	-
GASTOS						
<u>Gastos Operativos</u>						
Salario de Desarrollador Junior	- 2 400 000.00					
Cargas Sociales	- 1 008 000.00					
FPGA	- 94 050.00					
Mantenimiento (Serv. Prof Desarrollador)		- 800 000.00	- 800 000.00	- 800 000.00	- 800 000.00	- 800 000.00
Gasto de Electricidad	- 9 000.00	- 3 000.00	- 3 000.00	- 3 000.00	- 3 000.00	- 3 000.00
Gasto de Internet	- 72 810.00	- 24 270.00	- 24 270.00	- 24 270.00	- 24 270.00	- 24 270.00
Utilidad Operativa	- 3 583 860.00	- 827 270.00	- 827 270.00	- 827 270.00	- 827 270.00	- 827 270.00
<u>Gastos Financieros</u>						
Impuestos	- 895 965.00	- 206 817.50	- 206 817.50	- 206 817.50	- 206 817.50	- 206 817.50
Utilidad Neta	- 4 479 825.00	-1 034 087.50	-1 034 087.50	-1 034 087.50	-1 034 087.50	-1 034 087.50
Cambios en el CT						
Depreciación		- 108 000.00	- 108 000.00	- 108 000.00		
Inversiones						
Equipo de Computación	- 324 000.00					
FLUJO	- 4 803 825.00	-1 142 087.50	-1 142 087.50	-1 142 087.50	-1 034 087.50	-1 034 087.50

5.7.3 Caso 3: Costo del proyecto ejecutado utilizando alternativas en el mercado

El último caso que analizar resulta en un caso especial en donde se utiliza una tecnología distinta para realizar alguna tarea de clasificación fuera de línea. Y es que, en este caso, se debe tomar en cuenta que la aproximación debe tomar en cuenta varios factores para que haga sentido. Mientras que el consumo energético de una Artix7 en un diseño similar al implementado en este informe está en el orden de los mW, el uso de una GPU requiere de mayor energía para su operación. Gastos de electricidad y hardware resultan más elevados y entran en consideración nuevos subsistemas de alimentación. Si bien una implementación utilizando una GPU puede requerir un menor costo de ingeniería porque es un campo de desarrollo más respaldado por una comunidad, son los gastos adicionales a la implementación de un sistema con esta tecnología lo que lo hacen menos competitivo. Vuelve a ser necesario el análisis del nivel de precisión con el que se podría llegar a operar y si esto resulta en un factor importante.

5.7.4 Consideraciones del estudio económico

Este estudio confirma lo señalado por el estudiante de doctorado Luis León en su tesis [12], que destaca el elevado costo asociado a proyectos de ingeniería de sistemas embebidos. Este campo de especialización, menos explorado en comparación con otros, demanda un nivel técnico más elevado. Actualmente, estos proyectos no se perfilan como alternativas económicamente viables para muchas empresas, dado que implican el uso de tecnologías especializadas de difícil acceso en el mercado. Iniciativas como el framework de FAL buscan acercar estas tecnologías a un público menos especializado, reduciendo así la necesidad de una preparación técnica extensa y permitiendo a un grupo más amplio beneficiarse de estas tecnologías para facilitar desarrollos futuros. Esto, en última instancia, contribuye a hacer el trabajo más rentable y, en muchos casos, prescinde de la necesidad de contar con ingenieros de más alta jerarquía.

Asimismo, este análisis por sí mismo proporciona una evaluación del costo temporal para proyectos que requieren esta tecnología específica, siendo particularmente relevante cuando esta constituye la única alternativa.

5.8 Impacto Ambiental

El proyecto aquí desarrollado no presenta en sí mismo un impacto ambiental considerable a la hora de su desarrollo ya que durante su realización no se considera el uso de materiales o insumos que pudieran tener este tipo de afectaciones en materia ambiental. Sin embargo, el análisis sobre este se realiza en torno a los estudios presentados en los últimos años sobre el impacto que tienen otras tecnologías propias del Deep Learning y su comparación con el uso de FPGAs.

Generalmente, el benchmarking más común para las FPGAs se realiza con las GPUs. Este análisis se fundamenta en tres pilares sobre los cuales se conoce que estas tecnologías tienen los mayores impactos. Primero, el consumo energético asociado; segundo, el impacto por el proceso de fabricación; y, por último, el denominado e-waste (residuos asociados al desecho del hardware). En el presente informe únicamente se abordará el primero pues es el que de mayor interés resulta con los objetivos.

5.8.1 Consumo energético

El enfoque del estudio se centra en el impacto ambiental en el contexto de los modelos de inteligencia artificial, no en aplicaciones generales. Según un informe de OpenAI [62], en la era del aprendizaje profundo, los recursos computacionales necesarios para producir un modelo de inteligencia artificial de alta calidad se han duplicado en promedio cada 3.4 meses, lo que se traduce en un aumento de 300,000 veces entre 2012 y 2018. El estudio busca ilustrar cómo, gradualmente, el software diseñado para ejecutar estos modelos superará la tecnología de hardware, lo que generará una mayor necesidad de utilizar estos dispositivos para satisfacer el rápido progreso y dinamismo en el que se está desarrollando el campo.

En 2019, Strubell [63] presentó una comparación de la huella de carbono de un modelo de aprendizaje profundo versus varias fuentes bien conocidas. Este estudio revela que un modelo con un total de 213 millones de parámetros produce 626,200 libras de CO₂, lo que equivale hasta 5 veces la producción total de CO₂ de un vehículo a lo largo de su vida útil (36,156 lb). Como referencia, modelos entrenados como GPT-3 tienen alrededor de 175 mil millones de parámetros [64].

Estos modelos se han entrenado en grandes centros de datos impulsados principalmente por GPUs como unidades de procesamiento, con indicios de que el uso de estas tecnologías está en aumento en los años futuros [62]. Microsoft [65] presentó un artículo que discute el uso de hardware especializado, como FPGAs, para acelerar redes neuronales convolucionales profundas en centros de datos. Los autores presentan resultados iniciales que muestran un rendimiento prometedor y un bajo consumo de energía en comparación con GPUs de gama alta. El artículo incluye una comparación del rendimiento de clasificación de imágenes y el consumo de energía para diferentes configuraciones de hardware, donde las FPGAs, en una tarea, presentan un consumo de 25 W, mientras que la aproximación con GPUs utiliza 235 W.

Por último, en un estudio presentado por NVIDIA [66], se expone que desplegar modelos de inteligencia artificial para actuar en entornos del mundo real, un proceso conocido como inferencia, consume incluso más energía que el entrenamiento. Se estima que el 80% al 90% del costo de una red neuronal está en la inferencia en lugar del entrenamiento. Como ejemplo, consideremos la inteligencia artificial subyacente en un vehículo autónomo. Las redes neuronales deben entrenarse previamente para aprender a conducir. Después de completar el entrenamiento y desplegar el vehículo, el modelo realiza inferencias de manera continua para navegar, durante todo el tiempo que el vehículo esté en uso. Y es que cabe destacar que, cuantos más parámetros tiene el modelo, mayores son los requisitos energéticos para esta inferencia.

6. Conclusiones

El trabajo aquí presentado constituye dos desarrollos que individualmente presentaron una serie de resultados. Por un lado, se tiene el desarrollo de los aceleradores en HLS y por el otro la implementación de dichos aceleradores en una aplicación de visión por computadora. Las conclusiones por lo tanto se dirigen a estas dos etapas principales en el proyecto.

Inicialmente lo que constituye a la componente académica que envuelve este proyecto, se realizó el estudio de las técnicas de convolución. A partir de esto se determinó que la convolución separable representaba la aproximación de mayor valor a desarrollar durante este trabajo. Tomando en consideración el peso que representaba la necesidad de desarrollar esta dentro de la librería de FAL, así como su eficiencia y flexibilidad, este resultó como el caso de mayor interés. Establecidos los requerimientos para este, se validó la completitud de estos en un 100% una vez desarrollado el concepto.

Una aproximación inicial, puramente en software se implementó para el diseño inicial y validación de los algoritmos. De esta forma se podía comparar tanto en hardware como en software, los resultados obtenidos y tomar un marco de comparación para evaluar la calidad del proyecto. Esta aproximación, considerando el mismo set de datos con el que se entrenaría y desarrollaría la validación del sistema de clasificación, arrojó como resultado un 87% de precisión según los modelos, herramientas y librerías presentes en Python. Los modelos desarrollados en C++ puramente evaluando el desarrollo del algoritmo con la imagen, tuvieron un resultado perfecto respecto a la precisión en cada píxel, considerando que este se evaluó con una precisión de números flotantes.

En lo que resulta a la codificación de los aceleradores en Vivado HLS, para el consumo de recursos, se determinó que los recursos para ambos a partir del uso de Line Buffers eran influenciados únicamente por el tamaño del kernel mas no por el tamaño del dato o su parte entera, por lo que al menos para la aproximación utilizada, una mayor precisión en los cálculos no va a resultar en una compensación

con respecto al consumo de recursos y por lo tanto se puede optar por imágenes con mejores índices de calidad.

Los valores obtenidos para los distintos recursos fueron positivos según la referencia establecida, al alcanzar disminuciones de entre un 33%, como es el caso de la BRAM, y de hasta un 82%, como en el caso de los flip-flops. Si se toma en consideración la comparación directa de un caso como el convolucionador depthwise operando para una misma tarea que la opción de la convolución en Xilinx, la disminución en flip-flops y LUTs es de hasta un 70,8% y un 52,8%, respectivamente, lo cual representa una ventaja en ciertos modos de operación. Si bien, en este caso hay un aumento en el consumo de BRAM y DSPs, es importante considerar que este instrumento que se brinda, en comparación, tiene capacidades más flexibles. Esto demuestra ser una alternativa sólida para aplicaciones o usos donde el hardware presente limitaciones y diversificando las aplicaciones de uso.

En el caso de la latencia de los aceleradores, el fenómeno se repite, en donde es el tamaño del kernel el que determina la cantidad de ciclos de reloj que se tarda en operar la imagen. De esta manera queda en evidencia que hacer uso de un acelerador configurado para mayor precisión no es motivo de una menor mayor latencia. Tanto para la convolución depthwise como pointwise, el factor que determina los ciclos de reloj es el tamaño del kernel debido a la iteración sobre el mismo al realizar la operación de convolución. Este resulta en una opción con menos ciclos de reloj que las demás aproximaciones de convolución presentes en FAL, con un procesamiento de 5,7 y 14,9 ciclos por píxel respecto a los 32 ciclos por píxel dentro del mismo marco comparativo, resultando en hasta 2,14 veces menos ciclos.

El estudio ejecutado sobre la calidad de los resultados en la imagen arroja que efectivamente un menor tamaño de dato resulta en una imagen con mucha menor calidad porque el valor del píxel tiene menos precisión. Esto es importante considerarlo a la hora de una implementación, dado que el arrastre que puede existir del error durante la convolución capa por capa es un factor directo a la precisión del modelo general. Si se compara el resultado de una convolución del acelerador con

respecto al valor de la imagen teórica se obtienen resultados que indican que es mejor utilizar una mayor precisión y si bien se mostró anteriormente, hacer esto no tiene mayor afectación en el consumo de recursos y en la latencia. Se pueden considerar configurar el hardware a partir de 12 bits según los resultados, ya que se presentan para ambos casos un MSE de inclusive 0.95, un PSNR de 47dB, y un SSIM de 1.0 según el índice, que ya corresponden a valores dentro del rango de aceptación para el estándar de la academia.

Por último, lo que respecta a la red convolucional para la clasificación de imágenes, se logró la correcta integración de ambos aceleradores en la red, con una precisión de hasta 86% en su máxima capacidad operativa. Se realizaron las pruebas correspondientes para encontrar el punto operativo que ofreciera las mejores capacidades y se determinó que los aceleradores operando con un tamaño de dato de 20bits con 6bits de su parte entera para el acelerador de convolución pointwise y un tamaño de dato de 14bits con 7bits de su parte entera para el acelerador de convolución depthwise, se alcanzaba una precisión del modelo de 85%. Esto abre el camino a futuros desarrollos donde se pueda implementar este sistema en problemas de naturaleza crítica (ubicaciones remotas, no disponibilidad de internet o seguridad).

6.1 Recomendaciones

Como parte del proceso de diseño propuesto en el presente proyecto, el hacer una revisión del desarrollo corresponde a una etapa sumamente importante para buscar puntos de mejora y puntos de partida para futuras implementaciones. Las recomendaciones que se deben tomar en cuenta para una efectiva implementación se sintetizan a continuación.

- El proceso de optimización de Vivado HLS puede ser tan minucioso como se desee. Este se redujo a 2 semanas dentro del proyecto probando distintas combinaciones en base a la teoría, pero con la cantidad de directivas existentes en este ambiente, no se descarta que pudiera haber otras combinaciones que resultaran en incluso mejores resultados de latencia y consumo.
- La convolución aquí implementada toma la totalidad de la imagen y utiliza Line Buffers para ejecutarse en un algoritmo básico de convolución, pero no se descarta la utilidad de algoritmos como Winograd en combinación con Line Buffers que resulten en mejores resultados a cambio de un mayor desarrollo técnico.
- El ambiente de pruebas de HLS permite una flexibilidad muy grande para realizar todo tipo de desarrollos, si bien en este se limitó a realizar implementaciones con imágenes básicas como Lenna, con filtros básicos como uno gaussiano, no se descarta que los resultados en cuanto a la calidad de la imagen puedan variar en base a estos dos últimos.
- Tal como se mencionó, las capacidades de los aceleradores pueden resultar superiores a las que resultaron de las pruebas. Para aplicaciones que así lo requieran se puede experimentar con dichas capacidades y obtener aproximaciones superiores a las reportadas.
- El AxC Executer, simula algunas de las operaciones en software, y otras en hardware. Dentro de los próximos proyectos del laboratorio, se encuentra realizar pruebas con aproximaciones 100% en FPGA.

Bibliografía

- [1] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li y L. Fei-Fei, «ImageNet: A large-scale hierarchical image database,» de *2009 IEEE Conference on Computer Vision and Pattern Recognition*, Miami, Florida: IEEE, 2009, p. 248–255.
- [2] A. Krizhevsky, I. Sutskever y G. Hinton, «ImageNet classification with deep convolutional neural networks,» *Communications of the ACM*, vol. 60, pp. 84-90, 24 May 2017.
- [3] B. Hariharan, P. Arbeláez, R. Girshick y J. Malik, «Simultaneous Detection and Segmentation,» de *Computer Vision – ECCV 2014*, Springer International Publishing, 2014, pp. 297-312.
- [4] F. Radenović, G. Toliás y O. Chum, «CNN Image Retrieval Learns from BoW: Unsupervised Fine-Tuning with Hard Examples,» de *Computer Vision – ECCV 2016*, Amsterdam, The: Springer International Publishing, 2016, pp. 3-20.
- [5] J. Sengupta, R. Kubendran, E. Neftci y A. Andreou, «High-Speed, Real-Time, Spike-Based Object Tracking and Path Prediction on Google Edge TPU,» de *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 2020.
- [6] P. Molchanov, S. Tyree, T. Karras, T. Aila y J. Kautz, «Pruning convolutional neural networks for resource efficient inference,» de *5th International Conference on Learning Representations*, Toulon, France, 2017.
- [7] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto y H. Adam, *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*, 2017.
- [8] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov y L.-C. Chen., «Mobilenetv2: Inverted residuals and linearbottlenecks.,» de *2018 IEEE Conference on*

Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 2018, p. 4510–4520.

- [9] N. Ma, X. Zhang, H.-T. Zheng y J. Sun, «ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design,» de *Computer Vision – ECCV 2018*, Munich, Germany: Springer International Publishing, 2018, pp. 122-138.
- [10] G. Huang, S. Liu, L. van der Maaten y K. Q. Weinberger, «CondenseNet: An Efficient DenseNet Using Learned Group Convolutions,» de *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Salt Lake City, UT, USA, IEEE, 2018, p. 2752–2761.
- [11] C. J. D. Alvarez, *GPU vs FPGA: Alcances, ventajas y capacidades*, 2020.
- [12] L. G. León Vega, «Design of a library of generic accelerators of DNN-based inference,» Tecnológico de Costa Rica, Escuela de Ingeniería en electrónica, 2022.
- [13] J. Huang, X. Liu, T. Guo y Z. Zhao, «A High-Performance FPGA-Based Depthwise Separable Convolution Accelerator,» *Electronics*, vol. 12, 2023.
- [14] S. Sun, H. Jiang, M. Yin y C. Zhang, «Design of Efficient CNN Accelerator Based on Zynq Platform,» de *2020 15th International Conference on Computer Science & Education (ICCSE)*, 2020.
- [15] Y. Shen, «Accelerating CNN on FPGA: An Implementation of MobileNet on FPGA,» *KTH ROYAL INSTITUTE OF TECHNOLOGYSCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE*, 2019.
- [16] M. N. Rikiya Yamashita y K. Richard Kinh Gian Do, «Convolutional neural networks: an overview and application in radiology,» *Insights into Imaging*, vol. 9, p. 611–629, 2018.

- [17] Z. M. Chng, «Using Depthwise Separable Convolutions in Tensorflow,» 2022. [En línea]. Available: <https://machinelearningmastery.com/using-depthwise-separable-convolutions-in-tensorflow/>.
- [18] S. Ioffe y C. Szegedy, «Batch normalization: Accelerating deep network training by reducing internal covariate shift,» de *International conference on machine learning*, 2015.
- [19] D. S. Comas, A. Amalfitano, G. J. Meschino y V. L. Ballarin, «Interpretación y visualización de características en texturas mediante Redes Neuronales Convolucionales,» *2022 IEEE Biennial Congress of Argentina (ARGENCON)*, pp. 1-8, 2022.
- [20] A. Zlateski, Z. Jia, K. Li y F. Durand, «A Deeper Look at FFT and Winograd Convolutions,» 2018. [En línea]. Available: <https://api.semanticscholar.org/CorpusID:13723664>.
- [21] N. Malingan, «<https://www.scaler.com/topics/quantization-and-pruning/>,» 2023. [En línea]. Available: <https://www.scaler.com/topics/quantization-and-pruning/>.
- [22] Y. Fisher y K. Vladlen, *Multi-Scale Context Aggregation by Dilated Convolutions*, 2016.
- [23] L. Sifre y S. Mallat, «Rigid-Motion Scattering for Texture Classification,» *ArXiv*, vol. abs/1403.1687, 2014.
- [24] A. G. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le y H. Adam, «Searching for MobileNetV3,» *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 1314-1324, 2019.

- [25] A. J. Rodriguez Figueroa, «Diseño de algoritmos de convolución para acelerar redes neuronales profundas en FPGA mediante el uso de High-Level Synthesis,» Instituto Tecnológico de Costa Rica, Cartago, 2022.
- [26] K. Doğançay, «Chapter 1 - Introduction,» de *Partial-Update Adaptive Signal Processing*, K. Doğançay, Ed., Oxford, Academic Press, 2008, pp. 1-23.
- [27] L. Bai, Y. Zhao y X. Huang, «A CNN Accelerator on FPGA Using Depthwise Separable Convolution,» *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 65, pp. 1415-1419, 2018.
- [28] J. Alfaro-Badilla, «Diseño de un acelerador de hardware para simulaciones de redes neuronales biológicamente precisas utilizando un sistema multi-FPGA,» *tec.ac.cr*, 2017.
- [29] R. Tessier, K. Pocek y A. Dehon, «Reconfigurable Computing Architectures,» *Proceedings of the IEEE*, vol. 103, pp. 332-354, March 2015.
- [30] National Instruments, «LabVIEW FPGA,» 2023. [En línea]. Available: https://www.ni.com/docs/en-US/bundle/labview-fpga-module/page/lvfpgaconcepts/dsp48e_top.html#:~:text=A%20DSP48E%20slice%20is%20a,%2D%20or%20n%2Dstep%20counter..
- [31] Xilinx, «Vivado Design Suite User Guide: High-Level Synthesis,» 2018. [En línea]. Available: <https://docs.xilinx.com/v/u/2018.2-English/ug902-vivado-high-level-synthesis>.
- [32] D. Bailey, «The advantages and limitations of high level synthesis for FPGA based image processing,» 2015.
- [33] M. Gurel, «A Comparative Study between RTL and HLS for Image Processing Applications with FPGAs,» University of California, San Diego, 2016.

- [34] Xilinx, «Vivado Design Suite: AXI Reference Guide (UG1037),» 2017. [En línea]. Available: <https://docs.xilinx.com/v/u/en-US/ug1037-vivado-axi-reference-guide>.
- [35] M. A. D. Guzmán, R. G. Tejero, M. V. Gaudó y D. S. Gracia, «Avances en la síntesis de alto nivel para la generación de hardware en FPGA: Modelos y programabilidad,» *Jornada de Jóvenes Investigadores del I3A*, 2020.
- [36] C. B. de Oliveira, «LALP+: a framework for developing FPGA-based hardware accelerators (LALP+: um framework para o desenvolvimento de aceleradores de hardware em FPGAs),» 2015.
- [37] Q. Xu, T. Mytkowicz y N. Kim, «Approximate Computing: A Survey,» *IEEE Design & Test*, vol. 33, pp. 8-22, February 2016.
- [38] L. Leon-Vega, D. Cordero-Chavarria y J. Castro-Godinez, «Flexible Accelerator Library: Approximate Computing Executer (AxC Executer),» March 2023. [En línea].
- [39] J. Duarte, S. Han, P. Harris, S. Jindariani, E. Kreinar, B. Kreis, J. Ngadiuba, M. Pierini, R. Rivera, N. Tran y Z. Wu, «Fast inference of deep neural networks in FPGAs for particle physics,» *Journal of Instrumentation*, vol. 13, p. P07027, July 2018.
- [40] V. Nair, M. Chatterjee, N. Tavakoli, A. S. Namin y C. Snoeyink, «Fast Fourier Transformation for Optimizing Convolutional Neural Networks in Object Recognition,» arXiv preprint arXiv:2010.04257, 2020.
- [41] B. Van-Hoorick, «A tutorial on non-separable 2D convolutions in Vivado HLS,» 2019.
- [42] E. Elfatimi, R. Eryigit y L. Elfatimi, «Beans Leaf Diseases Classification Using MobileNet Models,» *IEEE Access*, vol. PP, pp. 1-1, 2022.

- [43] P. N. Srinivasu, J. G. SivaSai, M. F. Ijaz, A. K. Bhoi, W. Kim y J. J. Kang, «Classification of Skin Disease Using Deep Learning Neural Networks with MobileNet V2 and LSTM,» *Sensors*, vol. 21, p. 2852, 18 April 2021.
- [44] A. Soud, N. Sakli y H. Sakli, «Classification and Predictions of Lung Diseases from Chest X-rays Using MobileNet V2,» *Applied Sciences*, 2021.
- [45] J. D. Rose, K. Vijayakumar, L. Singh y S. K. Sharma, «Computer-aided diagnosis for breast cancer detection and classification using optimal region growing segmentation with MobileNet model,» *Concurrent Engineering*, vol. 30, pp. 181-189, 2022.
- [46] P. Ramachandran, B. Zoph y Q. V. Le, *Searching for Activation Functions*, 2017.
- [47] B. Zoph y Q. V. Le, *Neural Architecture Search with Reinforcement Learning*, 2017.
- [48] Y. Jewajinda y T. Thongkum, «A Resource-Limited FPGA-based MobileNetV3 Accelerator,» de *2022 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, 2022.
- [49] K. Ulrich y S. Eppinger, *Diseño y desarrollo de productos*, Ciudad de México: McGraw Hill Education, 2009.
- [50] F. Chollet, *Xception: Deep Learning with Depthwise Separable Convolutions*, 2017.
- [51] M. Tan y Q. V. Le, *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*, 2020.
- [52] Xilinx, «Vivado Design Suite User Guide: High-Level Synthesis,» 2018. [En línea]. Available: <https://docs.xilinx.com/v/u/2018.2-English/ug902-vivado-high-level-synthesis..>

- [53] «Unit Description - NVDLA Documentation,» *Nvdla.org*, 2018.
- [54] L. Ferretti, A. Cini, G. Zacharopoulos, C. Alippi y L. Pozzi, «Graph Neural Networks for High-Level Synthesis Design Space Exploration,» *ACM Trans. Des. Autom. Electron. Syst.*, vol. 28, December 2022.
- [55] K. Okarma, «Comparison of modern nonlinear multichannel filtering techniques using recent fullreference image quality assessment methods,» 2017.
- [56] National-Instruments, *Peak Signal-to-Noise Ratio as an Image Quality Metric*, 2023.
- [57] G. Prieto Renieblas, «Uso de SSIM como índice de calidad de imagen médica,» 2009.
- [58] J. A. Chaves Barrantes, IMPLEMENTACIÓN DE UN PROTOTIPO PARA LAMEDICIÓN DEL PULSO CARDÍACO MEDIANTE PROCESAMIENTO DE VÍDEO EN LA TARJETA DEDESARROLLO JETSON TX1, 2018.
- [59] Xilinx, «SDSoC Development Environment Help for 2019.1: Understanding the Hardware Function Optimization Methodology,» 2019. [En línea]. Available:
https://www.xilinx.com/htmldocs/xilinx2019_1/sdsoc_doc/optimizing-hardware-function-uww1504034429970.html.
- [60] M. Nadipally, «Chapter 2 - Optimization of Methods for Image-Texture Segmentation Using Ant Colony Optimization,» de *Intelligent Data Analysis for Biomedical Applications*, D. J. Hemanth, D. Gupta y V. E. Balas, Edits., Academic Press, 2019, pp. 21-47.
- [61] C. Baskin, N. Liss, A. Mendelson y E. Zheltonozhskii, «Streaming Architecture for Large-Scale Quantized Neural Networks on an FPGA-Based Dataflow Platform,» July 2017.

- [62] *Research: AI and compute*, 2018.
- [63] E. Strubell, A. Ganesh y A. Mccallum, «Energy and Policy Considerations for Deep Learning in NLP,» 2019.
- [64] K. Martineau, *Shrinking deep learning's carbon footprint*, 2020.
- [65] K. Ovtcharov, O. Ruwase, J.-Y. Kim, J. Fowers, K. Strauss y E. Chung, Accelerating Deep Convolutional Neural Networks Using Specialized Hardware, 2015.
- [66] R. Toews, «Deep Learning's Carbon Emissions Problem,» *Forbes*, October 2023.
- [67] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto y H. Adam, «Accelerating Convolutional Neural Networks for Mobile Applications,» de *Proceedings of the 24th ACM international conference on Multimedia*, 2017.
- [68] J. A. Chaves-Barrantes, «IMPLEMENTACIÓN DE UN PROTOTIPO PARA LA MEDICIÓN DEL PULSO CARDÍACO MEDIANTE PROCESAMIENTO DE VÍDEO EN LA TARJETA DE DESARROLLO JETSON TX1,» 2018.
- [69] L. Biewald, *Deep Learning and Carbon Emissions - Towards Data Science*, Towards Data Science, 2019.

Anexos

Anexo 1. Algoritmos

Algoritmo A.1 Data Load

```
1: input: image stream payload
2: output: pixel data
3:
4: do:
5:     payload = data packets in input
6:     for each data packet:
7:         low = Size of Data
8:         high = low + Size of Data - 1
9:         pixel data = data packet data from low to high
10:    return pixel data
11: while not the last packet
12: End Function
```

Algoritmo A.2 Data Write

```
1: input: convolved pixel data
2: output: image stream payload
3:
4: elements_left = pixels_at_output
5: while elements_left is not empty:
6:     for each data packet:
7:         low = Size of Data
8:         high = low + Size of Data - 1
9:         pixel data = convolved pixel data at input
10:        insert pixel data between low and high in image stream payload
11:    return image stream payload
12: while not the last packet
13: End Function
```

Anexo 2. Figuras

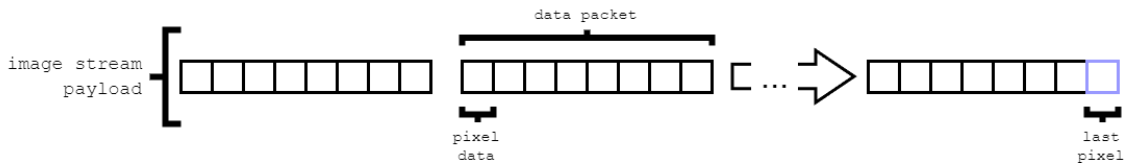


Figura A 1. Manejo de paquetes

Anexo 3. Repositorios y códigos de consulta.

Depthwise Convolution Accelerator	https://gitlab.com/ecas-lab-tec/approximate-flexible-acceleration-ml/flexible-accelerators-library-fal/-/tree/feature/execute-dse-in-dws/examples/depthwise-convolution?ref_type=heads
Pointwise Convolution Accelerator	https://gitlab.com/ecas-lab-tec/approximate-flexible-acceleration-ml/flexible-accelerators-library-fal/-/tree/feature/execute-dse-in-dws/examples/pointwise-convolution?ref_type=heads
AxC Executer: MobileNetV2	https://gitlab.com/ecas-lab-tec/approximate-flexible-acceleration-ml/axc-executer/-/tree/feature/add-fal-custom-depthconv2d-accelerator/examples/axc-mobilenetv2?ref_type=heads