



## UNIVERSIDAD DEL BÍO-BÍO

FACULTAD DE INGENIERÍA

DEPARTAMENTO DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

# Aceleración Hardware de un Algoritmo de Aprendizaje Activo para Clasificadores de Imágenes

**Por:** Angelo Roberto Barbieri Figueroa

**Profesor Guía:** Christopher Alejandro Flores Jara

**Profesor Co-Guía:** Wladimir Elias Valenzuela Fuentealba

CONCEPCIÓN - CHILE

ABRIL, 2025



## Resumen

La clasificación de imágenes es una técnica aplicada en múltiples áreas, fundamentada en algoritmos de aprendizaje supervisado de inteligencia artificial. El rendimiento de estos algoritmos depende de la calidad y cantidad de imágenes utilizadas durante el entrenamiento, lo que conlleva un elevado gasto en términos de tiempo, esfuerzos humanos y recursos económicos para el etiquetado, ya sea realizado por expertos o mediante procesos secuenciales. En este sentido, el aprendizaje activo (*Active Learning*, AL) permite seleccionar únicamente aquellas imágenes más informativas para la anotación, reduciendo la cantidad de ejemplos de entrenamiento que se necesitan para alcanzar un rendimiento adecuado según métricas de medición de desempeño en tareas de clasificación.

El aumento en la capacidad computacional ha permitido la implementación de algoritmos de inteligencia artificial. Sin embargo, la elevada demanda de recursos computacionales y energéticos ha motivado el desarrollo de aceleradores *hardware*. Los arreglos de compuertas programables (*Field Programmable Gate Arrays*, FPGAs) se consideran una opción viable para estas implementaciones debido a su capacidad de cómputo de grano fino, que permite un alto nivel de paralelismo en el procesamiento, sumado a su capacidad de re-configuración y adaptación a las necesidades específicas de cada aplicación.

Este trabajo describe el diseño, implementación y validación de un circuito digital que selecciona los índices más inciertos en un modelo de clasificación de imágenes, utilizando el margen entre las dos probabilidades más altas de cada predicción. La implementación se realiza en un sistema en chip (SoC) ZYNQ-7000 XC7Z020-1CLG400C de Xilinx, perteneciente a la familia Artix-7, que incorpora un procesador ARM Cortex-A9 de doble núcleo embebido. Se diseñó un algoritmo propio para la selección de índices que resuelve el desafío de obtener el máximo valor de una memoria en cada ciclo de reloj. Para ello, se divide la memoria en múltiples unidades, de modo que el máximo de cada una se calcula de forma anticipada, permitiendo el procesamiento en línea sin afectar el desempeño en AL. El circuito opera a 110 MHz, consume menos de 2 W y acelera la velocidad de cómputo en más de 100 veces en comparación con la implementación en un sistema de propósito general, utilizando el 47.4 % de los recursos disponibles (23,612 slices LUT, 46,629 slice registers y 65 unidades BRAM).

# Índice general

|   |           |
|---|-----------|
| <b>Resumen</b>  | <b>3</b>  |
| <b>1 Introducción</b>                                   | <b>1</b>  |
| 1.1 Objetivos . . . . .                                 | 2         |
| 1.1.1 Objetivos general . . . . .                       | 2         |
| 1.1.2 Objetivos específicos . . . . .                   | 2         |
| 1.2 Alcances y limitaciones . . . . .                   | 3         |
| 1.3 Contribuciones del autor . . . . .                  | 3         |
| <b>2 Estado del Arte y Motivaciones</b>                 | <b>4</b>  |
| 2.1 Clasificación de imágenes . . . . .                 | 4         |
| 2.1.1 Visión por computador en clasificación . . . . .  | 5         |
| 2.1.2 ¿Qué es una imagen? . . . . .                     | 6         |
| 2.1.3 Pre-procesamiento . . . . .                       | 7         |
| 2.1.4 Extracción de características . . . . .           | 8         |
| 2.1.5 Clasificación . . . . .                           | 9         |
| 2.2 Redes neuronales artificiales . . . . .             | 10        |
| 2.2.1 Redes neuronales convolucionales . . . . .        | 10        |
| 2.2.1.1 Arquitecturas Clásicas . . . . .                | 17        |
| 2.2.1.2 Arquitecturas más compactas . . . . .           | 24        |
| 2.2.2 Transformadores visuales . . . . .                | 26        |
| 2.3 Aprendizaje activo . . . . .                        | 29        |
| 2.3.1 ¿Qué es el aprendizaje activo? . . . . .          | 29        |
| 2.3.2 Escenarios de AL . . . . .                        | 30        |
| 2.3.3 Estrategias de consulta . . . . .                 | 31        |
| 2.4 Acelerador Hardware . . . . .                       | 32        |
| 2.4.1 ¿Qué es un FPGA? . . . . .                        | 33        |
| 2.5 Discusión . . . . .                                 | 36        |
| <b>3 Materiales y Métodos</b>                           | <b>38</b> |
| 3.1 Plataforma de aceleración . . . . .                 | 38        |
| 3.2 Conjuntos de datos y preprocesamiento . . . . .     | 40        |
| 3.3 Definición del problema y metodología . . . . .     | 43        |
| <b>4 Resultados: Clasificación y Aprendizaje Activo</b> | <b>45</b> |
| 4.1 Desempeño de clasificadores de imágenes . . . . .   | 45        |

|          |   |           |
|----------|---|-----------|
| 4.2      | Aprendizaje activo . . . . .                  | 48        |
| 4.2.1    | Aprendizaje activo en Fashion-MNIST . . . . . | 50        |
| 4.2.2    | Aprendizaje activo en CIFAR-10 . . . . .      | 54        |
| 4.2.3    | Selección de estrategia de consulta . . . . . | 58        |
| <b>5</b> | <b>Arquitectura Digital en SoC</b>            | <b>60</b> |
| 5.1      | Arquitectura general . . . . .                | 60        |
| 5.1.1    | Descripción de Margin Sampling . . . . .      | 62        |
| 5.2      | Diseño de la arquitectura digital . . . . .   | 63        |
| 5.2.1    | Emulación en <i>software</i> . . . . .        | 68        |
| 5.3      | Arquitectura digital . . . . .                | 74        |
| <b>6</b> | <b>Resultados: Implementación en SoC</b>      | <b>79</b> |
| 6.1      | Velocidad de computo . . . . .                | 80        |
| 6.2      | Utilización de recursos . . . . .             | 83        |
| 6.3      | Consumo de potencia . . . . .                 | 86        |
| <b>7</b> | <b>Conclusiones y trabajo futuro</b>          | <b>88</b> |
|          | <b>Referencias</b>                            | <b>90</b> |

# Índice de Tablas

|      |  |    |
|------|--|----|
| 4.1  | Cantidad de parámetros totales y parámetros entrenables. Fuente: Elaboración Propia. . . . .   | 46 |
| 4.2  | Hiperparámetros utilizados en el entrenamiento de las CNNs. Fuente: Elaboración propia. . . . .  | 46 |
| 4.3  | Desempeño de CNNs en los distintos conjuntos de imágenes. Fuente: Elaboración propia. . . . .  | 47 |
| 4.4  | AULC en Fashion-MNIST. Fuente: Elaboración propia. . . . .   | 51 |
| 4.5  | Cantidad mínima de ejemplos requeridos por los diferentes algoritmos de aprendizaje activo para alcanzar las métricas ACC y F1 en el conjunto de imágenes Fashion-MNIST utilizando MobileNetV1. Fuente: Elaboración propia. . . . .    | 52 |
| 4.6  | Cantidad mínima de ejemplos requeridos por los diferentes algoritmos de aprendizaje activo para alcanzar las métricas ACC y F1 en el conjunto de imágenes Fashion-MNIST utilizando EfficientNetB0. Fuente: Elaboración propia. . . . . | 53 |
| 4.7  | Cantidad mínima de ejemplos requeridos por los diferentes algoritmos de aprendizaje activo para alcanzar las métricas ACC y F1 en el conjunto de imágenes Fashion-MNIST utilizando ResNet50. Fuente: Elaboración propia. . . . .       | 54 |
| 4.8  | AULC en CIFAR-10. Fuente: Elaboración propia. . . . .  | 55 |
| 4.9  | Cantidad mínima de ejemplos requeridos por los diferentes algoritmos de aprendizaje activo para alcanzar las métricas ACC y F1 en el conjunto de imágenes CIFAR-10 usando la CNN MobileNetV1. . . . .                                  | 56 |
| 4.10 | Cantidad mínima de ejemplos requeridos por los diferentes algoritmos de aprendizaje activo para alcanzar las métricas ACC y F1 en el conjunto de imágenes CIFAR-10 usando la CNN EfficientNetB0. . . . .                               | 57 |
| 4.11 | Cantidad mínima de ejemplos requeridos por los diferentes algoritmos de aprendizaje activo para alcanzar las métricas ACC y F1 en el conjunto de imágenes CIFAR-10 usando la CNN ResNet50.   | 58 |
| 5.1  | Mínimo, máximo y promedio de índices no coincidentes para distinta cantidad de datos de entrada y distintas configuraciones de tamaño y cantidad de unidades de memoria. Fuente: Elaboración propia. .                                 | 69 |
| 5.2  | Estadísticas de desajuste por configuración de memoria . . . . .   | 69 |

|     |  |    |
|-----|--|----|
| 5.3 | AULC de la implementación base y la emulación. Fuente:<br>Elaboración propia. . . . .                        | 74 |
| 6.1 | Utilización de recursos en las jerarquías principales. Fuente:<br>Elaboración propia. . . . .                | 84 |
| 6.2 | Utilización de recursos de los módulos dentro de MSA. Fuente:<br>Elaboración propia. . . . .                 | 84 |
| 6.3 | Consumo de potencia dinámica de la implementación en SoC por<br>recurso. Fuente: Elaboración propia. . . . . | 86 |
| 6.4 | Consumo de potencia de implementación en SoC por módulo<br>jerárquico. Fuente: Elaboración propia. . . . .   | 87 |

# Índice de Figuras

|      |   |    |
|------|---|----|
| 2.1  | <i>Pipeline</i> de visión por computador. Fuente: Adaptado de [4]. . . . .  | 5  |
| 2.2  | Comparación de imagen en escala de grises y en color. Fuente: Adaptado de [4]. . . . .  | 7  |
| 2.3  | Preprocesamiento para quitar ruido de una imagen. El ruido del tipo sal y pimienta en (a) es removido aplicando un filtro de la mediana, dando como resultado (b). Fuente: Elaboración propia. . . . .  | 8  |
| 2.4  | Red neuronal convolucional. Fuente: Adaptado de [4]. . . . .  | 11 |
| 2.5  | Neurona artificial. Fuente: Adaptado de [4]. . . . .  | 12 |
| 2.6  | Red perceptrón multicapa. Fuente: Adaptado de [4]. . . . .  | 13 |
| 2.7  | Funciones de activación. Fuente: Adaptado de [10]. . . . .  | 15 |
| 2.8  | Dropout. Fuente: Adaptación de [4]. . . . .   | 16 |
| 2.9  | Arquitectura de LeNet. Fuente: Adaptado de [4, 5]. . . . .  | 17 |
| 2.10 | Arquitectura AlexNet. Fuente: Adaptado de [4, 6]. . . . .   | 19 |
| 2.11 | Arquitectura VGGNet16. Fuente: Adaptado de [4, 7]. . . . .  | 20 |
| 2.12 | Módulo Inception. Fuente: Adaptado de [4, 8]. . . . .   | 21 |
| 2.13 | Arquitectura GoogLeNet. Fuente: Adaptado de [4, 8]. . . . .   | 22 |
| 2.14 | Módulo ResNet (conexión residual). Fuente: Adaptado de [4, 9]. . . . .  | 23 |
| 2.15 | Arquitectura ResNet-50, Fuente: Adaptado de [4, 9]. . . . .   | 24 |
| 2.16 | Convolución <i>depthwise-separable</i> . Fuente: Adapatado de [44]. . . . .   | 25 |
| 2.17 | Escalamiento en redes EfficientNet. Fuente: Adaptado de [46]. . . . .   | 26 |
| 2.18 | Arquitectura de transformadores visuales. Fuente: Adaptado de [34]. . . . .   | 27 |
| 2.19 | Pasos de un algoritmo de AL con escenario <i>pool-based</i> . Fuente: Adaptado de [11, 12]. . . . .   | 30 |
| 2.20 | Arquitectura de un FPGA. Fuente: Adaptado de [15, 56] . . . . .   | 34 |
| 2.21 | Flujo de diseño en FPGA. Fuente: Adaptado de [15]. . . . .  | 35 |
| 3.1  | Tarjeta de desarrollo PYNQ-Z2 de Xilinx. Fuente: Adaptado de <a href="https://www.tulembedded.com/FPGA/ProductsPYNQ-Z2.html">https://www.tulembedded.com/FPGA/ProductsPYNQ-Z2.html</a> . Fecha de último acceso: 20 de Febrero de 2025 a las 18:36 hrs. . . . . | 39 |
| 3.2  | Ejemplos de imágenes en Fashion-MNIST. Fuente: Adaptado de <a href="https://github.com/zalandoresearch/fashion-mnist">https://github.com/zalandoresearch/fashion-mnist</a> . Fecha de último acceso: 16 de Febrero de 2025 a las 09:21 hrs. . . . .             | 41 |
| 3.3  | Ejemplos de imágenes en CIFAR-10. Fuente: Adaptado de <a href="https://www.cs.toronto.edu/~kriz/cifar.html">https://www.cs.toronto.edu/~kriz/cifar.html</a> . Fecha de último acceso: 16 de Febrero de 2025 a las 10:03 hrs. . . . .                            | 42 |
| 3.4  | Diagrama de flujo de la metodología propuesta. Fuente: Elaboración propia. . . . .  | 43 |

---

|      |   |    |
|------|---|----|
| 4.1  | Curvas de aprendizaje de los algoritmos de aprendizaje activo en el conjunto de datos Fashion-MNIST utilizando MobileNetV1, EfficientNetB0 y ResNet50. Fuente: Elaboración propia. . . . .                          | 51 |
| 4.2  | Curvas de aprendizaje de los algoritmos de aprendizaje activo en el conjunto de datos CIFAR-10 utilizando MobileNetV1, EfficientNetB0 y ResNet50. Fuente: Elaboración propia. . . . .                               | 55 |
| 5.1  | Arquitectura general del sistema. Fuente: Adaptado de <a href="https://hackmd.io/@ween168/rkZnpSxfl">https://hackmd.io/@ween168/rkZnpSxfl</a> . Fecha de último acceso: 7 de Marzo de 2025 a las 08:07 hrs. . . . . | 61 |
| 5.2  | Serie de pasos de <i>Margin Sampling</i> . Fuente: Elaboración propia. .  | 62 |
| 5.3  | Árbol de comparaciones binario para encontrar las dos mayores probabilidades. El punto que acompaña a los comparadores indica la salida del mayor. Fuente: Elaboración propia. . . . .                              | 63 |
| 5.4  | Árbol de comparaciones binario para encontrar el valor máximo. Como $N = 8$ , el número de ciclos que tarda en encontrar el máximo es $\log_2 8 = 3$ . Fuente: Elaboración propia. . . . .                          | 67 |
| 5.5  | Índices asignados por la implementación base para los índices no coincidentes, con una cantidad de datos de entrada igual a 2560. Fuente: Elaboración propia. . . . .   | 70 |
| 5.6  | Índices asignados por la implementación base para los índices no coincidentes, con una cantidad de datos de entrada igual a 5120. Fuente: Elaboración propia. . . . .   | 71 |
| 5.7  | Índices asignados por la implementación base para los índices no coincidentes, con una cantidad de datos de entrada igual a 10240. Fuente: Elaboración propia. . . . .  | 72 |
| 5.8  | Curvas de aprendizaje de ambas implementaciones, base ( <i>software</i> ) y emulación. Fuente: Elaboración propia. . . . .  | 73 |
| 5.9  | Arquitectura digital del acelerador <i>hardware</i> para <i>Margin Sampling</i> . Fuente: Elaboración propia. . . . .   | 75 |
| 5.10 | Arquitectura digital del acelerador <i>hardware</i> para <i>Margin Sampling</i> , con modificaciones para solucionar problemas de <i>timing</i> . Fuente: Elaboración propia. . . . .                               | 77 |
| 6.1  | Caso crítico en estudio (corresponde al caso con linea roja punteada). Fuente: Elaboración propia. . . . .  | 80 |
| 6.2  | Curva de aceleración. Fuente: Elaboración propia. . . . .   | 82 |
| 6.3  | Diseño implementado. Fuente: Elaboración propia. . . . .  | 85 |

# Capítulo 1

## Introducción

El desarrollo tecnológico ha impulsado la capacidad computacional así como la disponibilidad de grandes volúmenes de datos [1, 2, 3]. En el área de visión por computador las imágenes son un recurso valioso, a partir de la cuales es posible extraer información útil para análisis y toma de decisiones. En ese sentido, la clasificación de imágenes ha emergido como una de las técnicas más utilizadas en el procesamiento de dichas fuentes.

La clasificación de imágenes permite asignar etiquetas predefinidas a las imágenes según su contenido [4]. Entre sus aplicaciones se encuentran la medicina, donde se utiliza para el diagnóstico de enfermedades a partir de imágenes médicas; los automóviles autónomos, donde ayuda a identificar señales de tráfico y peatones; la agricultura, al detectar plagas y enfermedades en cultivos. En los últimos años, con el desarrollo de la inteligencia artificial (*Artificial Intelligence*, AI), particularmente, las redes neuronales profundas (*Deep Neural Networks*, DNNs) se han posicionado como referentes no solo en tareas de clasificación, sino también en otras tareas de visión por computador como la detección de objetos, generación de imágenes, entre otras [4, 5, 6, 7, 8, 9, 10].

Independiente del método de clasificación, es crucial contar con un gran número de imágenes etiquetadas. Aunque obtener imágenes puede no ser una tarea difícil en ciertas áreas, el proceso de etiquetado suele ser costoso en términos de tiempo y recursos. Esto se debe a la necesidad de contar con anotadores especializados, como médicos que identifican tumores cerebrales en imágenes de resonancia magnética, y/o a una extensa serie de pasos experimentales que a menudo deben llevarse a

cabo antes de realizar el etiquetado, como en pruebas de laboratorio [11]. Para reducir el esfuerzo y los costos asociados al etiquetado de datos, el aprendizaje activo (*Active Learning*, AL) sugiere que el modelo identifique y seleccione las imágenes más informativas de un conjunto de datos no etiquetados [11, 12].

Los algoritmos de aprendizaje activo involucran un proceso iterativo y la manipulación de grandes volúmenes de datos, esto los hace computacionalmente costosos, más aún si se trata de imágenes y clasificadores basados en redes neuronales profundas [11, 12, 13, 14, 15]. Si bien este tipo de algoritmos puede ser procesado por *Unidades de Procesamiento Gráfico* (*Graphic Processing Units*, GPUs), los FPGAs son menos costosos en términos de espacio, tienen un menor consumo de potencia, en general el precio de una placa de desarrollo equivale a una fracción del precio de una GPU y ofrecen la ventaja de re-configure, permitiendo adaptar el *hardware* a las necesidades específicas de la aplicación [14, 15]. Dado el creciente interés en AL, se han reportado algunos trabajos en los últimos años relacionados a implementaciones en FPGA, que principalmente se enfocan en mejorar las arquitecturas de las redes neuronales [16, 17, 18, 19].

## 1.1. Objetivos

### 1.1.1. Objetivos general

Desarrollar e implementar un acelerador *hardware* para un algoritmo de aprendizaje activo utilizado en la clasificación de imágenes.

### 1.1.2. Objetivos específicos

1. Recopilar literatura relacionada con la clasificación de imágenes, aprendizaje activo y aceleración *hardware*.
2. Construir conjuntos de datos con imágenes etiquetadas para problemas de clasificación.
3. Implementar en *software* un algoritmo de aprendizaje activo para clasificadores de imágenes basados en redes neuronales.
4. Diseñar, implementar y validar una arquitectura digital para la aceleración *hardware* de un algoritmo de aprendizaje activo aplicado a clasificadores de

imágenes basados en redes neuronales.

5. Evaluar el rendimiento del aprendizaje activo entre la implementación basada en *software* y por *hardware*, considerando métricas como la velocidad de procesamiento, la eficiencia energética y la exactitud predictiva de los clasificadores implementados.

## 1.2. Alcances y limitaciones

1. Los conjuntos de imágenes utilizados son obtenidos desde fuentes públicas como *Kaggle*<sup>1</sup>, *TensorFlow Datasets*<sup>2</sup> entre otros.
2. La manipulación de los conjuntos de imágenes (análisis, exploratorio, preprocessamiento, entre otros) es realizada utilizando el lenguaje de programación *Python* versión 3.10.12, desde la plataforma *Google Colaboratory*.
3. El desarrollo de los clasificadores y los algoritmos de aprendizaje activo se lleva a cabo principalmente utilizando *TensorFlow* como marco de trabajo, junto con bibliotecas como *Keras* y *Scikit-Learn*.
4. La síntesis e implementación se realiza en el IDE Vivado versión 2024.1. La plataforma utilizada es PYNQ-Z2 que incluye un sistema en chip (*Sistem on Chip*, SoC) ZYNQ-7000.

## 1.3. Contribuciones del autor

1. Análisis comparativo de estrategias de consultas de AL para la clasificación de imágenes.
2. Algoritmo de selección en línea de los  $k$  menores/mayores valores de un conjunto de datos.
3. Diseño, implementación y validación de una arquitectura digital para la aceleración por *hardware* de una estrategia de consulta de AL.

---

<sup>1</sup><https://www.kaggle.com/>. Fecha de último acceso: 9 de Marzo de 2025 a las 13:44 hrs.

<sup>2</sup><https://www.tensorflow.org/datasets/catalog/overview>. Fecha de último acceso: 9 de Marzo de 2025 a las 13:45 hrs.

## Capítulo 2

# Estado del Arte y Motivaciones

### 2.1. Clasificación de imágenes

Es esencial que un sistema de inteligencia artificial pueda comprender su entorno y, de esta forma, tomar decisiones informadas. La visión por computadora (*Computer Vision, CV*) se define como un área de la inteligencia artificial que se ocupa de la percepción visual, extrayendo e interpretando información relevante desde imágenes digitales, videos u otras entradas visuales. Basado en la visión humana, un sistema de visión consiste en dos componentes principales: un dispositivo de adquisición de datos que captura información (función del ojo humano) y un dispositivo que es capaz de interpretar la información detectada (función del cerebro) [4].

#### Dispositivos de adquisición de datos

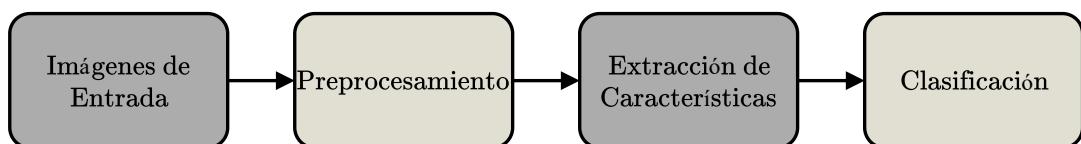
Entre las capacidades sensoriales humanas, la visión es el sentido más avanzado. Sin embargo, mientras que la visión humana está limitada a una estrecha banda de luz visible del espectro electromagnético, los dispositivos de adquisición de imágenes abarcan todo el espectro electromagnético, desde los rayos gamma hasta las ondas de radio. Además, las tecnologías actuales permiten generar representaciones visuales a partir de fuentes que normalmente no se asocian con imágenes. Dado que los sistemas de visión están diseñados para realizar tareas específicas, es crucial seleccionar el dispositivo de detección que mejor se ajuste a las características del problema [4, 20].

## Dispositivo interpretador

Como dispositivo interpretador, se utilizan algoritmos de visión por computador, que actúan como el cerebro del sistema de visión. Inspirados en el aprendizaje de las neuronas biológicas, donde una señal de salida se envía a otras neuronas conectadas si se activan suficientes señales de entrada, los científicos desarrollaron un cerebro artificial con neuronas artificiales. De este modo nacen las redes neuronales artificiales (*Artificial Neural Networks*, ANNs). Aunque cada neurona realiza una función simple por sí sola, la agrupación de neuronas en capas y la conexión de múltiples capas entre sí forman una red capaz de aprender. El uso de redes con múltiples capas ocultas de neuronas se denomina aprendizaje profundo (*Deep Learning*, DL) [4].

### 2.1.1. Visión por computador en clasificación

La clasificación es la tarea de asignar una etiqueta a una imagen desde un conjunto de categorías predefinidas [4]. La Figura 2.1 muestra como el trabajo que realiza el dispositivo interpretador se divide en una serie de pasos.



**Figura 2.1:** *Pipeline* de visión por computador. Fuente: Adaptado de [4].

La extracción de características se puede realizar mediante métodos manuales o automáticos. En un enfoque tradicional, se utilizan métodos de extracción de características manuales y algoritmos de clasificación convencionales, que no participan directamente en la creación del espacio de características. En contraste, el enfoque basado en redes neuronales emplea técnicas de aprendizaje profundo que integran tanto la extracción de características como la clasificación en un solo proceso [4, 10, 20].

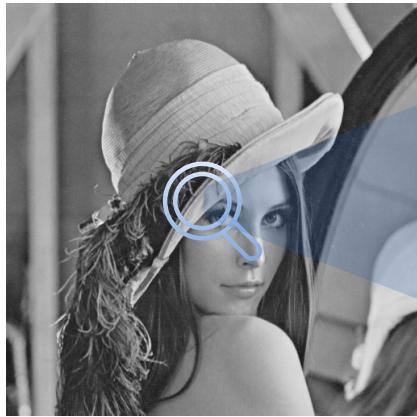
### 2.1.2. ¿Qué es una imagen?

Una imagen puede ser definida como una función bidimensional  $f(x, y)$ , donde  $x$  e  $y$  son las coordenadas espaciales en un plano, y la amplitud de  $f$  para cualquier par de coordenadas  $(x, y)$  se denomina intensidad o nivel de gris en ese punto. Es importante notar que una imagen se compone de un número finito de elementos llamados píxeles, cada uno con una posición en el plano  $xy$  y un valor específico. Por ejemplo, en muchas imágenes en escala de grises se codifica la intensidad de cada píxel en un rango de 0 (negro) a 255 (blanco) utilizando 8 bits, aunque existen otras codificaciones posibles [4, 20].

En el caso de imágenes a color, cada píxel tiene un valor de intensidad para cada canal de color. Por ejemplo, en el sistema RGB (*Red, Green, Blue*), cada píxel se representa mediante su intensidad en el canal rojo, intensidad en el canal verde e intensidad en el canal azul. Esta representación se extiende a otros sistemas de color como HSV (*Hue, Saturation, Value*) [4].

La Figura 2.2 muestra que las imágenes a color son tratadas como matrices 3D. Dada la composición de una imagen digital, en un computador estas se tratan como matrices de píxeles. En una imagen en escala de grises, cada píxel determina el valor de intensidad de un solo color, lo que se puede representar mediante una matriz 2D. En contraste, en las imágenes a color, como en el sistema RGB, se utilizan tres matrices, una para cada canal: una matriz para la intensidad del color rojo, otra para la intensidad del color verde y una tercera para la intensidad del color azul [4].

$f(250 : 255, 250 : 255)$



|     |     |     |     |     |     |
|-----|-----|-----|-----|-----|-----|
| 187 | 189 | 192 | 197 | 195 | ... |
| 190 | 196 | 197 | 199 | 193 | ... |
| 193 | 197 | 199 | 192 | 158 | ... |
| 199 | 199 | 189 | 149 | 108 | ... |
| 201 | 183 | 130 | 100 | 98  | ... |
| ... | ... | ... | ... | ... | ... |

(a) Imagen en escala de grises

$f(250 : 255, 250 : 255)$



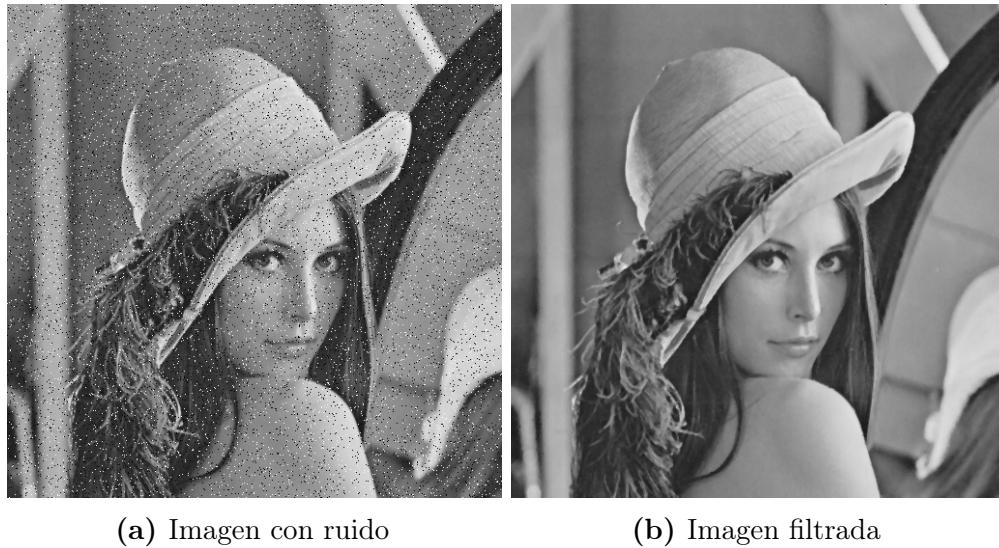
| Canales RGB | 214 | 215 | 215 | 223 | 219 | ... |
|-------------|-----|-----|-----|-----|-----|-----|
| 177         | 180 | 183 | 187 | 188 | 185 | ... |
| 191         | 188 | 180 | 194 | 191 | 195 | ... |
| 170         | 169 | 181 | 180 | 168 | 175 | ... |
| 175         | 180 | 176 | 171 | 173 | 176 | ... |
| 176         | 180 | 174 | 163 | 109 | 169 | ... |
| 169         | 173 | 161 | 93  | 79  | 176 | ... |
| 176         | 147 | 81  | 74  | 67  | 147 | ... |
| ...         | ... | ... | ... | ... | ... | ... |

(b) Imagen en color

**Figura 2.2:** Comparación de imagen en escala de grises y en color. Fuente: Adaptado de [4].

### 2.1.3. Pre-procesamiento

La Figura 2.3 muestra un pre-procesamiento típico. En esta etapa, se realizan operaciones sobre las imágenes como re-dimensionamiento, normalización, aumento de datos, eliminación de ruido, corrección de iluminación, segmentación, transformaciones geométricas y filtrado de bordes. El objetivo es mejorar la calidad de la información, de modo que se facilite el análisis y procesamiento computacional en los pasos posteriores [4, 20].



(a) Imagen con ruido

(b) Imagen filtrada

**Figura 2.3:** Preprocesamiento para quitar ruido de una imagen. El ruido del tipo sal y pimienta en (a) es removido aplicando un filtro de la mediana, dando como resultado (b). Fuente: Elaboración propia.

#### 2.1.4. Extracción de características

El principal objetivo de la extracción características es obtener la información más relevante de los datos originales y representar esta información en un espacio dimensional reducido, donde el conjunto de características extraídas se denomina vector de características [21]. En el enfoque tradicional existen distintos métodos. La forma más primitiva consiste en considerar los píxeles individuales como características. Patrón Binario Local (*Local Binary Pattern*, LBP) es una técnica que compara el valor de intensidad de un píxel central (umbral) con todos los píxeles que lo rodean (vecinos) y asigna un número binario que representa la textura de ese vecindario. Luego, se construye un histograma para analizar la distribución de texturas en la imagen. Esta técnica, utilizada en la clasificación de texturas y reconocimiento facial, entre otras aplicaciones, fue introducida en 1994 por Timo Ojala, Matti Pietikäinen y David Harwood [22]. En 1999, David Lowe presentó un método llamado Características de Escala Invariante (*Scale-Invariant Feature Transform*, SIFT) que transforma una imagen en múltiples vectores de características invariantes a transformaciones geométricas de la imagen, tales como traslación, cambios de escala y rotación. Además, estos vectores son parcialmente invariantes a los cambios de iluminación y adecuados para imágenes

de 3 canales [23]. Una técnica similar a *SIFT*, pero que utiliza aproximaciones para acelerar el cálculo, reducir el tiempo de procesamiento y es más robusta, llamada Características Robusta Aceleradas (*Speeded-Up Robust Features*, SURF), fue presentada por Herbet Bay, Tinne Tuytelaars y Luc Van Gool en 2006 [24]. En 2001, con la intención de reconocer rostros en tiempo real, Paul Viola y Michael Jones presentaron el método de Características de Haar. Este método utiliza características basadas en la suma de píxeles dentro de áreas rectangulares. Para calcular estas sumas de manera eficiente, introdujeron el concepto de imagen integral, una representación de la imagen que permite obtener rápidamente la suma de píxeles en cualquier rectángulo [25]. La técnica de Bolsa de Palabras Visuales (*Bag of Visual Words*, BoVW) fue introducida en 2004 por Gabriella Csurka y sus colaboradores [26]. Esta técnica extrae vectores de características de zonas específicas de una imagen y luego agrupa estos vectores utilizando técnicas de *clustering* para crear las "palabras visuales". Finalmente, cada imagen puede ser representada por un histograma de "palabras visuales". En 2005, Navneet Dalal y Bill Triggs utilizaron un método de Histogramas de Direcciones de Gradientes (*Histogram of Gradients*, HOG) para la detección de peatones. Este método construye histogramas de direcciones de los gradientes dentro de celdas (subdivisiones de la imagen). Luego, forma bloques a partir de la unión de múltiples celdas; estos bloques son normalizados y concatenados para formar el vector de características, mejorando así la invariancia a la iluminación y el contraste [27].

### 2.1.5. Clasificación

Del mismo modo, existen varios algoritmos de clasificación, entre los cuales se destacan algunos por su relevancia histórica. Existe una colección de clasificadores basados en el teorema de Bayes, conocidos como clasificadores Naive Bayes. Los primeros desarrollos de estos clasificadores se originan en trabajos estadísticos a finales de la década de 1960. El principio detrás de estos clasificadores es que asumen independencia condicional entre las características. Esto significa que cada característica contribuye de manera independiente a la probabilidad de una clase específica, sin tener en cuenta las posibles relaciones o interdependencias entre ellas [28]. En 1967, Thomas Cover y Peter Hart introdujeron el algoritmo de K-vecinos más Cercanos (*K-Nearest Neighbor*, K-NN). Este método etiqueta una determinada instancia en base a la mayoría de etiquetas que poseen sus

k-vecinos más cercanos [29]. Uno de los trabajos más importantes fue presentado en 1995 por Corinna Cortes y Vladimir Vapnik, máquinas de vectores de soporte (*Support Vector Machine*, SVM), este algoritmo encuentra el hiperplano que separa las diferentes clases en el espacio de características de manera que maximiza la distancia entre las muestras más cercanas de cada clase. Es conocido por su eficacia en problemas de clasificación binaria [30]. Otro tipo de algoritmos utilizados en clasificación son los Árboles de Decisión (*Decision Tree*, DT). Estos se basan en una estructura de árbol y funcionan como un flujo de preguntas. Cada pregunta representa un nodo de decisión que divide el conjunto de datos en subconjuntos más pequeños hasta llegar a los nodos terminales, donde se obtiene una decisión final (etiqueta). Entre los trabajos más influyentes se encuentran los algoritmos propuestos por John Ross Quinlan [31, 32]. En 2001, Leo Breiman introdujo los Bosques Aleatorios (*Random Forest*, RF), que son un conjunto de árboles de decisión entrenados con diferentes subconjuntos del conjunto de datos; la decisión final se toma en función de la decisión de muchos árboles [33].

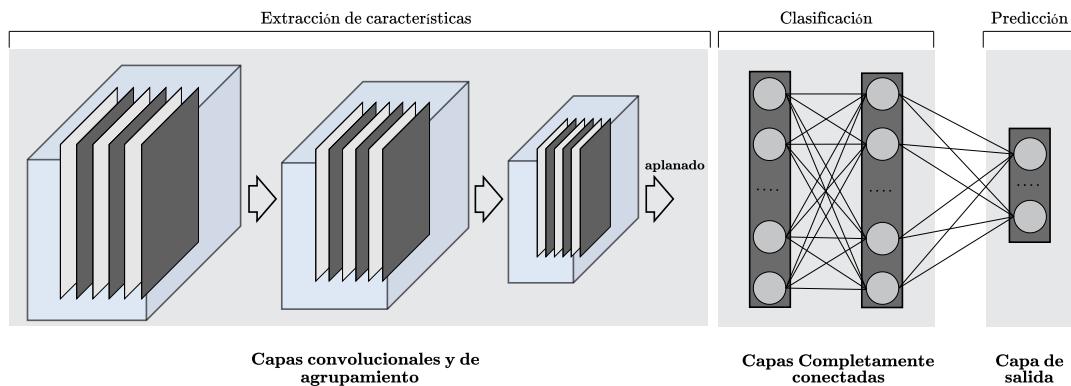
Técnicas más modernas, como las Redes Neuronales Convolucionales (*Convolutional Neural Networks*, CNNs) y los Transformadores Visuales (*Visual Transformers*, ViT), integran la extracción de características y la clasificación dentro de una única arquitectura [4, 10, 34].

## 2.2. Redes neuronales artificiales

En esta sección se revisan redes neuronales utilizadas en clasificación de imágenes como redes neuronales convolucionales y transformadores visuales.

### 2.2.1. Redes neuronales convolucionales

Esta sección explora los componentes básicos de las redes neuronales convolucionales (*Convolutional Neural Network*, CNN) (ver Figura 2.4), lo cual es fundamental para entender su funcionamiento y decisiones tomadas en el diseño de arquitecturas expuestas más adelante.



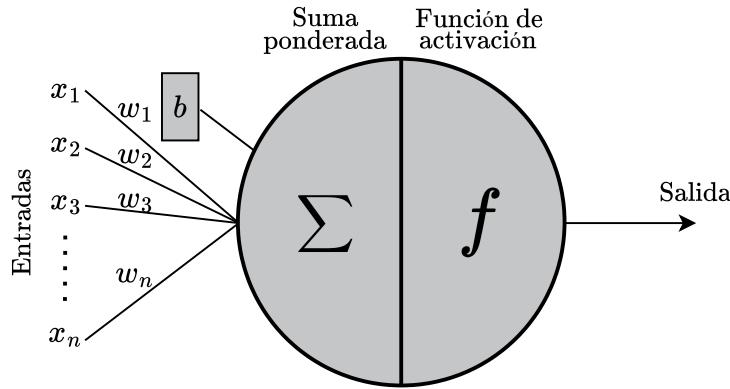
**Figura 2.4:** Red neuronal convolucional. Fuente: Adaptado de [4].

## Neurona

Las neuronas son la unidad elemental del sistema nervioso, estas reciben, procesan y envían información hacia otras neuronas en forma de señales químicas o eléctricas. Inspiradas en el funcionamiento de las neuronas biológicas, se han creado las neuronas artificiales, también conocidas como perceptrones. Como es mostrado en la Figura 2.5, al igual que las neuronas biológicas, las neuronas artificiales procesan matemáticamente múltiples entradas para producir una respuesta que puede ser transmitida. La salida de una neurona se define por:

$$f \left( b + \sum_{i=1}^n (x_i \cdot w_i) \right), \quad (2.1)$$

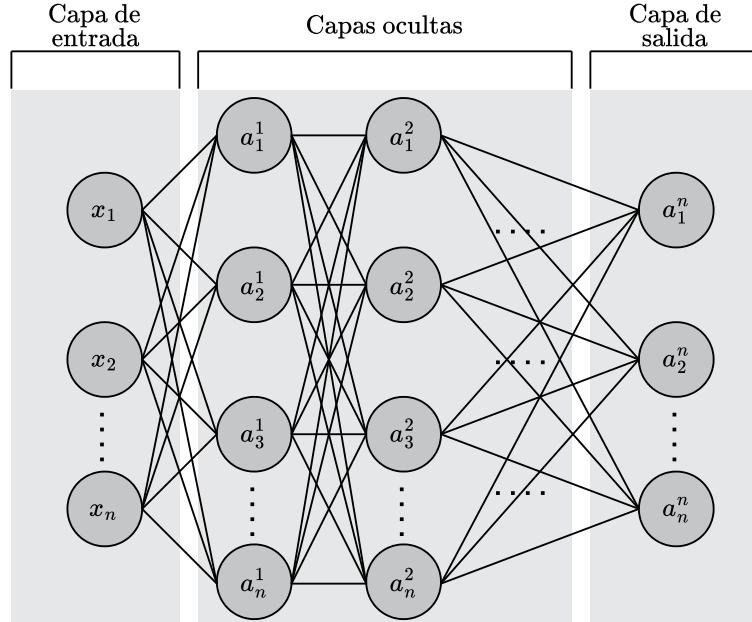
donde  $f(\cdot)$  es la función de activación (se detallará más adelante),  $x_i$  representa la señal de entrada,  $n$  es el número total de señales,  $w_i$  es el peso asignado a la entrada  $x_i$ , y  $b$  es el sesgo [4, 10].



**Figura 2.5:** Neurona artificial. Fuente: Adaptado de [4].

### Perceptrón multicapa

Perceptrón multicapa (*Multi-Layer Perceptron*, MLP) como se muestra en la Figura 2.6, se constituye de una capa de entrada, una o más capas ocultas y una capa de salida. Cada neurona en una determinada capa, se encuentra conectada a todas las neuronas de la capa anterior y a todas las neuronas de la capa siguiente, por lo tanto, las salidas de las neuronas de una capa, son las entradas a las neuronas de la siguiente capa, permitiendo la transferencia de información entre capas a través de la red. Las capas anteriores en una red neuronal aprenden características generales de los datos, mientras que las capas posteriores se especializan en aprender características cada vez más específicas y detalladas [4, 10].



**Figura 2.6:** Red perceptrón multicapa. Fuente: Adaptado de [4].

El proceso de aprendizaje en un MLP se basa en dos fases principales: *feedforward* y *backpropagation*. En la fase de *feedforward*, la información se propaga desde la capa de entrada hasta la capa de salida, generando predicciones basadas en las características aprendidas. En la fase de *backpropagation*, se calcula el error entre las predicciones y las respuestas reales, y este error se utiliza para ajustar los pesos de la red mediante un algoritmo de optimización. La actualización para un determinado peso está dado por:

$$w_{\text{nuevo}} = w_{\text{antiguo}} - \alpha \cdot \frac{\partial E}{\partial w}, \quad (2.2)$$

donde  $w_{\text{nuevo}}$  es el valor actualizado del peso,  $w_{\text{antiguo}}$  es el valor del peso antes de la actualización,  $\alpha$  es la tasa de aprendizaje y  $\frac{\partial E}{\partial w}$  es el gradiente de la función error  $E$  con respecto al peso  $w$  [4].

### Capa convolucional

Una capa convolucional es un componente fundamental en las CNNs. Esta capa está compuesta por varios filtros convolucionales, cada uno de los cuales contiene pesos, generalmente con dimensiones de 3x3 (9 pesos), 5x5 (25 pesos) o 7x7 (49

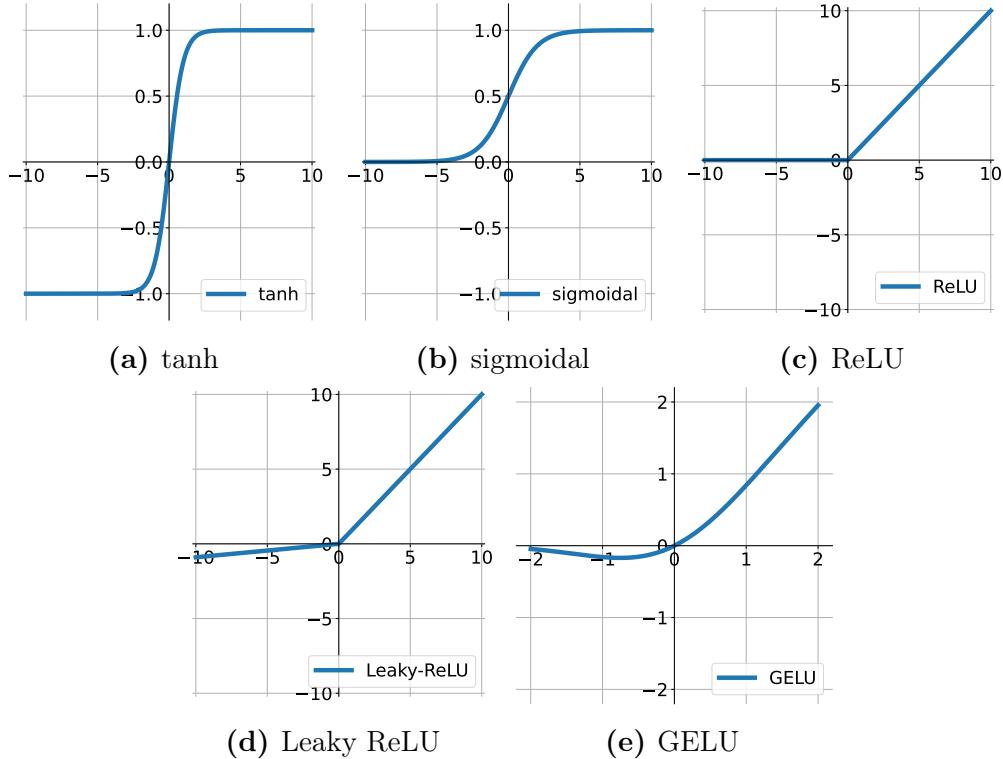
pesos). El número de canales de cada filtro debe ser igual al número de canales de la imagen de entrada. Estos filtros recorren la imagen en pequeñas regiones, píxel por píxel, extrayendo características relevantes y generando mapas de características. Las capas anteriores extraen características generales de las imágenes, como líneas, bordes y texturas. Por otro lado, las capas posteriores se enfocan en extraer información más específica y compleja, como formas y patrones detallados [4, 10].

### **Capa de *pooling* (agrupamiento)**

Una capa de *pooling* es otro componente importante en las CNNs. Su función es reducir las dimensiones espaciales (ancho y alto) de los mapas de características, disminuyendo así la cantidad de parámetros y el costo computacional, mientras se retienen las características más relevantes. Existen varios tipos de *pooling*, siendo los más comunes el *max pooling* (agrupamiento máximo) y el *average pooling* (agrupamiento promedio) [4, 10].

### **Función de activación**

La suma ponderada realizada por un perceptrón es una operación lineal que relaciona las entradas con la salida. Para abordar la necesidad de clasificar datos que no son linealmente separables, se introducen las funciones de activación, también conocidas como funciones de activación no lineales. Como muestra la Figura 2.7, estas funciones añaden no linealidad a la red y mejoran su desempeño. Existen distintas funciones de activación, entre ellas se encuentran: tanh, que ajusta todos los valores entre -1 y 1; *sigmoidal*, que ajusta todos los valores a una probabilidad entre 0 y 1; *softmax*, una generalización de la función *sigmoidal* que se utiliza cuando hay dos o más clases; ReLU, que actúa como una función identidad para valores mayores a cero y es 0 para cualquier entrada menor o igual a cero; *Leaky ReLU*, que a diferencia de ReLU introduce una pendiente aproximadamente de 0.01 para valores negativos; y GELU, que utiliza una distribución gaussiana para decidir cuánto de la entrada pasa a la siguiente capa. Debido a la rapidez de la función ReLU en el entrenamiento con descenso de gradiente, es la más utilizada en las capas ocultas de las redes neuronales, mientras que para problemas multiclase se suele utilizar la función *softmax* en la capa de salida, ya que proporciona una distribución de probabilidad sobre las diferentes clases [4, 5, 35, 36, 37, 38].



**Figura 2.7:** Funciones de activación. Fuente: Adaptado de [10].

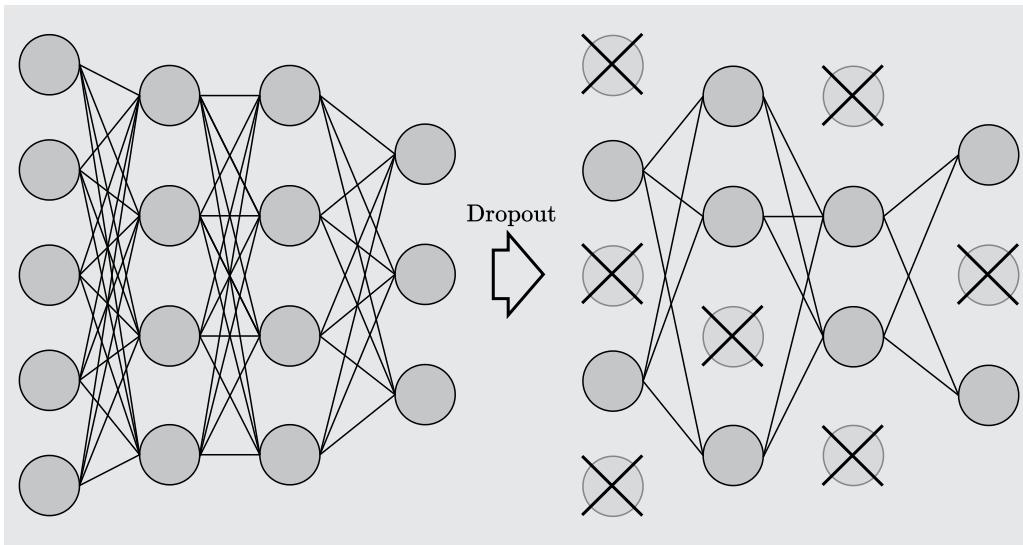
### Capa completamente conectada

Las capas completamente conectadas (*Fully-Connected*, FC), también conocidas como capas densas, se ubican después de las capas encargadas de la extracción de características. Estas capas están conformadas por neuronas que están completamente conectadas a todas las neuronas de la capa anterior y a todas las neuronas de la capa posterior. Las capas completamente conectadas utilizan las características extraídas por las capas anteriores para realizar la clasificación de los datos. En otras palabras, actúan como el clasificador de la red [4, 10].

### Técnicas de regularización

Problemas como el sobreajuste y el subajuste están directamente relacionados con la complejidad de los datos de entrada en comparación con la complejidad de la red neuronal. Para reducir el riesgo de sobreajuste y mejorar la capacidad de generalización del modelo, se utilizan técnicas de regularización. Como se muestra en la Figura 2.8, *Dropout* implica apagar aleatoriamente una fracción de las neuronas durante el entrenamiento, lo que evita que el modelo dependa

demasiado de patrones específicos del conjunto de entrenamiento. Por otro lado, la regularización  $L2$  añade una penalización proporcional al cuadrado de los valores de los pesos a la función de pérdida, ayudando a controlar la magnitud de los pesos. Además, la normalización por lotes ajusta y escala las salidas de las neuronas en cada capa para mantener activaciones en un rango más estable, acelerando el entrenamiento y mejorando la estabilidad del modelo. La detención temprana es otra técnica utilizada para prevenir el sobreajuste, donde alguna métrica del rendimiento del modelo es monitoreada y evaluada según un criterio predefinido para decidir si el entrenamiento debe continuar o detenerse [4, 10, 39].



**Figura 2.8:** Dropout. Fuente: Adaptación de [4].

## Función de Error

La función de error proporciona una medida del desempeño de la red basada en las predicciones generadas en la capa de salida. Entre las más utilizadas en problemas de clasificación se encuentran: el Error Cuadrático Medio (*Mean Squared Error*, MSE), que evalúa la diferencia promedio entre las predicciones y los valores verdaderos, y la Entropía Cruzada, que mide la disimilitud entre la distribución de probabilidad predicha y la distribución real de las clases [4, 10].

## Optimizador

Un mejor desempeño de la red está directamente relacionado con la reducción

del error. Una vez definida la función de error, el objetivo es encontrar los pesos (es decir, los parámetros de la función de error) que minimicen dicho error, convirtiendo este proceso en un problema de optimización. El método de optimización más utilizado en redes neuronales es el descenso del gradiente. Este método busca minimizar la función de error actualizando los parámetros en la dirección opuesta al gradiente de la función. El tamaño de los pasos hacia el mínimo local está determinado por la tasa de aprendizaje  $\alpha$ . Algunos algoritmos de descenso del gradiente son: Descenso del Gradiente Estocástico (*Stochastic Gradient Descent*, SGD), Momentum, RMSprop, Adam y Adagrad [4, 10, 40, 41, 42].

### 2.2.1.1. Arquitecturas Clásicas

#### LeNet

Lecun y colaboradores propusieron la arquitectura LeNet-5 en 1998, cuyo objetivo era clasificar imágenes de caracteres escritos a mano [5]. Como es mostrado en la Figura 2.9, LeNet cuenta con 5 capas de pesos (capas entrenables): 3 capas convolucionales (6 filtros de 5x5, 16 filtros de 5x5 y 120 filtros de 5x5), intercaladas con capas de agrupamiento promedio, y dos capas completamente conectadas (una con 84 neuronas y otra de salida con 10 neuronas). En las capas ocultas se utiliza la función de activación *sigmoidal*, mientras que en la capa de salida se emplea *softmax*.

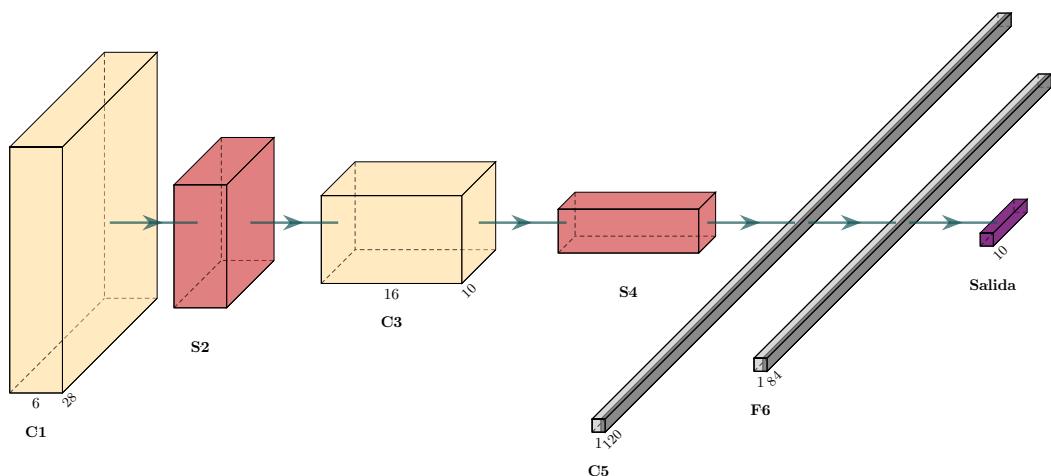


Figura 2.9: Arquitectura de LeNet. Fuente: Adaptado de [4, 5].

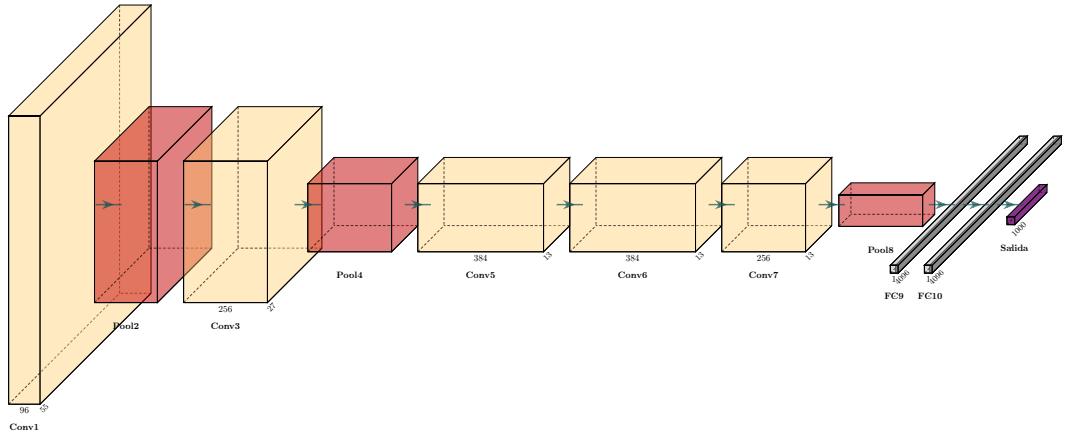
Cabe destacar que esta arquitectura fue pionera en su época y sirvió de inspiración para numerosos trabajos posteriores. A día de hoy con sus 61706 parámetros se considera una red relativamente sencilla y menos efectiva para abordar problemas más complejos.

## AlexNet

La principal motivación detrás de AlexNet fue desarrollar una red capaz de superar el desempeño en problemas más complejos [6]. Krizhevsky y sus colaboradores diseñaron una red entrenada con 1.2 millones de imágenes de alta resolución, provenientes de 1,000 clases del conjunto de datos ImageNet [1]. La red ganó de manera destacada la competencia de clasificación de imágenes ILSVRC<sup>1</sup> en 2012. Como muestra la Figura 2.10, la arquitectura consiste en 8 capas de pesos: 5 capas convolucionales (96 filtros de 11x11, 256 filtros de 5x5, 384 filtros de 3x3, 384 filtros de 3x3 y 256 filtros de 3x3), seguidas de capas de agrupamiento máximo (max pooling) después de las primeras dos y la última capa del extractor de características. Además, incluye 3 capas completamente conectadas (las dos primeras con 4096 neuronas cada una y la capa de salida con 1000 neuronas). La función de activación utilizada en las capas ocultas es ReLU, mientras que para la capa de salida se emplea *softmax*. Para prevenir el sobreajuste, se aplicó *dropout* en las capas completamente conectadas de 4096 neuronas con una probabilidad de 0.5. AlexNet aumenta considerablemente el número de parámetros a 62 millones.

---

<sup>1</sup><https://www.image-net.org/challenges/LSVRC/>. Fecha de último acceso: 24 de Febrero de 2025 a las 15:45 hrs.



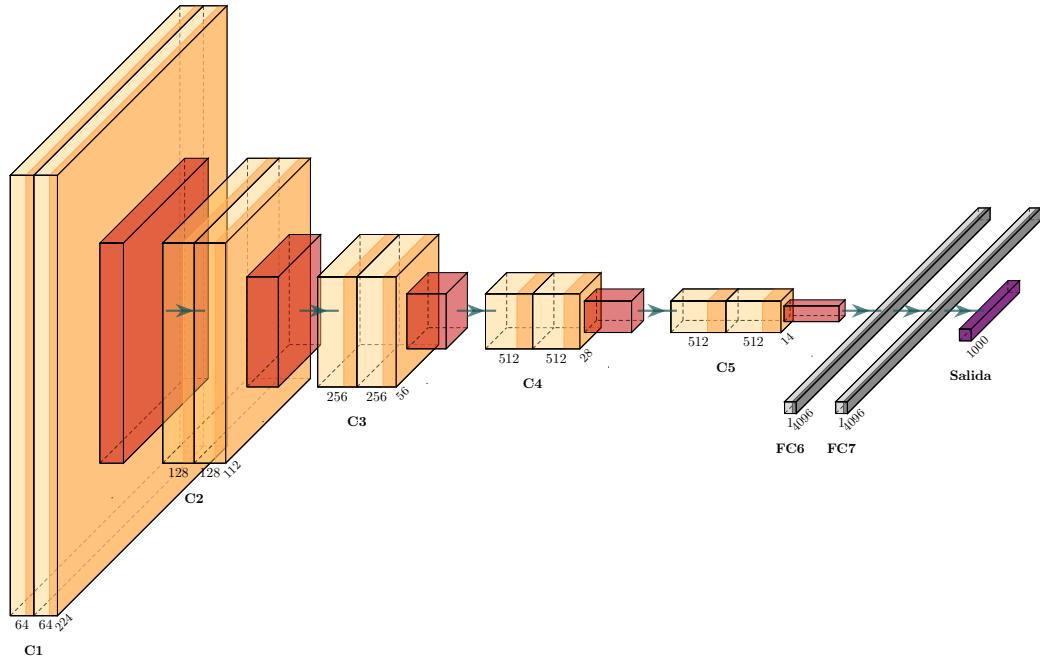
**Figura 2.10:** Arquitectura AlexNet. Fuente: Adaptado de [4, 6].

## VGGNet

Karen Simonyan y Andrew Zisserman crearon en 2014 una arquitectura aún más profunda que AlexNet, la red VGGNet [7]. Esta arquitectura facilita algunas decisiones de diseño, como la elección del tamaño de los filtros en las capas convolucionales, utilizando pequeños filtros de 3x3 en cada capa. La idea detrás de esto es que múltiples pequeños filtros sucesivos añaden más profundidad y, por lo tanto, aumentan la capacidad de la red para aprender características más complejas.

VGGNet agrupa varias capas convolucionales idénticas que mantienen las mismas dimensiones de la entrada y luego agrega una capa de agrupamiento máximo que reduce las dimensiones a la mitad. Existen distintas configuraciones de VGGNet que difieren en la organización de las capas convolucionales. Una configuración común es VGG16 mostrada en la Figura 2.11, que se conforma de un primer bloque con dos capas convolucionales de 64 filtros de 3x3, un segundo bloque con dos capas convolucionales de 128 filtros de 3x3, un tercer bloque con tres capas convolucionales de 256 filtros de 3x3, un cuarto bloque con tres capas convolucionales de 512 filtros de 3x3 y un quinto bloque con tres capas convolucionales de 512 filtros de 3x3. Luego de los bloques, hay tres capas completamente conectadas, iguales en todas las configuraciones: las dos primeras de 4096 neuronas cada una y la capa de salida de 1000 neuronas. En las capas ocultas se utiliza ReLU como función de activación y en la salida *softmax*. *Dropout* y regularización  $L2$  se emplean para evitar el sobreajuste. Esta configuración

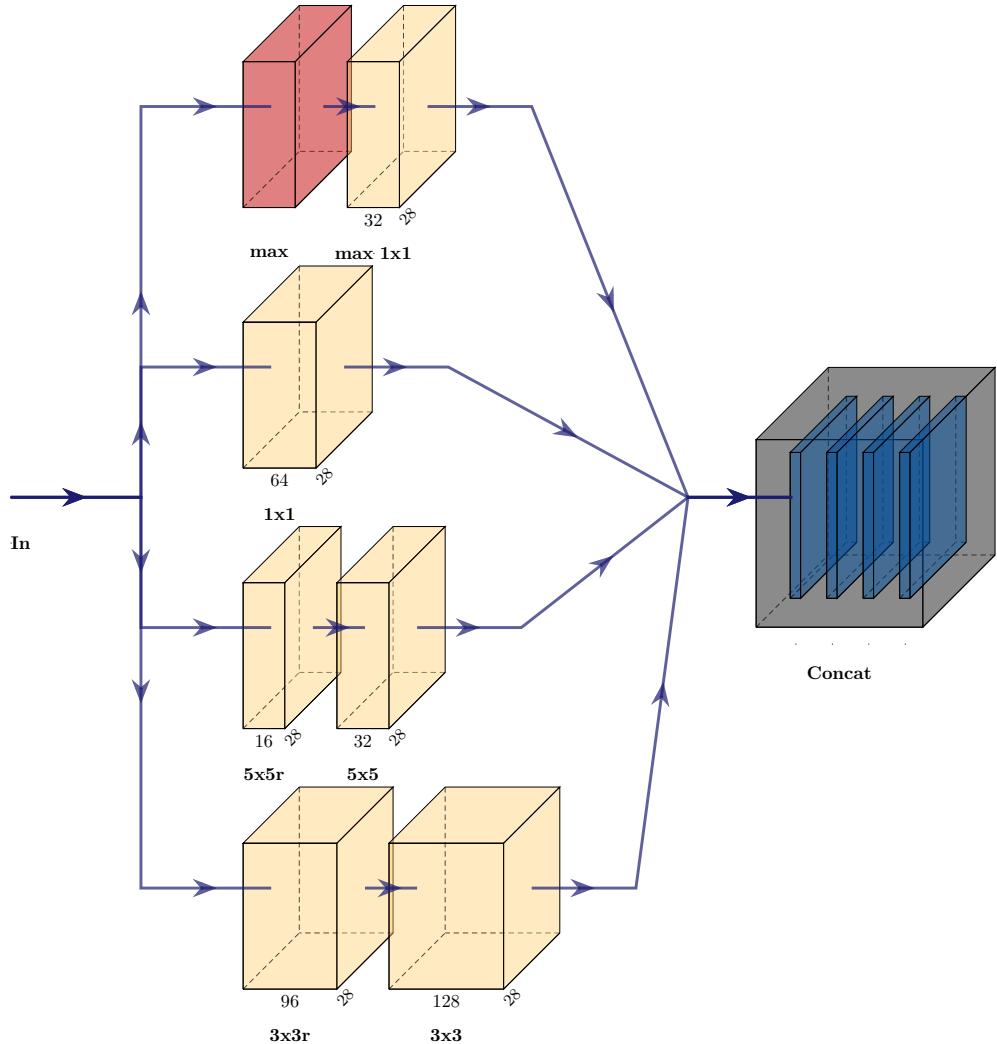
posee 138 millones de parámetros.



**Figura 2.11:** Arquitectura VGGNet16. Fuente: Adaptado de [4, 7].

### Inception (GoogLeNet)

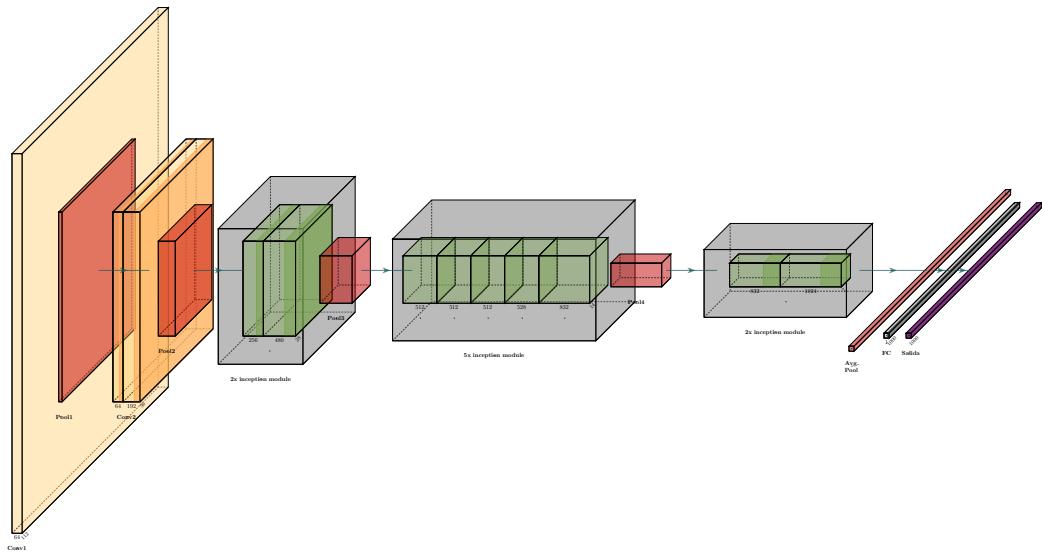
Inception fue presentada en 2014 por un grupo de investigadores de Google [8]. Esta arquitectura permite construir redes mucho más profundas con un menor número de parámetros (12 veces menos) en comparación con VGGNet e introduce un componente clave llamado módulo *Inception*. Mientras que las arquitecturas previas incluían capas convolucionales con distintos tamaños de filtros, seguidas de capas de agrupamiento intercaladas, el módulo *Inception* agrupa estas decisiones en un solo módulo. De esta forma, como se muestra en la Figura 2.12, la salida de un módulo *inception* está dada por la concatenación de la salida de 4 ramas: una con una capa convolucional de 1x1; otra con una capa convolucional de 3x3; una tercera con una capa convolucional de 5x5; y una cuarta que aplica una capa de agrupamiento máximo de 3x3. Además, dado el costo computacional que agregan los filtros de mayor tamaño (como 5x5), se añaden capas de reducción de dimensiones en cada una de las ramas, es decir, capas convolucionales de 1x1 con un bajo número de filtros.



**Figura 2.12:** Módulo Inception. Fuente: Adaptado de [4, 8].

Los creadores participaron en la competencia ILSVRC en 2014 con una versión de Inception llamada GoogLeNet. Esta arquitectura es mostrada en la Figura 2.13, la cual se compone de tres partes principales. La parte A comienza con una capa convolucional de 64 filtros de 7x7, seguida de una capa de agrupamiento máximo de 3x3. Luego, se aplica una capa convolucional con 64 filtros de 1x1 (para reducción de dimensiones), seguida de una capa convolucional con 192 filtros de 3x3, y finaliza con una capa de agrupamiento máximo de 3x3. La parte B está formada por nueve módulos Inception organizados en tres bloques principales, con 2, 5 y 2 módulos Inception, respectivamente. Cada bloque está seguido por una capa de agrupamiento: los primeros dos bloques por una capa de agrupamiento máximo de 3x3 y el último bloque por una capa de agrupamiento promedio de

7x7. Finalmente, la parte C de la arquitectura incluye una capa completamente conectada de 1000 neuronas con dropout, seguida por la capa de salida con 1000 neuronas. En las capas ocultas se utiliza la función de activación ReLU, mientras que en la capa de salida se emplea *softmax* para la clasificación.



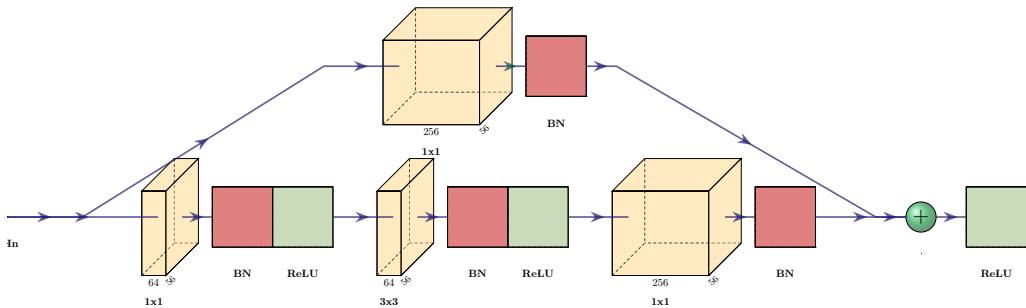
**Figura 2.13:** Arquitectura GoogLeNet. Fuente: Adaptado de [4, 8].

### Red Neuronal Residual (ResNet)

La red neuronal residual fue desarrollada en 2015 por un equipo de Microsoft Research [9]. El diseño de redes muy profundas ofrece una mayor capacidad de aprendizaje, pero también presenta dos problemas principales: el sobreajuste, que ResNet mitiga mediante un extenso uso de normalización por lotes, y el desvanecimiento del gradiente, que ocurre cuando el gradiente utilizado para actualizar los pesos se vuelve muy pequeño, impidiendo el aprendizaje efectivo en las capas anteriores. Para abordar este problema, ResNet incorpora un módulo denominado módulo residual con una conexión de salto. Esta red logró resultados destacados en la competencia ILSVRC 2015.

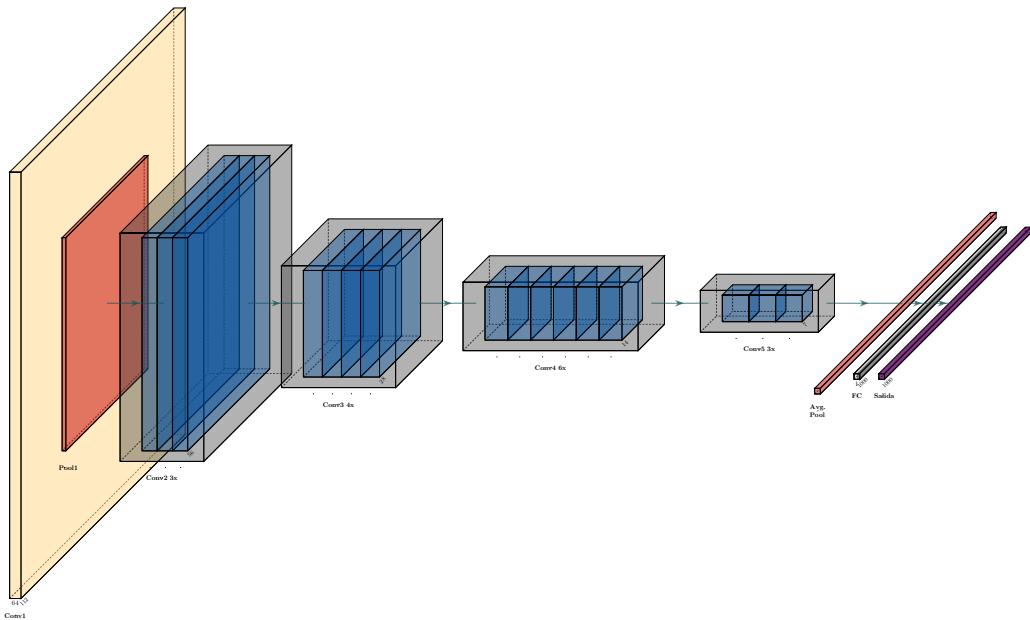
Como es mostrado en la Figura 2.14, un módulo residual se compone de dos ramas: una de ellas incluye tres capas convolucionales con filtros de 1x1, 3x3 y 1x1, respectivamente. Cada una de estas capas utiliza normalización por lotes y la función de activación ReLU. La otra rama es una conexión de salto que suma

directamente la entrada del módulo a la salida de la primera rama antes de aplicar la función de activación en la última capa convolucional. Cuando las dimensiones de la entrada y la salida de la rama convolucional no coinciden, se utiliza una conexión de salto con reducción, lograda mediante una capa convolucional de 1x1 en la rama de la conexión de salto, que ajusta las dimensiones de la entrada para que coincidan con las de la salida del bloque residual, permitiendo así la suma entre la entrada y la salida.



**Figura 2.14:** Módulo ResNet (conexión residual). Fuente: Adaptado de [4, 9].

Al igual que VGGNet, ResNet tiene distintas configuraciones que varían en la profundidad: ResNet-18, ResNet-34, ResNet-50, ResNet-101 y ResNet-152. En el caso de ResNet-50, mostrada en la Figura 2.15, esta tiene una primera parte que se compone de una capa convolucional con 64 filtros de 7x7 con función de activación ReLU, seguida de una capa de agrupamiento máximo de 3x3. Luego, la red se divide en cuatro bloques de módulos residuales. El primer bloque contiene tres módulos con capas convolucionales de 64, 64, y 256 filtros respectivamente; el segundo bloque contiene cuatro módulos con capas convolucionales de 128, 128, y 512 filtros respectivamente; el tercer bloque incluye seis módulos con capas convolucionales de 256, 256, y 1024 filtros respectivamente; y el cuarto bloque tiene tres módulos con capas convolucionales de 512, 512, y 2048 filtros respectivamente. Finalmente, la red incluye una capa de agrupamiento promedio y una capa completamente conectada de salida con 1000 neuronas y función de activación *softmax*.



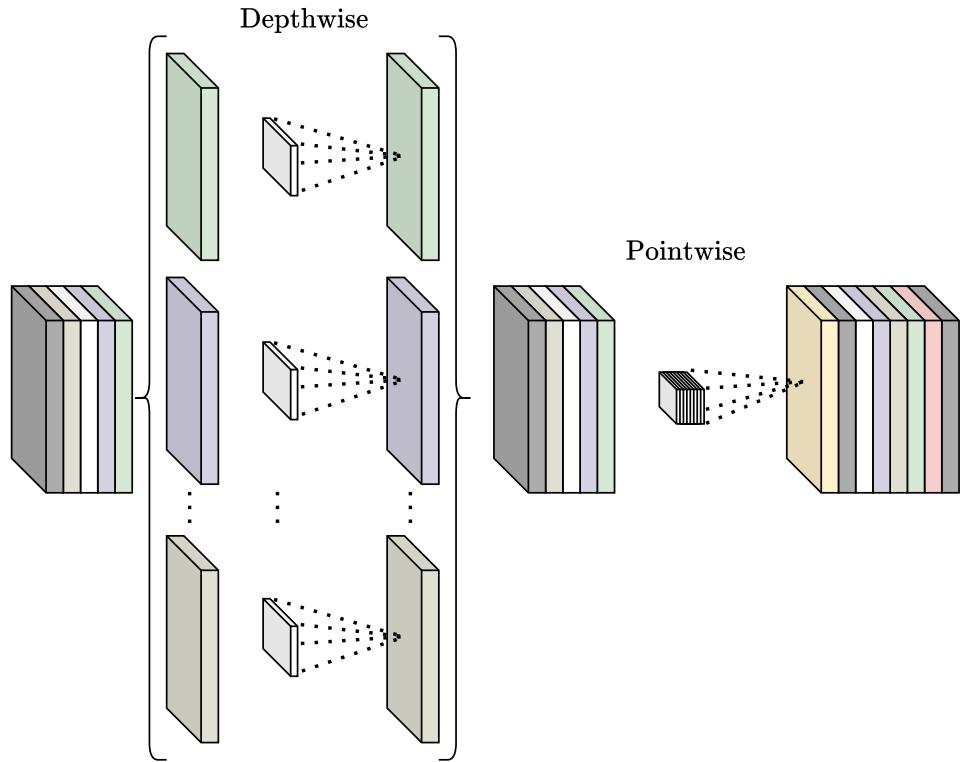
**Figura 2.15:** Arquitectura ResNet-50, Fuente: Adaptado de [4, 9].

Como se puede notar, la tendencia es diseñar arquitecturas cada vez más complejas y profundas para alcanzar un mejor desempeño. Sin embargo, esto también aumenta el número de operaciones y parámetros.

### 2.2.1.2. Arquitecturas más compactas

#### MobileNet

En 2017, Google presentó MobileNetV1 [43], una red ligera diseñada para aplicaciones en dispositivos móviles y sistemas embebidos. A diferencia de la convolución estándar, en esta red se utiliza convolución separable *depth-wise* como muestra la Figura 2.16, que consiste en una convolución *depth-wise* 3x3 seguida por una convolución *point-wise* 1x1, esta convolución reduce entre 8 a 9 veces el número de operaciones de la convolución estándar. Además, introduce 2 hiperparámetros: *width multiplier*  $\alpha$ , que disminuye el número de canales uniformemente en cada capa, y *resolution multiplier*  $\rho$ , que reduce la resolución uniformemente en cada capa. La elección de estos hiperparámetros resulta en un balance entre latencia y exactitud.



**Figura 2.16:** Convolución *depthwise-separable*. Fuente: Adapatado de [44].

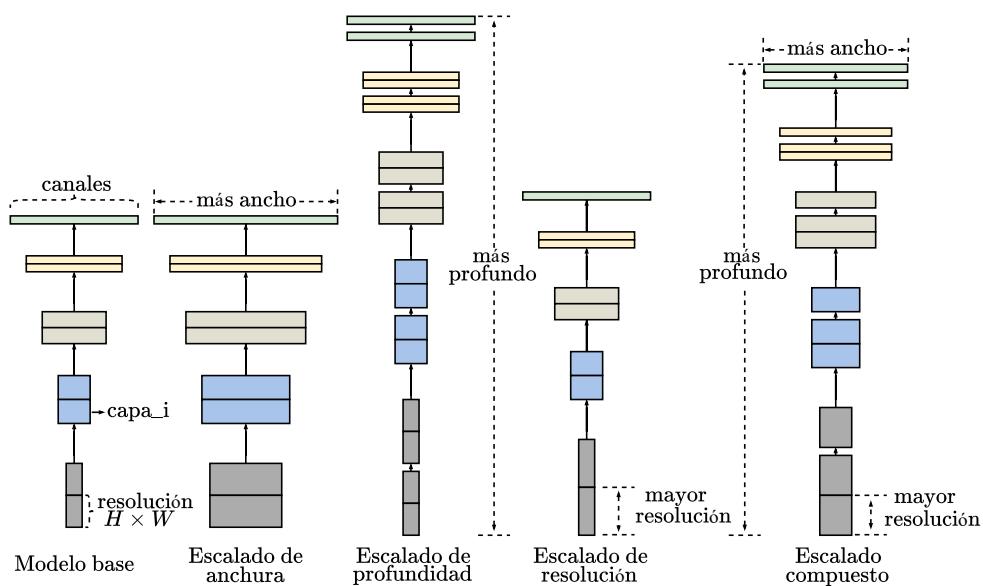
## MobileNetV2

En 2018, nuevamente Google, presenta MobileNetV2 [45], esta arquitectura introduce *inverted residuals*, estos bloques buscan solucionar la pérdida de información relevante que puede ocurrir al utilizar ReLU en espacios de baja dimensionalidad, es decir, en mapas de características de pocos canales. A diferencia del módulo ResNet estándar que sigue la secuencia 1x1 (compresión) → 3x3 → 1x1 (expansión), *inverted residuals* sigue la secuencia 1x1 (expansión) → 3x3 → 1x1 (compresión), donde la última capa corresponde a un *linear bottleneck*, esta capa no aplica no-linealidad y de esta forma evita pérdida de información. Dado que los *linear bottlenecks* poseen toda la información necesaria, la conexión residual es entre dichas capas.

## EfficientNet

En 2019, Tan y Le presentaron EfficientNet [46], una familia de redes convolucionales que introduce un innovador método de escalado compuesto. Como muestra la Figura 2.17, esta técnica permite ampliar de forma uniforme

la profundidad, el ancho y la resolución de la red, partiendo de una arquitectura base, EfficientNet-B0, que utiliza bloques de convolución invertida similares a los de MobileNetV2, complementados con módulos de *squeeze-and-excitation*. El escalado compuesto asigna coeficientes fijos para aumentar simultáneamente el número de capas, la cantidad de canales y la resolución de las imágenes de entrada. Como resultado, la familia EfficientNet, que abarca modelos desde EfficientNet-B0 hasta EfficientNet-B7, ha demostrado superar a arquitecturas anteriores en tareas de clasificación, ofreciendo mejoras en desempeño y eficiencia computacional.

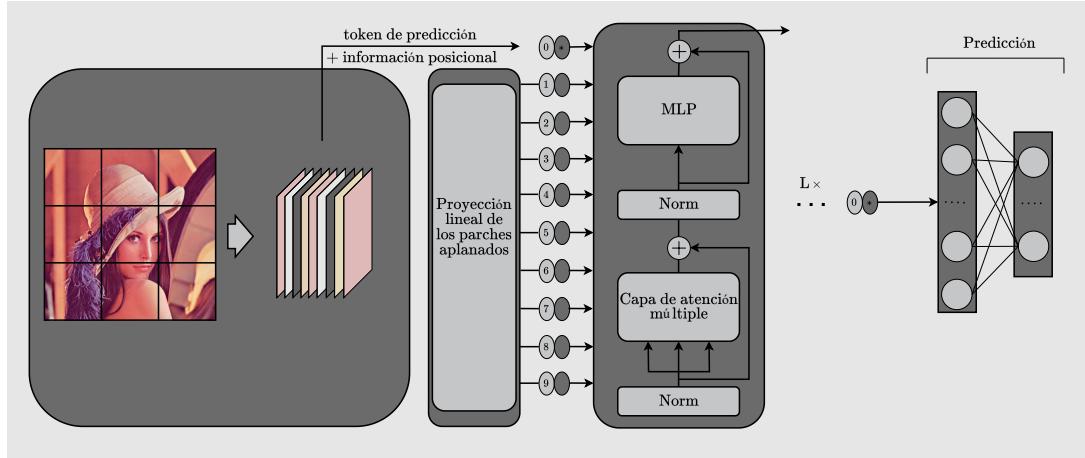


**Figura 2.17:** Escalamiento en redes EfficientNet. Fuente: Adaptado de [46].

## 2.2.2. Transformadores visuales

Los transformadores visuales (*Visual Transformer*, ViT) fueron introducidos en 2021 por Alexey Dosovitskiy y sus colaboradores [34]. Estos modelos, inspirados en la exitosa investigación de Ashish Vaswani y su equipo en 2017 [47], que se centró en el procesamiento del lenguaje natural (NLP), aprovechan un mecanismo de atención para capturar relaciones entre parches dentro de las imágenes.

Esta sección describe las partes y procesos esenciales que conforman un ViT (ver Figura 2.18).



**Figura 2.18:** Arquitectura de transformadores visuales. Fuente: Adaptado de [34].

### Embebido de parches

En una etapa inicial, las imágenes son subdivididas en parches de igual dimensión. Si las dimensiones de una imagen son  $(H, W, C)$ , donde  $H$  es la altura,  $W$  es el ancho, y  $C$  es el número de canales, y las dimensiones de cada parche son  $(P, P)$ , entonces el número de parches (subdivisiones de la imagen) está dado por  $N = H \times W / P^2$ . Luego, cada parche es aplandado (flattened) para formar un vector de características de dimensiones  $(1, P^2 \times C)$ , llamado token. Estos tokens se ajustan mediante una proyección lineal entrenable a una dimensión  $D$ , de modo que cada token tiene dimensiones  $(1, D)$ . Junto a los token, es concatenado un *token de predicción*, este token obtiene información del resto de tokens y es utilizado para realizar la predicción final. Además, se añade (suma) información relativa a la posición de los tokens dentro de las imágenes. Finalmente, una secuencia de tokens es enviada hacia el transformador [34, 47].

### Atención

Es el mecanismo que busca relaciones entre las distintas subdivisiones (parches) de las imágenes. A través de este proceso, el modelo aprende a comprender el contexto global de la imagen evaluando la importancia relativa de cada parche en función de los demás parches presentes. El cálculo de la atención se define por la siguiente fórmula:

$$\text{attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V, \quad (2.3)$$

donde  $Q$ ,  $K$  y  $V$  son las proyecciones lineales de las consultas, claves y valores, respectivamente, y  $d_k$  es la dimensión de las claves [34, 47].

Además, *atención múltiple* permite ejecutar  $k$  operaciones de atención en paralelo denominadas *cabezas*, donde cada cabeza utiliza proyecciones lineales independientes, generalmente con dimensiones  $D/k$ . Las salidas de todas las cabezas se concatenan y proyectan a la dimensión original  $D$ . De esta manera, el modelo puede captar aspectos diferentes en las relaciones de las entradas. Esta operación es realizada por la *capa de atención múltiple* [34, 47].

### Codificador

Un codificador, es una capa que se compone de dos subcapas: atención múltiple y perceptrón multicapa. A la salida de ambas subcapas, se añade normalización por lotes y una conexión de salto (conexión residual). Un transformador visual incluye varios codificadores [34, 47].

### Predicción

Una vez que una determinada entrada pasa a través todas las capas de codificación, el *token* de predicción es separado de los demás *token* y pasado a un clasificador compuesto de una o más capas de densas. Este se encarga de generar la predicción final del modelo [34, 47].

Las arquitecturas ViT son más robustas ante imágenes ruidosas o imágenes aumentadas que las CNNs, si bien el mecanismo de atención permite aprender de mejor forma las características de las imágenes, un ViT requiere grandes volúmenes de imágenes para generalizar, mientras que las CNNs generalizan con una menor cantidad imágenes [48].

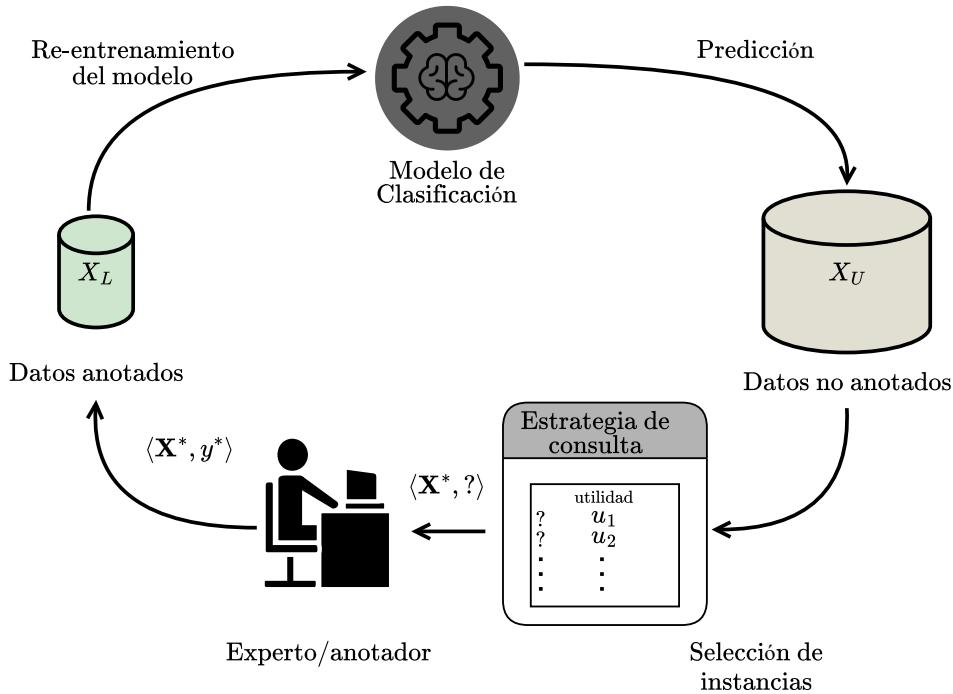
## 2.3. Aprendizaje activo

Para el entrenamiento de clasificadores de imágenes basados en redes neuronales, es necesario recolectar suficiente cantidad de imágenes etiquetadas (datos de entrenamiento). Si bien en ciertas áreas, la tarea de obtener datos no etiquetados es lograda con facilidad, el proceso de etiquetado puede volverse costoso en tiempo y recursos, por factores como la necesidad de un anotador experto o una serie de pasos previos a la anotación. En escenarios de tal tipo, AL ofrece una solución que disminuye los costos involucrados en el etiquetado. La idea detrás de AL es que un modelo de aprendizaje automático puede alcanzar un buen desempeño siendo entrenado con menos datos etiquetados, si es capaz de seleccionar progresivamente los datos de los cuales aprende [11, 12].

Esta sección revisa las ideas claves y componentes fundamentales de algoritmos de aprendizaje activo.

### 2.3.1. ¿Qué es el aprendizaje activo?

AL es un sub-área del aprendizaje automático. Como se muestra en la Figura 2.19, con la intención de superar los problemas asociados al proceso de etiquetado, AL selecciona un pequeño conjunto de las instancias (datos) más informativas desde un conjunto de datos no etiquetados, para consultar a un anotador (experto) la etiqueta correspondiente. Este difiere del aprendizaje tradicional (*Passive Learning*, PL), en el cuál las instancias son seleccionadas aleatoriamente [11, 12].



**Figura 2.19:** Pasos de un algoritmo de AL con escenario *pool-based*. Fuente: Adaptado de [11, 12].

### 2.3.2. Escenarios de AL

Hay tres principales escenarios en los que ALs pueden consultar por etiquetas. En *membership query synthesis* el modelo de aprendizaje activo genera instancias de forma sintética. El gran problema que presenta esta configuración es que las instancias generadas podrían no ser interpretables por un anotador humano. Sin embargo, en aplicaciones experimentales puede ser aprovechada esta característica [11, 12]. En *stream-based selective sampling*, instancias no etiquetadas son seleccionadas una a la vez y consultadas según una medida de su información contenida. Esto lo vuelve una opción aplicable cuando los recursos de memoria o procesamiento son escasos [11, 12]. El escenario más conocido, y más utilizado en clasificación de imágenes es *pool-based*. En esta configuración, se obtiene una medida de la información de todas/algunas instancias en un gran conjunto de datos no etiquetados  $X_U$  a partir de una estrategia de consulta  $q$  que utiliza un clasificador  $h$  entrenado en un pequeño conjunto de datos etiquetados  $X_L$ . Las  $n_q$

instancias más informativas son consultadas a un experto  $E$  para ser etiquetadas, de este modo, el clasificador es re-entrenado hasta que se cumple un criterio de detención  $SC$  [11, 12].

---

**Algoritmo 1** Enfoque de aprendizaje activo tipo *pool*.

---

- 1: **Sea:**
  - 2:  $X_L, Y_L$ : Imágenes de entrenamiento etiquetadas
  - 3:  $X_U$ : Imágenes no etiquetadas
  - 4:  $X_q$ : Imágenes seleccionadas
  - 5:  $Y_q$ : Etiquetas para  $X_q$
  - 6:  $E$ : Experto en el dominio
  - 7:  $h$ : Algoritmo de clasificación
  - 8:  $q$ : Estrategia de consulta
  - 9:  $n_q$ : Cantidad de ejemplos seleccionados
  - 10:  $SC$ : Criterio de detención
  - 11:
  - 12: **Inicialización:**
  - 13: Seleccionar  $n_q$  ejemplos desde  $X_U$  y luego etiquetarlos con  $E$  para definir un conjunto inicial de entrenamiento  $(X_I, Y_I)$
  - 14: Actualizar  $X \leftarrow X \cup X_I$
  - 15: Actualizar  $Y \leftarrow Y \cup Y_I$
  - 16:
  - 17: **Selección:**
  - 18: **while**  $SC$  no se haya cumplido **do**
  - 19:     Entrenar un algoritmo de clasificación  $h$  usando  $(X_L, Y_L)$
  - 20:     Seleccionar  $n_q$  ejemplos  $X_q$  desde  $X_U$  usando  $q(\cdot)$
  - 21:     Actualizar  $X_U \leftarrow X_U \setminus X_q$
  - 22:     Solicitar etiquetas  $Y_q$  para  $X_q$  utilizando  $E$
  - 23:     Actualizar  $X_L \leftarrow X_L \cup X_q$
  - 24:     Actualizar  $Y_L \leftarrow Y_L \cup Y_q$
  - 25: **end while**
- 

### 2.3.3. Estrategias de consulta

Lo que diferencia un algoritmo de AL de otro, es la estrategia de consulta empleada, es decir, la forma en que el modelo de aprendizaje activo evalúa cuáles son las instancias más informativas [11, 49]. Un enfoque basado en la información plantea seleccionar las instancias de las cuales el modelo está más incierto [11, 49]. Por ejemplo, Lewis y Gale, propusieron seleccionar la instancia más incierta para un clasificador probabilístico (0.5 en clasificación binaria) [50]. Otra técnica presentada por Schefer et al., calcula el margen entre las dos etiquetas más probables para una instancia y selecciona las instancias con el menor margen [11, 49]. En la selección

basada en entropía, se elige la instancia con la mayor entropía, considerando todas las posibles etiquetas [11, 12, 13]. Otro método denominado consulta por comité (QBC) fue propuesto por Seung et al., emplea múltiples clasificadores y selecciona las instancias en las que el comité de clasificadores presenta mayor desacuerdo [51]. Otro enfoque es el basado en la representación, que se centra en la estructura (características, relaciones) de los datos no etiquetados, para representar completamente el espacio de entrada. Settles et al., presentaron una estrategia de consulta que selecciona instancias desde las zonas con mayor densidad de instancias [13]. Otros métodos agrupan (*clustering*) los datos no etiquetados y luego consultan por aquellas instancias que encuentran más cercanas a los centroides [52, 53].

## 2.4. Acelerador Hardware

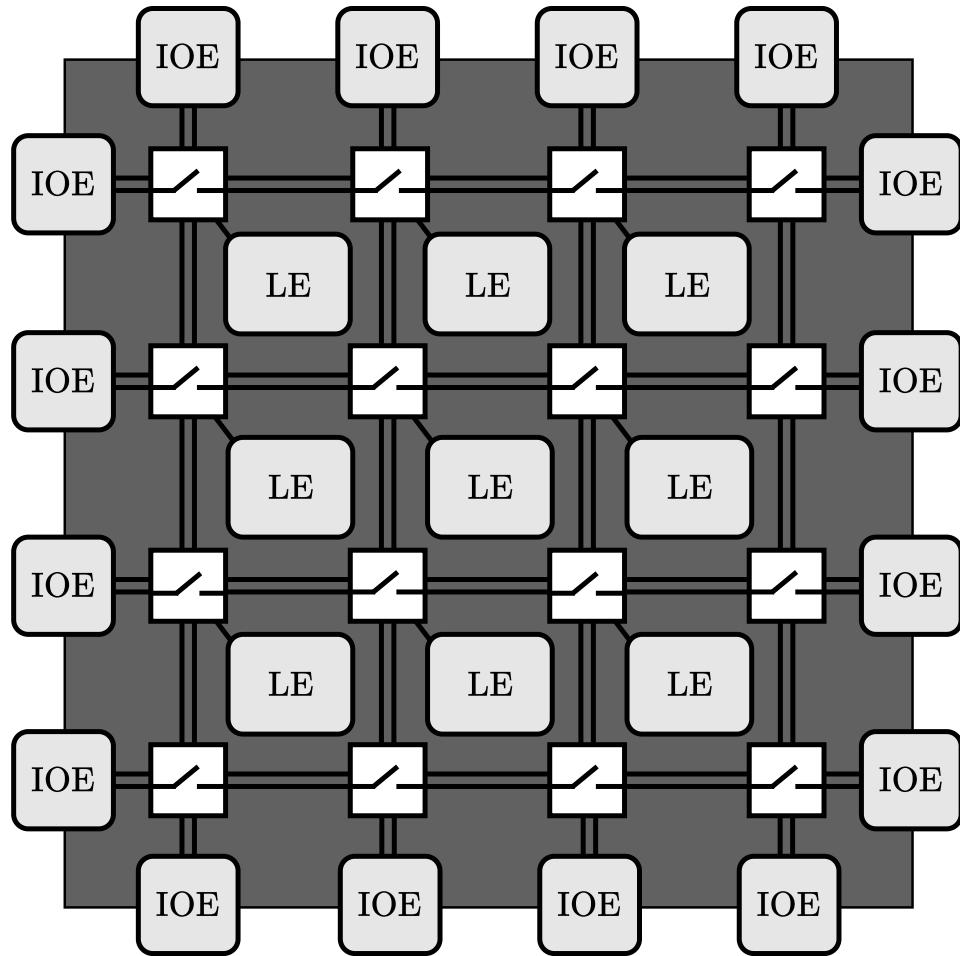
En muchas aplicaciones computacionales, como el procesamiento de señales, la simulación numérica y la computación científica, la eficiencia y el tiempo de ejecución son factores críticos. Los algoritmos empleados en estas áreas suelen involucrar una gran cantidad de operaciones matemáticas intensivas, lo que genera una alta demanda de recursos computacionales [14, 54, 55]. Tradicionalmente, los sistemas basados en arquitecturas de propósito general, como las CPUs, ejecutan estos algoritmos de manera secuencial, donde cada operación se procesa individualmente en la unidad aritmética lógica (*Arithmetic Logic Unit*, ALU), bajo el control de la CPU [15]. Sin embargo, este enfoque serial puede convertirse en un cuello de botella cuando se requiere un alto rendimiento computacional.

Para mitigar esta limitación, se emplean arquitecturas que explotan el paralelismo en distintas formas. Las GPUs, por ejemplo, aprovechan un gran número de núcleos para procesar múltiples operaciones en paralelo, lo que ha permitido grandes avances en la aceleración de aplicaciones científicas y de cómputo intensivo [14, 15]. No obstante, en escenarios donde el consumo de energía, el tamaño o la latencia son factores clave, se consideran alternativas como los FPGAs (arreglos de compuertas programables) y los circuitos integrados de aplicación específica (*Application-Specific Integrated Circuit*, ASIC). Los FPGAs permiten la implementación de arquitecturas específicas con alta eficiencia energética y flexibilidad, mientras que los ASICs ofrecen un rendimiento superior a costa de una menor reconfigurabilidad

y mayores costos de fabricación [14, 15].

#### 2.4.1. ¿Qué es un FPGA?

Los FPGAs son circuitos integrados reconfigurables. La idea detrás de los FPGAs es tener un circuito genérico, que puede ser re-programado para adoptar un propósito específico que se ajusta a los requerimientos de una aplicación [15, 56]. De esta forma, un FPGA permite paralelizar cómputos y lograr implementaciones más eficientes en términos de velocidad, consumo de potencia, ancho de banda y utilización de recursos [15, 56]. Como muestra la Figura 2.20, los componentes principales de un FPGA son los Elementos Lógicos (*Logic Element*, LE), también conocidos como Bloques Lógicos Configurables (*Configurable Logic Block*, CLB). Estos bloques permiten implementar funciones de lógica combinacional y secuencial. Los LEs se organizan en una estructura de rejilla y están interconectados mediante una matriz de conexiones reconfigurables, que permite conectar los bloques de diversas maneras. Además, el arreglo de LEs está rodeado por Elementos de Entrada/Salida (*Input/Output Element*, IOE) que facilitan la interacción con el exterior [14, 15, 56].



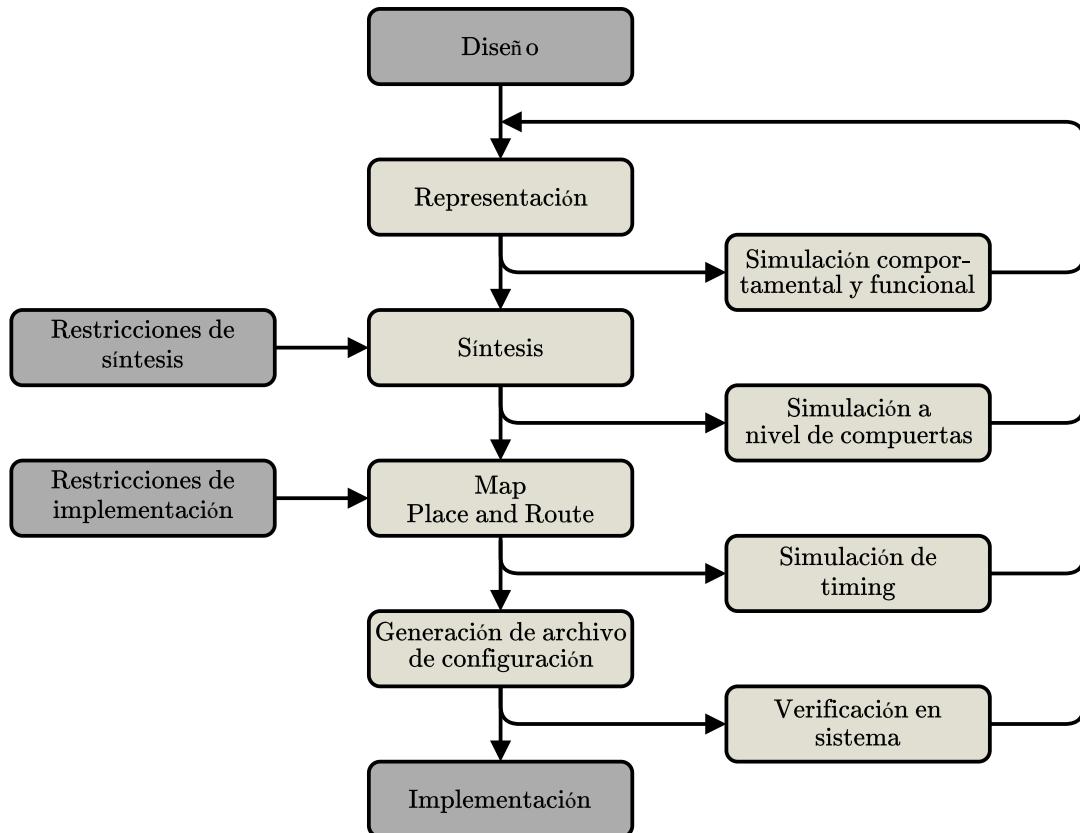
**Figura 2.20:** Arquitectura de un FPGA. Fuente: Adaptado de [15, 56]

Además, FPGAs modernas integran otros bloques funcionales como: multiplicadores, interfaces de entrada/salida de alta velocidad, procesadores de señales digitales (*Digital Signal Processing*, DSP), e incluso CPUs [15, 56].

Representar un circuito utilizando los recursos lógicos de un FPGA puede tornar la programación del dispositivo tediosa y propensa a errores. Además, la portabilidad entre distintos dispositivos se complica debido a las diferencias en su arquitectura. Para superar estas limitaciones, se han desarrollado los *Lenguajes de Descripción de Hardware* (*Hardware Description Language*, HDL), que permiten implementar una arquitectura digital a un nivel de abstracción superior mediante la descripción de su comportamiento lógico. Otra alternativa es la *Síntesis de Alto Nivel* (*High Level Synthesis*, HLS), que facilita el diseño de *hardware* a partir de lenguajes de alto nivel, como C/C++. Esta metodología simplifica la transición de software a

*hardware*, aunque ofrece un menor control sobre los detalles que proporcionan los HDL [15, 56].

Implementar un diseño implica seguir una serie de pasos, como se ilustra en la Figura 2.21.



**Figura 2.21:** Flujo de diseño en FPGA. Fuente: Adaptado de [15].

En primer lugar, el **Diseño** consiste en definir un circuito digital que cumpla con los requerimientos de la aplicación. En la fase de **Respresentación**, se describe el diseño mediante HDL o HLS, de modo que sea comprensible tanto para humanos como para la máquina. La etapa de **Síntesis** transforma esta descripción en un circuito funcional implementado en compuertas lógicas. Posteriormente, **Map** asigna la lógica a los recursos disponibles en el FPGA. Finalmente, en la fase de **Place-and-Route**, la lógica mapeada se asocia a bloques específicos en el FPGA y se determinan las rutas necesarias para interconectar dichos bloques [15].

En el diseño de *hardware* en FPGA *parallelización* y *pipeline* son técnicas esenciales para mejorar el rendimiento y la eficiencia en el procesamiento. La *parallelización*

consiste en dividir una tarea en múltiples sub-tareas que pueden ejecutarse simultáneamente, lo que permite aprovechar al máximo los recursos disponibles y procesar grandes volúmenes de datos en un ciclo de reloj. Por otro lado, el *pipeline* es una técnica que divide un proceso en varias etapas secuenciales. Cada etapa realiza una parte específica de la tarea, y, gracias a la concurrencia, diferentes datos pueden estar en distintas fases del proceso al mismo tiempo. Esto reduce la latencia y aumenta el *throughput* (capacidad de procesamiento) del sistema. De este modo, cada bloque del *pipeline* procesa una parte del flujo de datos, permitiendo que, una vez que un dato avanza a la siguiente etapa, el bloque liberado pueda comenzar a procesar otro dato [15].

Existen numerosos trabajos que buscan acelerar el proceso de inferencia de clasificadores de imágenes [16, 17]. E incluso, hay un creciente desarrollo de *frameworks* que permiten reducir el tiempo de diseño de algoritmos de aprendizaje automático en FPGAs para su configuración e implementación, como HLS4ML<sup>2</sup>. Además, dado lo costoso y tardío que puede volverse hacer pruebas y mediciones de distintas arquitecturas *hardware*, se han propuesto métodos que utilizan aprendizaje activo en el proceso de búsqueda de configuraciones *hardware* que reducen la latencia de modelos de aprendizaje automático [18, 19]. Por ejemplo Ji *et al.* proponen un modelo de predicción de latencia donde el AL permite seleccionar únicamente aquellas arquitecturas de redes neuronales que aportan la información más útil para entrenar el modelo, evitando medir la latencia de todas las opciones posibles y reduciendo el costo computacional [18].

## 2.5. Discusión

Las redes neuronales convolucionales han demostrado ser una opción confiable para la clasificación de imágenes debido a su capacidad para extraer características espaciales de entradas en dos dimensiones y realizar la clasificación a través de capas de neuronas completamente conectadas. En este contexto, se han destacado en la clasificación de imágenes, mostrando una alta efectividad para reconocer patrones complejos y realizar predicciones más precisas en comparación con enfoques tradicionales [5, 6, 9, 23, 29, 46].

---

<sup>2</sup><https://fastmachinelearning.org/hls4ml/>. Fecha de último acceso: 3 de Marzo de 2025 a las 22:19 hrs.

Sin embargo, el rendimiento de una CNN está estrechamente relacionado con la calidad y cantidad de imágenes, que en un marco de aprendizaje supervisado deben estar etiquetadas. El aprendizaje activo propone reducir los esfuerzos en la anotación de datos, permitiendo al modelo seleccionar aquellos más informativos. Estudios previos han demostrado que el uso de algoritmos de aprendizaje activo en un escenario *pool-based* permite obtener resultados superiores a los logrados con aprendizaje pasivo, reduciendo así la cantidad de datos de entrenamiento necesarios [57, 58].

Un aspecto fundamental es el *hardware* sobre el que se ejecutan los algoritmos de aprendizaje automático. Aunque se han logrado avances en este sentido, siendo las GPUs la principal plataforma de implementación, en términos de velocidad de cómputo aún hay margen de mejora a través de *hardware* específico. En respuesta a esto, ha surgido un creciente interés en el uso de *hardware* re-configurable, como las FPGAs, para diseñar soluciones de aplicación específica que aumenten la capacidad de procesamiento de estos algoritmos. Sin embargo, aunque existen trabajos que combinan aprendizaje activo y FPGAs, no se ha explorado en profundidad la aceleración del proceso de selección de los ejemplos más informativos desde el conjunto de datos no etiquetado [18, 19]. Esto sugiere que aún existe un espacio para explorar el uso de arquitecturas *hardware* que aceleren los cálculos asociados con las estrategias de consulta.

## Capítulo 3

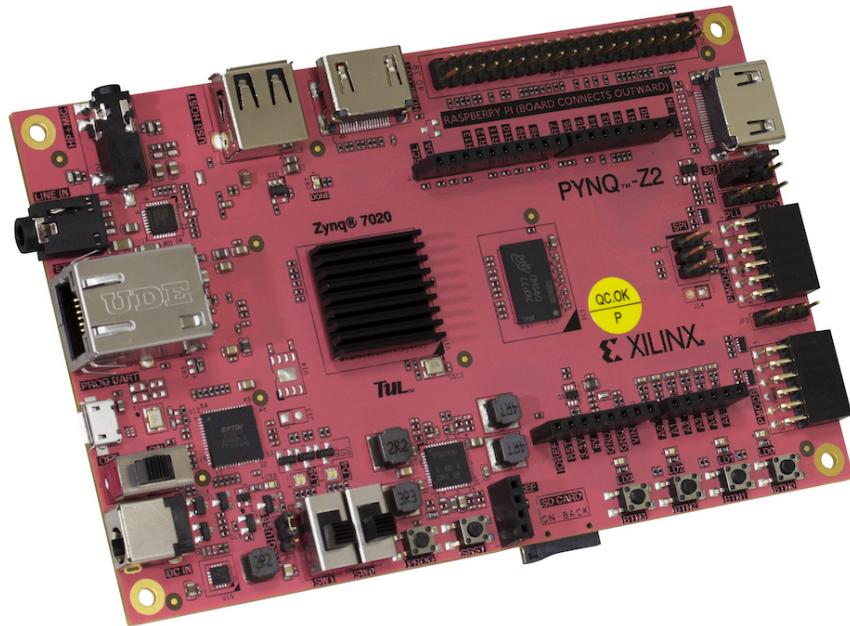
# Materiales y Métodos

Este capítulo presenta la plataforma de *hardware* utilizada, PYNQ-Z2 con el SoC ZYNQ-7000, junto con sus características principales. Además, se describen los conjuntos de datos Fashion-MNIST y CIFAR-10, así como el pre-procesamiento aplicado. Finalmente, se define el problema abordado y se expone la metodología, que incluye la evaluación en *software* y la implementación en *hardware* para acelerar una estrategia de consulta en aprendizaje activo.

### 3.1. Plataforma de aceleración

En este trabajo se utiliza la tarjeta de desarrollo PYNQ-Z2 de Xilinx, mostrada en la Figura 3.1, que incluye un SoC ZYNQ-7000 XC7Z020-1CLG400C. Las características principales de la plataforma son [59]:

- Procesador ARM Cortex-A9 de doble núcleo de hasta 667[MHz] de operación.
- 512[MB] de memoria DDR3 de 1050[Mbps] de ancho de banda.
- Un reloj de 125[MHz] para la lógica programable.
- Puertos de entrada y salida de video HDMI.
- 2 interruptores, 4 pulsadores, 4 LEDs y 2 LEDs RGB.



**Figura 3.1:** Tarjeta de desarrollo PYNQ-Z2 de Xilinx. Fuente: Adaptado de <https://www.tulembedded.com/FPGA/ProductsPYNQ-Z2.html>. Fecha de último acceso: 20 de Febrero de 2025 a las 18:36 hrs.

La lógica programable dentro del SoC es equivalente a un FPGA Artix-7 con las siguientes características [59]:

- 85.000 celdas lógicas programables (CLB).
- 53.200 Look-Up Tables (LUTs).
- 106.400 flip-flops.
- 140 BRAM equivalentes a 630[KB].
- 220 slices DSP.
- Memoria On-Chip de 256[KB].

La programación del procesador ARM Cortex-A9 se efectúa a través de PYNQ<sup>1</sup> (Python Productivity for ZYNQ). PYNQ es un proyecto de código abierto de AMD, que permite programar en un lenguaje de alto nivel, *Python*. En este estudio se utiliza la imagen versión 3.0.1. Este nivel de abstracción facilita el proceso de diseño y permite la creación de *Overlays*, que es un concepto análogo a lo que son

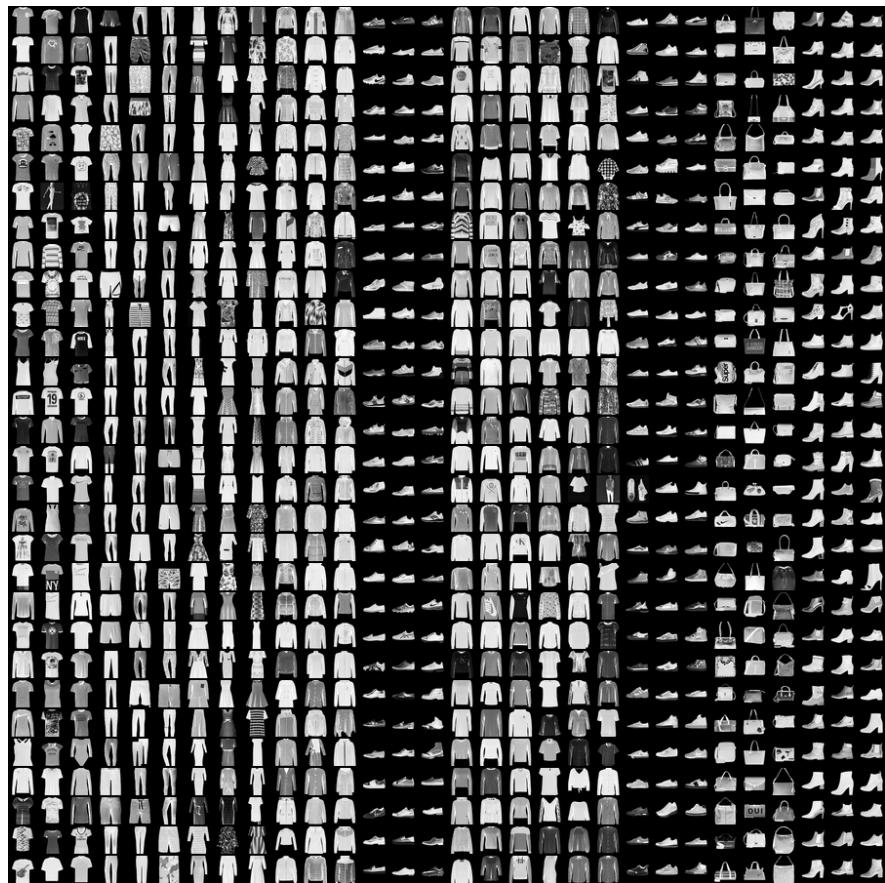
<sup>1</sup><http://www.pynq.io/>. Fecha de último acceso: 21 de Febrero de 2025 a las 14:44 hrs.

las librerías en *software* pero aplicado en *hardware*. De este modo, a través de un *Overlay* se pueden crear aplicaciones más complejas [60]. PYNQ proporciona métodos para la interacción entre el procesador y la lógica programable a través de los buses AXI disponibles. La implementación en lógica programable se desarrolla en Vivado 2024.2, el entorno de diseño de Xilinx, y la arquitectura *hardware* se describe en *Verilog*. El costo comercial de la tarjeta bordea los 130 USD.

### 3.2. Conjuntos de datos y preprocesamiento

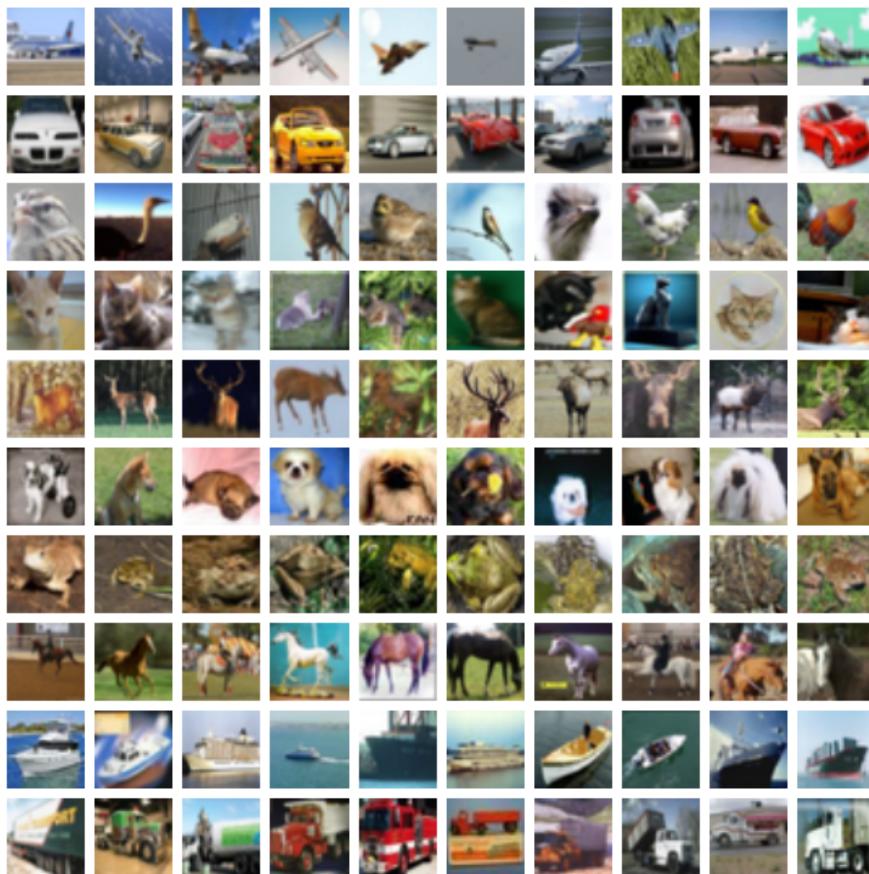
Para evaluar el método propuesto, se seleccionan los conjuntos de datos estándar Fashion-MNIST y CIFAR-10, ampliamente utilizados en tareas de clasificación. Estos conjuntos incluyen una diversidad de categorías, lo que permite analizar la capacidad de generalización del enfoque. En particular, CIFAR-10 abarca clases heterogéneas, como animales y vehículos, lo que refuerza la validez de los resultados para su aplicación en otros dominios de imágenes.

Fashion-MNIST [2] consta de 70000 imágenes en escala de grises con dimensiones de  $28 \times 28$  píxeles, correspondientes a distintas prendas de vestir. Cada clase contiene 7000 imágenes, y las categorías incluidas son: *T-shirt/top*, *Trouser*, *Pullover*, *Dress*, *Coat*, *Sandal*, *Shirt*, *Sneaker*, *Bag* y *Ankle boot*. La Figura 3.2 muestra algunas imágenes representativas del conjunto.



**Figura 3.2:** Ejemplos de imágenes en Fashion-MNIST. Fuente: Adaptado de <https://github.com/zalandoresearch/fashion-mnist>. Fecha de último acceso: 16 de Febrero de 2025 a las 09:21 hrs.

Por otro lado, CIFAR-10 [3] contiene 60000 imágenes en color de  $32 \times 32$  píxeles distribuidas en 10 clases, con 6000 imágenes por categoría. Las clases incluidas son: *airplane*, *automobile*, *bird*, *cat*, *deer*, *dog*, *frog*, *horse*, *ship* y *truck*. La Figura 3.3 presenta ejemplos de imágenes de este conjunto.



**Figura 3.3:** Ejemplos de imágenes en CIFAR-10. Fuente: Adaptado de <https://www.cs.toronto.edu/~kriz/cifar.html>. Fecha de último acceso: 16 de Febrero de 2025 a las 10:03 hrs.

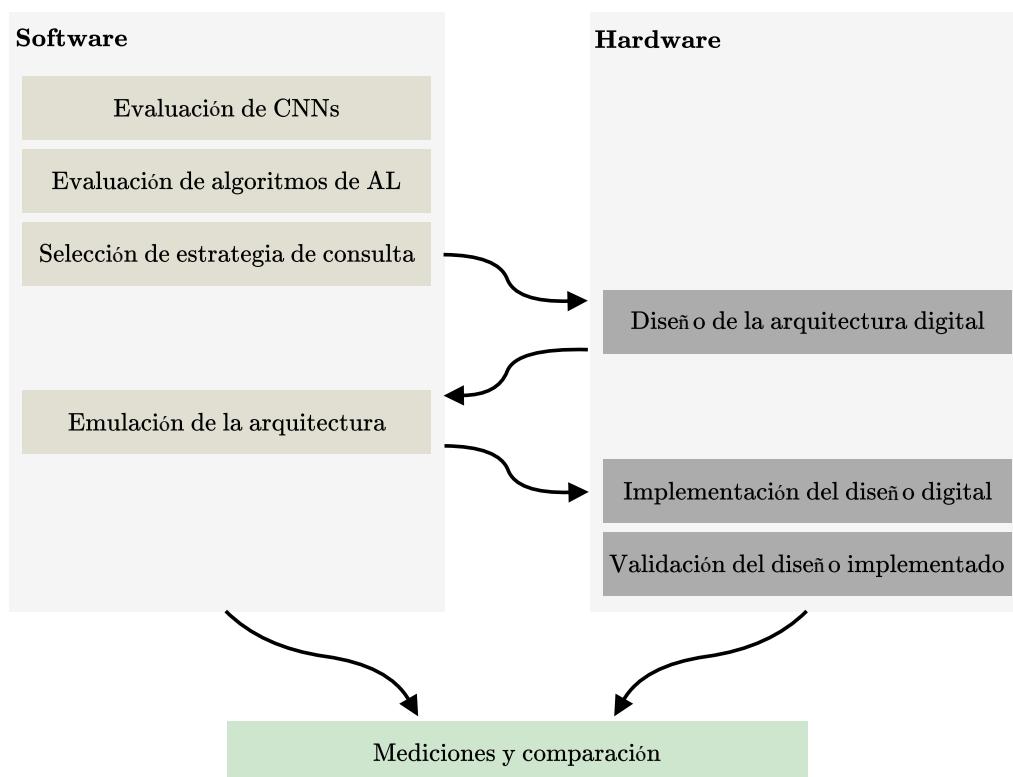
Dado el límite de recursos disponibles en la lógica programable del SoC ZYNQ-7000, se selecciona un subconjunto de 6.400 imágenes de cada conjunto de datos. Como Fashion-MNIST y CIFAR-10 contienen 10 clases, se eligen las primeras 640 imágenes por clase para los experimentos.

El pre-procesamiento aplicado a todas las imágenes se adapta a la arquitectura de red utilizada. En primer lugar, todas las imágenes se redimensionan a  $224 \times 224$  píxeles. Además, en el caso de Fashion-MNIST, que contiene imágenes en escala de grises, el canal único se replica en los tres canales RGB para mantener la compatibilidad con la arquitectura de la red [6, 7, 8, 9, 43, 46, 57].

### 3.3. Definición del problema y metodología

Los algoritmos de aprendizaje activo involucran un proceso iterativo y un gran volumen de datos, lo que puede requerir un alto consumo de recursos computacionales. Este trabajo propone acelerar parcialmente un algoritmo de aprendizaje activo mediante su implementación en *hardware*. En particular, se busca acelerar la estrategia de consulta que haya demostrado mejor desempeño en las pruebas realizadas en *software* con distintos modelos de clasificación sobre los conjuntos de datos CIFAR-10 y Fashion-MNIST.

El método propuesto se divide en dos etapas principales, como resume la Figura 3.4: (i) *software* y (ii) *hardware*.



**Figura 3.4:** Diagrama de flujo de la metodología propuesta. Fuente: Elaboración propia.

La fase (i) se compone de los siguientes pasos:

- Evaluación del desempeño de distintas arquitecturas CNN en los conjuntos de datos CIFAR-10 y Fashion-MNIST.

- Evaluación de algoritmos de aprendizaje activo en un escenario *pool-based*, con estrategias de consultas basadas en la información, empleando los clasificadores evaluados previamente.
- Comparación de los resultados obtenidos y selección de la estrategia de consulta más eficiente para su posterior aceleración.

Los pasos en la etapa (ii) son los siguientes:

- Diseño de una arquitectura digital para el acelerador en *hardware* de la estrategia de consulta seleccionada.
- Implementación del acelerador *hardware*, mediante descripción de la arquitectura en *Verilog*.
- Validación de la arquitectura digital implementada a través de pruebas y simulaciones.

Finalmente, se realiza un análisis comparativo entre la implementación en *software* y en *hardware*, evaluando principalmente tiempo de ejecución y consumo de potencia en un caso crítico.

## Capítulo 4

# Resultados: Clasificación y Aprendizaje Activo

Este capítulo expone los resultados obtenido por los clasificadores empleados, así como el desempeño de los algoritmos de aprendizaje activo. Se detallan los experimentos realizados, los recursos computacionales utilizados y se discuten los resultados obtenidos.

### 4.1. Desempeño de clasificadores de imágenes

Para evaluar el desempeño de algoritmos de AL en diferentes clasificadores de imágenes, se implementaron tres arquitecturas CNNs: MobileNetV1, EfficientNetB0 y ResNet50. Tanto MobileNetV1 como EfficientNetB0 fueron seleccionadas debido a que presentan una menor cantidad de parámetros y son adecuadas para aplicaciones embebidas, mientras que ResNet50 por su reconocido desempeño en tareas de clasificación y para servir como referencia comparativa [9, 43, 46]. Cada modelo se empleó como extractor de características pre-entrenado en ImageNet ILSVRC, al que se añadió una capa de clasificación final con un número de neuronas correspondiente a las clases de cada conjunto de datos. La Tabla 4.1 muestra la cantidad de parámetros, así como los parámetros entrenables de cada red.

**Tabla 4.1:** Cantidad de parámetros totales y parámetros entrenables. Fuente: Elaboración Propia.

| Red            | Parámetros Totales | Parámetros Entrenables |
|----------------|--------------------|------------------------|
| MobileNetV1    | 3,239,114          | 10,250                 |
| EfficientNetB0 | 4,062,381          | 12,810                 |
| ResNet50       | 23,608,202         | <b>20,490</b>          |

En negrita, el valor de la mayor cantidad de parámetros entrenables.

Las imágenes fueron pre-procesadas de acuerdo a la arquitectura respectiva a través de la función en *Keras* correspondiente, esto es:

- MobileNetV1: Los píxeles de entrada son escalados entre  $-1$  y  $1$  [43].
- EfficientNetB0: La función no realiza un preprocessamiento en particular, este es realizado dentro del modelo. Se esperan píxeles no normalizados en el rango de  $0$  a  $255$  [46].
- ResNet50: Conversión de espacio de color RGB a BGR, seguida de una normalización por canal mediante sustracción de las medias de ImageNet ILSVRC, sin aplicar escalado adicional [9].

Los hiperparámetros comunes utilizados en el entrenamiento se resumen en la Tabla 4.2 [61, 62, 63].

**Tabla 4.2:** Hiperparámetros utilizados en el entrenamiento de las CNNs. Fuente: Elaboración propia.

| Hiperparámetro      | Configuración    |
|---------------------|------------------|
| Tasa de aprendizaje | 0.001            |
| Tamaño de lote      | 32               |
| Épocas máximas      | 100              |
| Optimizador         | Adam             |
| Función de error    | Entropía cruzada |

Adicionalmente, se implementa un criterio de detención temprana basado en el error de validación, con paciencia de 10 épocas.

El desempeño de los clasificadores fue evaluado usando validación cruzada estratificada de 5 particiones, corriendo los experimentos 5 veces y promediando

los resultados de las métricas *Accuracy* (ACC) y *F1-Score* (F1) [61, 64]. ACC y F1 son calculadas como muestran las ecuaciones 4.1 y 4.2 respectivamente [4]:

$$ACC = \frac{TP + TN}{TP + FP + TN + FN}, \quad (4.1)$$

$$F1 = \frac{2TP}{2TP + FP + FN}, \quad (4.2)$$

donde  $TP$ ,  $TN$ ,  $FP$  y  $FN$  son, respectivamente, los resultados de clasificación verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos. En el caso de los valores F1, son ponderados de acuerdo con el número de ejemplos en los datos de prueba.

Los resultados comparativos en cuanto a ACC y F1 se presentan en la Tabla 4.3.

**Tabla 4.3:** Desempeño de CNNs en los distintos conjuntos de imágenes. Fuente: Elaboración propia.

| CNN            | ACC (%)      |               | F1 (%)       |               |
|----------------|--------------|---------------|--------------|---------------|
|                | CIFAR-10     | Fashion-MNIST | CIFAR-10     | Fashion-MNIST |
| MobileNetV1    | 84.17        | 87.35         | 84.16        | 87.23         |
| EfficientNetB0 | <b>88.25</b> | <b>89.28</b>  | <b>88.25</b> | <b>89.18</b>  |
| ResNet50       | 87.23        | 87.40         | 87.28        | 87.42         |

En negrita, los valores con el mayor desempeño en cada fila para una determinada métrica de desempeño.

La Tabla 4.3 muestra que EfficientNetB0 obtuvo el mayor ACC (88.25 % en CIFAR-10 y 89.28 % en Fashion-MNIST) y F1 (88.25 % en CIFAR-10 y 89.18 % en Fashion-MNIST). Por otro lado, MobileNetV1 presentó el menor rendimiento en ambos conjuntos de datos, con un ACC de 84.17 % y un F1 de 84.16 % en CIFAR-10, y un ACC de 87.35 % y un F1 de 87.23 % en Fashion-MNIST.

La implementación de los modelos fue hecha en la plataforma Google Colaboratory Pro, utilizando la GPU NVIDIA Tesla T4 de 16 GB de VRAM y la opción con capacidad de memoria RAM de 51 GB.

## 4.2. Aprendizaje activo

El escenario de AL en este estudio es *pool-based*, donde se asume un gran conjunto de datos no etiquetados  $X_U$  y un pequeño conjunto de datos etiquetados  $X_L$ . Dado lo anterior y que, al igual que en la evaluación de desempeño para tareas de clasificación, se empleó validación cruzada con 5 particiones, los datos se ordenan del siguiente modo: para cada partición, el 80 % de las instancias se asignó a entrenamiento, y el 20 % restante se reservó para prueba. Dentro de los datos de entrenamiento [57, 58]:

- Inicialmente, se seleccionó aleatoriamente un 10 % como conjunto etiquetado y el restante 90 % como conjunto no etiquetado.
- En cada iteración (o consulta), se etiquetó un 10 % adicional de los datos no etiquetados disponibles.
- Este proceso se repitió hasta alcanzar 10 iteraciones, momento en el que el conjunto etiquetado equivalía al 100 % de los datos de entrenamiento.

Dado el tamaño total del subconjunto de datos (6400), el porcentaje asignado a entrenamiento (80 %) y el porcentaje etiquetado en cada iteración (10 %), el número de imágenes etiquetadas por consulta se calcula como  $\text{batch} = 6400 \times 0,8 \times 0,1 = 512$ .

Este valor (512) fue elegido intencionalmente por ser una potencia de 2 ( $2^9$ ), lo que simplifica decisiones de diseño y análisis en el acelerador *hardware*.

Los experimentos realizados evalúan el desempeño de tres estrategias de consulta basadas en la incertidumbre del modelo: *Least Confident*, *Margin Sampling* y *Entropy*. Todas ellas buscan identificar aquellas instancias en el conjunto de datos no etiquetado en las que el modelo tiene menor certeza sobre su predicción, de modo que al etiquetarlas se pueda mejorar su desempeño. Cada estrategia emplea un criterio distinto para medir la incertidumbre.

*Least Confident* selecciona las instancias cuya predicción más probable tiene menor confianza. En otras palabras, se eligen los ejemplos en los que la diferencia entre 1 y la mayor probabilidad asignada es más grande. La ecuación (4.3) muestra cómo se calcula [11]:

$$\mathbf{x}^* = \underset{\mathbf{x} \in X_U}{\operatorname{argmax}} \ 1 - P_h(\hat{y}|\mathbf{x}), \quad (4.3)$$

donde  $\mathbf{x}^*$  es la instancia más incierta,  $P_h(\hat{y}|\mathbf{x})$  representa la probabilidad más alta que el modelo asigna a alguna clase para la instancia  $\mathbf{x}$ , usando el modelo  $h$ .

Aunque este método detecta instancias con alta incertidumbre, solo toma en cuenta la probabilidad de la clase más probable e ignora la relación con las demás clases.

*Margin Sampling* a diferencia de *Least Confident*, considera las dos clases con mayor probabilidad y selecciona las instancias en las que la diferencia entre ellas es más pequeña. Esto indica que el modelo tiene dificultades para decidir entre ambas clases, como se expresa en la ecuación (4.4) [11]:

$$\mathbf{x}^* = \underset{\mathbf{x} \in X_U}{\operatorname{argmin}} \ P_h(\hat{y}_1|\mathbf{x}) - P_h(\hat{y}_2|\mathbf{x}), \quad (4.4)$$

donde  $\mathbf{x}^*$  es la instancia más incierta,  $P_h(\hat{y}_1|\mathbf{x})$  y  $P_h(\hat{y}_2|\mathbf{x})$  son la primera y segunda probabilidades más altas, respectivamente.

Un margen reducido sugiere que la instancia se encuentra en una zona incierta entre dos clases, por lo que conocer su etiqueta puede ayudar al modelo a mejorar su capacidad de discriminación. Sin embargo, en problemas con muchas clases, esta estrategia podría pasar por alto información relevante de las demás clases.

Para problemas con una gran cantidad de clases, una estrategia más robusta es *Entropy*, ya que tiene en cuenta todas las probabilidades. Esta estrategia mide la incertidumbre total de una distribución de probabilidad: valores altos indican mayor incertidumbre, mientras que valores bajos sugieren predicciones más seguras. La ecuación (4.5) define este criterio [11]:

$$\mathbf{x}^* = \underset{\mathbf{x} \in X_U}{\operatorname{argmax}} - \sum_i P_h(\hat{y}_i|\mathbf{x}) \log P_h(\hat{y}_i|\mathbf{x}), \quad (4.5)$$

donde  $y_i$  abarca todas las posibles etiquetas de clase y  $P_h(\hat{y}_i|\mathbf{x})$  es la probabilidad condicional de la clase  $y_i$  para la instancia no etiquetada  $\mathbf{x}$ .

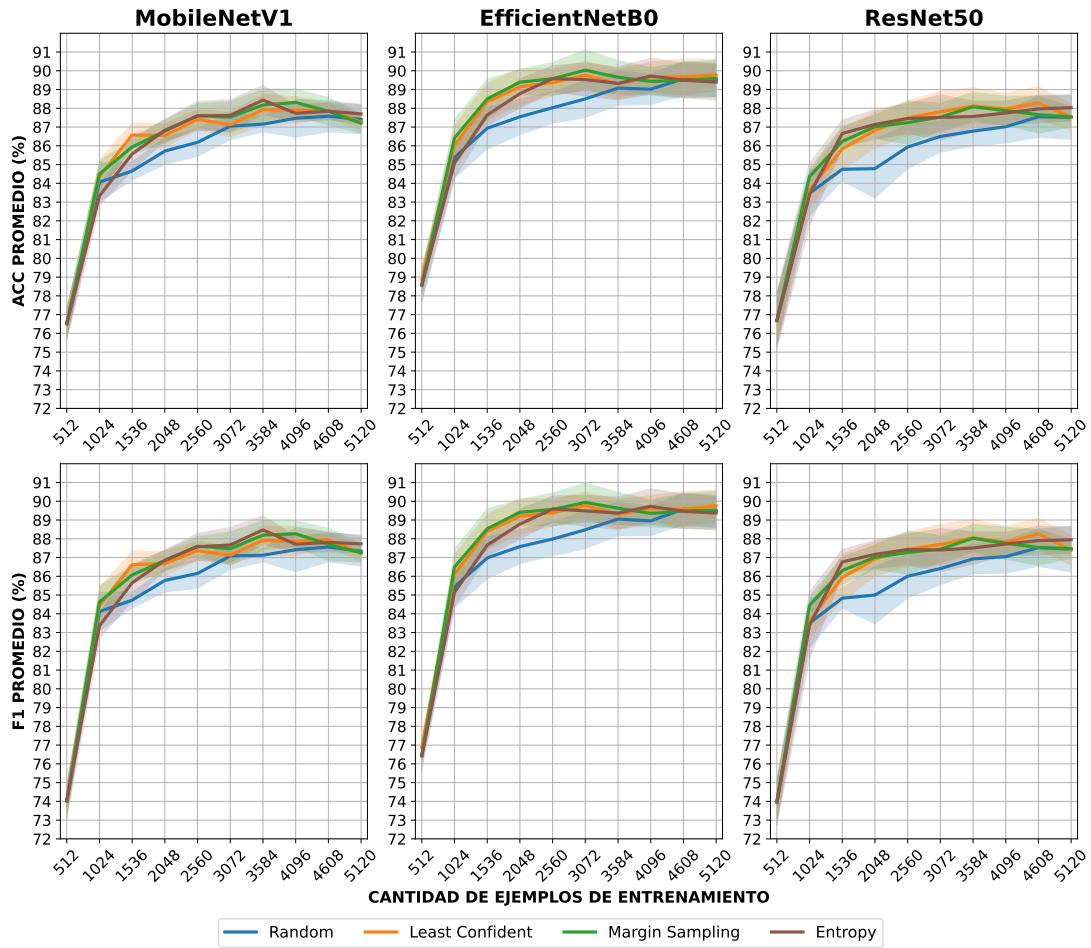
Para ilustrar cómo funcionan estas estrategias, se tienen dos instancias  $x_1$  y  $x_2$ ,

y tres clases A, B y C. Las probabilidades para  $x_1$  son (0.4, 0.2, 0.4) y para  $x_2$ , (0.35, 0.37, 0.28). Con *Least Confident*, se selecciona  $x_2$  porque la diferencia entre 1 y la mayor probabilidad (0.37) es mayor que la diferencia entre 1 y la mayor probabilidad de  $x_1$  (0.4). Con *Margin Sampling*, se selecciona  $x_1$  ya que su diferencia entre las dos clases más probables es menor ( $0.4 - 0.4 = 0$ ) frente a  $x_2$  ( $0.37 - 0.35 = 0.02$ ). Finalmente, con *Entropy*, también se selecciona  $x_2$  porque tiene mayor entropía, siendo la entropía de  $x_2$  aproximadamente 1.575 y la de  $x_1$  aproximadamente 1.571 [11].

A continuación, se presenta una comparación del desempeño de las estrategias de consulta en aprendizaje activo frente al comportamiento observado cuando las instancias son seleccionadas de forma aleatoria, es decir, en un escenario de aprendizaje pasivo. Las evaluaciones se realizaron utilizando los tres modelos de clasificación previamente descritos, aplicados a cada uno de los conjuntos de imágenes.

#### 4.2.1. Aprendizaje activo en Fashion-MNIST

La Figura 4.1 muestra las curvas de aprendizaje en términos del promedio de las iteraciones de validación cruzada junto a la desviación estándar en ACC y F1 alcanzados para una determinada cantidad de datos etiquetados, considerando los tres modelos de clasificación. En este sentido, la Tabla 4.4 muestra el área bajo cada curva de aprendizaje (AULC).



**Figura 4.1:** Curvas de aprendizaje de los algoritmos de aprendizaje activo en el conjunto de datos Fashion-MNIST utilizando MobileNetV1, EfficientNetB0 y ResNet50. Fuente: Elaboración propia.

**Tabla 4.4:** AULC en Fashion-MNIST. Fuente: Elaboración propia.

| CNN            | AULC       |                 |               |         |           |                 |               |         |
|----------------|------------|-----------------|---------------|---------|-----------|-----------------|---------------|---------|
|                | Curvas ACC |                 |               |         | Curvas F1 |                 |               |         |
|                | Random     | Least Confident | Margin        | Entropy | Random    | Least Confident | Margin        | Entropy |
| MobileNetV1    | 0.7718     | 0.7776          | <b>0.7785</b> | 0.7770  | 0.7706    | 0.7765          | <b>0.7774</b> | 0.7759  |
| EfficientNetB0 | 0.7881     | 0.7956          | <b>0.7965</b> | 0.7931  | 0.7870    | 0.7946          | <b>0.7954</b> | 0.7921  |
| ResNet50       | 0.7688     | 0.7781          | <b>0.7781</b> | 0.7778  | 0.7679    | 0.7764          | <b>0.7765</b> | 0.7762  |

En negrita, los valores de la estrategia con el mayor AULC en cada fila para una determinada métrica de desempeño.

La Figura 4.1 muestra que en general el aprendizaje pasivo se ve superado por el aprendizaje activo. En MobileNetV1, tanto *Margin* como *Entropy* alcanzan un desempeño superior al 88 % en ACC y F1 con 3584 ejemplos, equivalente al 70 % del total de datos de entrenamiento. En cuanto a EfficientNetB0, con

una reducción del 40 % de datos etiquetados *least confident*, *Margin* y *Entropy* superan el 89 % en ACC y F1. Para ResNet50, *Least Confident* y *Margin* alcanzan aproximadamente 88 % de ACC y F1 con una reducción de ejemplos etiquetados del 30 %. Por otro lado, la Tabla 4.4 muestra que el área bajo la curva es similar entre estrategias, con *Margin* presentando un mayor área para todas las curvas.

Las siguientes tablas muestran la cantidad mínima de ejemplos etiquetados necesarios para alcanzar un determinado desempeño en las métricas ACC y F1. La Tabla 4.5 presenta los resultados obtenidos con MobileNetV1, la Tabla 4.6 los correspondientes a EfficientNetB0 y la Tabla 4.7 aquellos obtenidos con ResNet50.

**Tabla 4.5:** Cantidad mínima de ejemplos requeridos por los diferentes algoritmos de aprendizaje activo para alcanzar las métricas ACC y F1 en el conjunto de imágenes Fashion-MNIST utilizando MobileNetV1. Fuente: Elaboración propia.

| ACC (%) | Random | Least Confident | Margin      | Entropy     | F1 (%)  | Random | Least Confident | Margin      | Entropy     |
|---------|--------|-----------------|-------------|-------------|---------|--------|-----------------|-------------|-------------|
| 76.42 % | —      | —               | —           | —           | 73.93 % | —      | —               | —           | —           |
| 76.52 % | 512    | 512             | 512         | 512         | 74.03 % | 512    | 512             | 512         | 512         |
| 76.62 % | 1024   | 1024            | 1024        | 1024        | 74.13 % | 1024   | 1024            | 1024        | 1024        |
| 83.42 % | 1024   | 1024            | 1024        | 1536        | 83.43 % | 1024   | 1024            | 1024        | 1536        |
| 84.12 % | 1536   | 1024            | 1024        | 1536        | 84.13 % | 1536   | 1024            | 1024        | 1536        |
| 84.42 % | 1536   | 1536            | <b>1024</b> | 1536        | 84.53 % | 1536   | 1536            | <b>1024</b> | 1536        |
| 84.52 % | 1536   | 1536            | 1536        | 1536        | 84.73 % | 2048   | 1536            | 1536        | 1536        |
| 84.72 % | 2048   | 1536            | 1536        | 1536        | 85.73 % | 2048   | 1536            | 1536        | 2048        |
| 85.62 % | 2048   | 1536            | 1536        | 2048        | 85.83 % | 2560   | 1536            | 1536        | 2048        |
| 85.82 % | 2560   | 1536            | 1536        | 2048        | 86.13 % | 2560   | <b>1536</b>     | 2048        | 2048        |
| 86.02 % | 2560   | <b>1536</b>     | 2048        | 2048        | 86.23 % | 3072   | <b>1536</b>     | 2048        | 2048        |
| 86.22 % | 3072   | <b>1536</b>     | 2048        | 2048        | 86.63 % | 3072   | 2048            | 2048        | 2048        |
| 86.62 % | 3072   | 2560            | 2048        | 2048        | 86.73 % | 3072   | 2560            | 2048        | 2048        |
| 86.82 % | 3072   | 2560            | <b>2048</b> | 2048        | 86.93 % | 3072   | 2560            | 2560        | 2560        |
| 86.92 % | 3072   | 2560            | 2560        | 2560        | 87.13 % | 4096   | 2560            | 2560        | 2560        |
| 87.12 % | 3584   | 2560            | 2560        | 2560        | 87.43 % | 4608   | 3584            | 2560        | 2560        |
| 87.22 % | 4096   | 2560            | 2560        | 2560        | 87.63 % | —      | 3584            | 3584        | <b>3072</b> |
| 87.42 % | 4096   | 3584            | 2560        | 2560        | 87.73 % | —      | 3584            | 3584        | 3584        |
| 87.52 % | 4608   | 3584            | 2560        | 2560        | 87.93 % | —      | 4608            | 3584        | 3584        |
| 87.62 % | —      | 3584            | 3584        | <b>3072</b> | 88.03 % | —      | —               | 3584        | 3584        |
| 87.72 % | —      | 3584            | 3584        | 3584        | 88.23 % | —      | —               | 4096        | <b>3584</b> |
| 88.02 % | —      | —               | 3584        | 3584        | 88.33 % | —      | —               | —           | <b>3584</b> |
| 88.22 % | —      | —               | 4096        | <b>3584</b> | 88.47 % | —      | —               | —           | —           |
| 88.32 % | —      | —               | —           | <b>3584</b> | —       | —      | —               | —           | —           |
| 88.44 % | —      | —               | —           | —           | —       | —      | —               | —           | —           |

**En negrita**, las cantidades mínimas de datos requeridas cuando son menores que en las demás estrategias. El símbolo “—” indica que no se alcanzó el desempeño especificado en la fila.

Los resultados muestran que, para valores de ACC y F1 inferiores al 86 %, las estrategias *Least Confident* y *Margin* requieren menos ejemplos en comparación con las demás, alcanzando un desempeño cercano al 86 % con solo 1536 ejemplos en ambos casos. Para desempeños superiores al 87 % y cercanos al 88 %, *Entropy* se mantiene competitivo, e incluso en algunos casos necesita menos ejemplos que

*Least Confident* y *Margin*.

**Tabla 4.6:** Cantidad mínima de ejemplos requeridos por los diferentes algoritmos de aprendizaje activo para alcanzar las métricas ACC y F1 en el conjunto de imágenes Fashion-MNIST utilizando EfficientNetB0. Fuente: Elaboración propia.

| ACC (%) | Random | Least Confident | Margin      | Entropy | F1 (%)  | Random | Least Confident | Margin      | Entropy |
|---------|--------|-----------------|-------------|---------|---------|--------|-----------------|-------------|---------|
| 78.46 % | —      | —               | —           | —       | 76.33 % | —      | —               | —           | —       |
| 78.56 % | 512    | 512             | 512         | 512     | 76.43 % | 512    | 512             | 512         | 512     |
| 78.66 % | 1024   | <b>512</b>      | 1024        | 1024    | 76.53 % | 1024   | <b>512</b>      | 1024        | 1024    |
| 78.96 % | 1024   | 1024            | 1024        | 1024    | 76.93 % | 1024   | 1024            | 1024        | 1024    |
| 85.16 % | 1024   | 1024            | 1024        | 1536    | 85.23 % | 1024   | 1024            | 1024        | 1536    |
| 85.46 % | 1536   | 1024            | 1024        | 1536    | 85.53 % | 1536   | 1024            | 1024        | 1536    |
| 86.06 % | 1536   | 1536            | <b>1024</b> | 1536    | 86.13 % | 1536   | 1536            | <b>1024</b> | 1536    |
| 86.46 % | 1536   | 1536            | 1536        | 1536    | 86.53 % | 1536   | 1536            | 1536        | 1536    |
| 86.96 % | 2048   | 1536            | 1536        | 1536    | 87.03 % | 2048   | 1536            | 1536        | 1536    |
| 87.56 % | 2560   | 1536            | 1536        | 1536    | 87.63 % | 2560   | 1536            | 1536        | 1536    |
| 87.66 % | 2560   | 1536            | 1536        | 2048    | 87.73 % | 2560   | 1536            | 1536        | 2048    |
| 88.06 % | 3072   | 1536            | 1536        | 2048    | 88.03 % | 3072   | 1536            | 1536        | 2048    |
| 88.36 % | 3072   | 2048            | <b>1536</b> | 2048    | 88.43 % | 3072   | 2048            | <b>1536</b> | 2048    |
| 88.56 % | 3584   | 2048            | 2048        | 2048    | 88.53 % | 3584   | 2048            | <b>1536</b> | 2048    |
| 88.86 % | 3584   | 2048            | 2048        | 2560    | 88.63 % | 3584   | 2048            | 2048        | 2048    |
| 89.16 % | 4608   | 2560            | <b>2048</b> | 2560    | 88.83 % | 3584   | 2048            | 2048        | 2560    |
| 89.46 % | 4608   | 3072            | 2560        | 2560    | 89.13 % | 4608   | 2048            | 2048        | 2560    |
| 89.66 % | —      | 3072            | 3072        | 4096    | 89.23 % | 4608   | 2560            | <b>2048</b> | 2560    |
| 89.76 % | —      | 3072            | 3072        | —       | 89.43 % | 4608   | 3072            | 2560        | 2560    |
| 89.86 % | —      | —               | <b>3072</b> | —       | 89.63 % | —      | 3072            | 3072        | 4096    |
| 90.03 % | —      | —               | —           | —       | 89.83 % | —      | —               | <b>3072</b> | —       |
|         |        |                 |             |         | 89.94 % | —      | —               | —           | —       |

**En negrita**, las cantidades mínimas de datos requeridas cuando son menores que en las demás estrategias. El símbolo “—” indica que no se alcanzó el desempeño especificado en la fila.

Como se observa en la Tabla 4.6, *Margin* alcanza aproximadamente un 86 % de ACC y F1 con solo 1024 ejemplos, es decir, 512 menos que las demás estrategias. Este patrón se mantiene en valores cercanos al 88.5 % y 89 %, donde nuevamente *Margin* requiere menos ejemplos que sus competidores para alcanzar el mismo rendimiento.

**Tabla 4.7:** Cantidad mínima de ejemplos requeridos por los diferentes algoritmos de aprendizaje activo para alcanzar las métricas ACC y F1 en el conjunto de imágenes Fashion-MNIST utilizando ResNet50. Fuente: Elaboración propia.

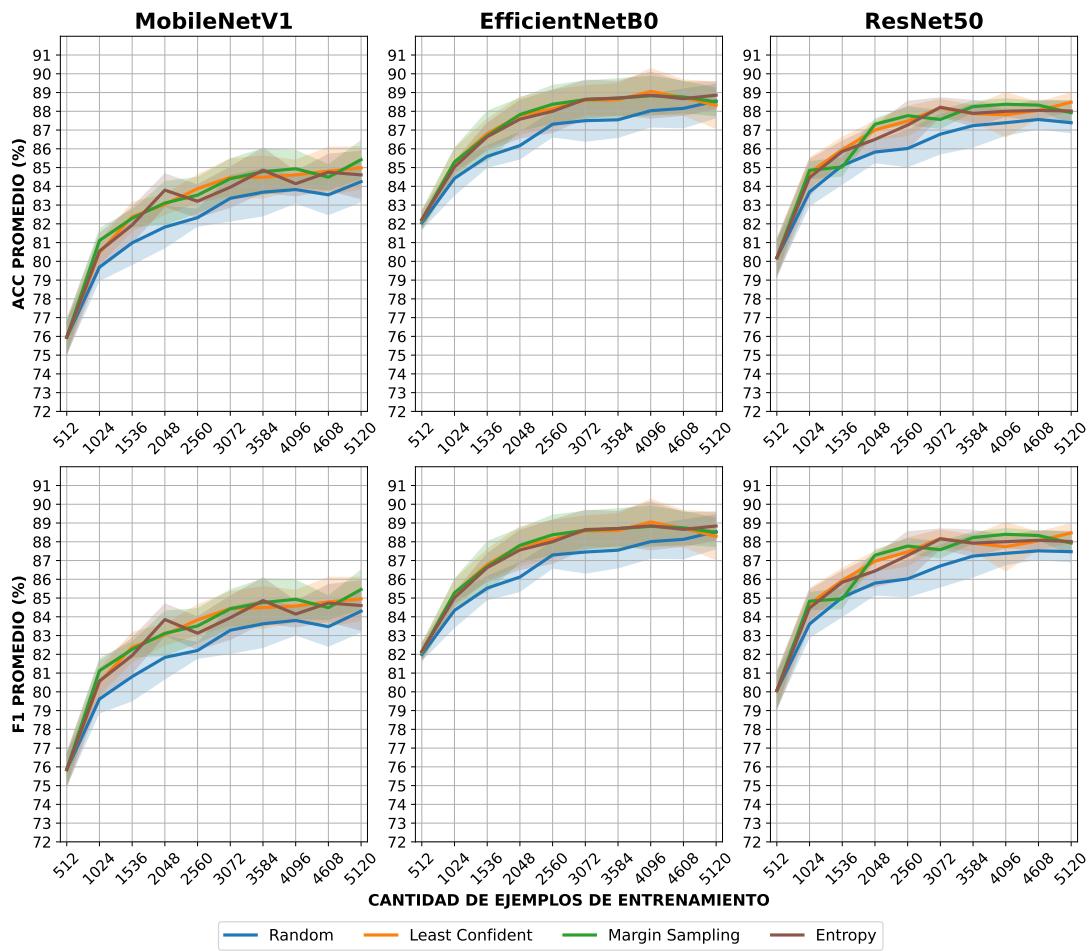
| ACC (%) | Random | Least Confident | Margin      | Entropy     | F1 (%)  | Random | Least Confident | Margin      | Entropy     |
|---------|--------|-----------------|-------------|-------------|---------|--------|-----------------|-------------|-------------|
| 76.59 % | —      | —               | —           | —           | 73.87 % | —      | —               | —           | —           |
| 76.69 % | 512    | 512             | 512         | 512         | 73.97 % | 512    | 512             | 512         | 512         |
| 76.79 % | 1024   | 1024            | 1024        | 1024        | 74.07 % | 1024   | 1024            | 1024        | 1024        |
| 83.49 % | 1536   | 1024            | 1024        | 1536        | 83.47 % | 1024   | 1024            | 1024        | 1536        |
| 83.69 % | 1536   | 1536            | <b>1024</b> | 1536        | 83.57 % | 1536   | 1024            | 1024        | 1536        |
| 84.39 % | 1536   | 1536            | 1536        | 1536        | 83.67 % | 1536   | 1536            | <b>1024</b> | 1536        |
| 84.79 % | 2560   | 1536            | 1536        | 1536        | 84.47 % | 1536   | 1536            | 1536        | 1536        |
| 85.89 % | 2560   | 2048            | 1536        | 1536        | 84.87 % | 2048   | 1536            | 1536        | 1536        |
| 85.99 % | 3072   | 2048            | 1536        | 1536        | 85.07 % | 2560   | 1536            | 1536        | 1536        |
| 86.29 % | 3072   | 2048            | 2048        | <b>1024</b> | 85.97 % | 2560   | 2048            | 1536        | 1536        |
| 86.59 % | 3584   | 2048            | 2048        | <b>1024</b> | 86.07 % | 3072   | 2048            | 1536        | 1536        |
| 86.69 % | 3584   | 2048            | 2048        | 2048        | 86.37 % | 3072   | 2048            | 2048        | <b>1024</b> |
| 86.79 % | 4096   | 2048            | 2048        | 2048        | 86.47 % | 3584   | 2048            | 2048        | <b>1024</b> |
| 86.89 % | 4096   | 2560            | 2048        | 2048        | 86.87 % | 3584   | 2048            | 2048        | 2048        |
| 87.09 % | 4608   | 2560            | 2560        | <b>1024</b> | 86.97 % | 4096   | 2560            | 2048        | 2048        |
| 87.19 % | 4608   | 2560            | 2560        | 2560        | 87.07 % | 4608   | 2560            | 2560        | <b>1024</b> |
| 87.29 % | 4608   | 2560            | 3072        | 2560        | 87.27 % | 4608   | 2560            | 2560        | 2560        |
| 87.49 % | 4608   | 3072            | 3072        | 3072        | 87.37 % | 4608   | 2560            | 3072        | 2560        |
| 87.59 % | —      | <b>3072</b>     | 3584        | 4096        | 87.47 % | 4608   | <b>3072</b>     | 3584        | 3584        |
| 87.79 % | —      | <b>3072</b>     | 3584        | 4608        | 87.57 % | —      | <b>3072</b>     | 3584        | 4096        |
| 87.89 % | —      | 3584            | 3584        | 4608        | 87.77 % | —      | 3584            | 3584        | 4608        |
| 87.99 % | —      | 3584            | 3584        | 5120        | 87.97 % | —      | 3584            | 3584        | —           |
| 88.09 % | —      | <b>3584</b>     | —           | —           | 88.07 % | —      | <b>4608</b>     | —           | —           |
| 88.19 % | —      | <b>4608</b>     | —           | —           | 88.26 % | —      | —               | —           | —           |
| 88.28 % | —      | —               | —           | —           | —       | —      | —               | —           | —           |

**En negrita**, las cantidades mínimas de datos requeridas cuando son menores que en las demás estrategias. El símbolo “—” indica que no se alcanzó el desempeño especificado en la fila.

La Tabla 4.7 muestra que las estrategias de consulta de aprendizaje activo mantienen un desempeño similar para valores de ACC y F1 inferiores o cercanos al 87 %, con algunos resultados destacados de *Entropy*, que alcanza un 86.59 % con solo 1536 instancias etiquetadas. Para valores superiores o cercanos al 87.5 %, *Least Confident* demuestra un mejor desempeño, requiriendo igual o menor cantidad de ejemplos que las demás estrategias.

#### 4.2.2. Aprendizaje activo en CIFAR-10

La Figura 4.2 muestra las curvas de aprendizaje en términos del promedio de las iteraciones de validación cruzada junto a la desviación estándar en ACC y F1 alcanzados para una determinada cantidad de datos etiquetados en CIFAR-10. También, la Tabla 4.8 muestra el área que hay bajo cada una de las curvas expuestas.



**Figura 4.2:** Curvas de aprendizaje de los algoritmos de aprendizaje activo en el conjunto de datos CIFAR-10 utilizando MobileNetV1, EfficientNetB0 y ResNet50. Fuente: Elaboración propia.

**Tabla 4.8:** AULC en CIFAR-10. Fuente: Elaboración propia.

| CNN            | AULC       |                 |               |         |           |                 |               |         |
|----------------|------------|-----------------|---------------|---------|-----------|-----------------|---------------|---------|
|                | Curvas ACC |                 |               |         | Curvas F1 |                 |               |         |
|                | Random     | Least Confident | Margin        | Entropy | Random    | Least Confident | Margin        | Entropy |
| MobileNetV1    | 0.7393     | 0.7486          | <b>0.7493</b> | 0.7474  | 0.7387    | 0.7484          | <b>0.7492</b> | 0.7474  |
| EfficientNetB0 | 0.7800     | 0.7882          | <b>0.7884</b> | 0.7876  | 0.7796    | 0.7880          | <b>0.7883</b> | 0.7875  |
| ResNet50       | 0.7734     | 0.7813          | <b>0.7815</b> | 0.7803  | 0.7730    | 0.7811          | <b>0.7813</b> | 0.7802  |

En negrita, los valores de la estrategia con el mayor AULC en cada fila para una determinada métrica de desempeño.

Los algoritmos que consideran aprendizaje activo superan el rendimiento obtenido por el aprendizaje pasivo dada una cantidad de ejemplos etiquetados en la mayoría de los casos. Con MobileNetV1, los algoritmos de aprendizaje activo rondan un desempeño aproximado de 84 % en ACC y F1 con una reducción en ejemplos

etiquetados de 40 %. Para EfficientNetB0, considerando solo un 40 % de ejemplos, es decir, una reducción de 60 %, se alcanza un desempeño aproximado de un 1 % menor que el desempeño máximo para las tres estrategias. Utilizando ResNet50, tanto *least confident* como *entropy* alcanzan un desempeño similar al rendimiento máximo obtenido superior a 88 % de ACC y F1 con un 60 % de ejemplos, *Margin* logra este rendimiento con 70 % de datos. Al igual que como ocurre para Fashion-MNIST, para CIFAR-10 *Margin* presenta mayor área para todas las curvas en comparación a los demás algoritmos.

Las tablas 4.9, 4.10 y 4.11 presentan el desempeño de los algoritmos de aprendizaje activo en términos de la cantidad mínima de ejemplos etiquetados necesarios para alcanzar un determinado valor de ACC y F1, correspondientes a MobileNetV1, EfficientNetB0 y ResNet50, respectivamente.

**Tabla 4.9:** Cantidad mínima de ejemplos requeridos por los diferentes algoritmos de aprendizaje activo para alcanzar las métricas ACC y F1 en el conjunto de imágenes CIFAR-10 usando la CNN MobileNetV1.

| ACC (%) | Random | Least Confident | Margin      | Entropy     | F1 (%)  | Random      | Least Confident | Margin      | Entropy     |
|---------|--------|-----------------|-------------|-------------|---------|-------------|-----------------|-------------|-------------|
| 75.85 % | —      | —               | —           | —           | 75.75 % | —           | —               | —           | —           |
| 75.95 % | 512    | 512             | 512         | 512         | 75.85 % | 512         | 512             | 512         | 512         |
| 76.05 % | 1024   | 1024            | 1024        | 1024        | 75.95 % | 1024        | 1024            | 1024        | 1024        |
| 79.75 % | 1536   | 1024            | 1024        | 1024        | 79.65 % | 1536        | 1024            | 1024        | 1024        |
| 80.55 % | 1536   | 1536            | <b>1024</b> | 1536        | 80.55 % | 1536        | 1536            | 1024        | 1024        |
| 81.05 % | 2048   | 1536            | <b>1024</b> | 1536        | 80.65 % | 1536        | 1536            | <b>1024</b> | 1536        |
| 81.15 % | 2048   | 1536            | 1536        | 1536        | 80.85 % | 2048        | 1536            | <b>1024</b> | 1536        |
| 81.85 % | 2560   | 1536            | 1536        | 1536        | 81.15 % | 2048        | 1536            | 1536        | 1536        |
| 81.95 % | 2560   | 1536            | 1536        | 2048        | 81.85 % | 2560        | 1536            | 1536        | 1536        |
| 82.35 % | 3072   | <b>1536</b>     | 2048        | 2048        | 81.95 % | 2560        | 1536            | 1536        | 2048        |
| 82.45 % | 3072   | 2048            | 2048        | 2048        | 82.25 % | 3072        | 1536            | 1536        | 2048        |
| 83.05 % | 3072   | 2560            | 2048        | 2048        | 82.35 % | 3072        | <b>1536</b>     | 2048        | 2048        |
| 83.15 % | 3072   | 2560            | 2560        | <b>2048</b> | 82.45 % | 3072        | 2048            | 2048        | 2048        |
| 83.45 % | 3584   | 2560            | 2560        | <b>2048</b> | 83.05 % | 3072        | 2560            | 2048        | 2048        |
| 83.55 % | 3584   | 2560            | 3072        | <b>2048</b> | 83.15 % | 3072        | 2560            | 2560        | <b>2048</b> |
| 83.75 % | 4096   | 2560            | 3072        | <b>2048</b> | 83.35 % | 3584        | 2560            | 2560        | <b>2048</b> |
| 83.85 % | 5120   | <b>2560</b>     | 3072        | 3072        | 83.55 % | 3584        | 2560            | 3072        | <b>2048</b> |
| 83.95 % | 5120   | 3072            | 3072        | 3072        | 83.65 % | 4096        | 2560            | 3072        | <b>2048</b> |
| 84.05 % | 5120   | 3072            | 3584        | 83.85 %     | 5120    | 3072        | 3072            | 3072        | <b>2048</b> |
| 84.25 % | —      | 3072            | 3584        | 83.95 %     | 5120    | 3072        | 3072            | 3584        | —           |
| 84.45 % | —      | <b>3072</b>     | 3584        | 84.35 %     | —       | <b>3072</b> | <b>3072</b>     | 3584        | —           |
| 84.55 % | —      | 4096            | 3584        | 84.45 %     | —       | 3584        | 3584            | 3584        | —           |
| 84.65 % | —      | 4608            | 3584        | 84.55 %     | —       | 4096        | 3584            | 3584        | —           |
| 84.85 % | —      | 5120            | 4096        | <b>3584</b> | 84.65 % | —           | 4608            | 3584        | 3584        |
| 84.95 % | —      | 5120            | 5120        | —           | 84.85 % | —           | 5120            | 4096        | <b>3584</b> |
| 85.05 % | —      | —               | 5120        | —           | 84.95 % | —           | 5120            | 5120        | —           |
| 85.42 % | —      | —               | <b>5120</b> | —           | 85.05 % | —           | —               | 5120        | —           |
|         |        |                 |             |             | 85.46 % | —           | —               | <b>5120</b> | —           |

**En negrita**, las cantidades mínimas de datos requeridas cuando son menores que en las demás estrategias. El símbolo “—” indica que no se alcanzó el desempeño especificado en la fila.

Para desempeños inferiores a 83 % de ACC y F1, *Least Confident* y *Margin* en general necesitan menos ejemplos que las demás estrategias, donde se destaca

que *Margin* alcanza un 81.05 % en ACC y 80.85 % en F1 con solo 1024 ejemplos etiquetados. Desde desempeños superiores a 83 % en ambas métricas, *Entropy* presenta mejores resultados, aproximándose al 84 % de ACC y F1 con 2048 ejemplos. Por encima de 84 % *Least Confident* logra 84.45 % de ACC con 3072 datos y 84.35 % de F1 también con 3072 ejemplos, con *Margin* manteniéndose competitivo.

**Tabla 4.10:** Cantidad mínima de ejemplos requeridos por los diferentes algoritmos de aprendizaje activo para alcanzar las métricas ACC y F1 en el conjunto de imágenes CIFAR-10 usando la CNN EfficientNetB0.

| ACC (%) | Random | Least       | Confident   | Margin      | Entropy | F1 (%)  | Random      | Least       | Confident   | Margin      | Entropy     |
|---------|--------|-------------|-------------|-------------|---------|---------|-------------|-------------|-------------|-------------|-------------|
| 81.95 % | —      | —           | —           | —           | —       | 81.89 % | —           | —           | —           | —           | —           |
| 82.05 % | 512    | 512         | 512         | 512         | 512     | 81.99 % | 512         | 512         | 512         | 512         | 512         |
| 82.15 % | 1024   | 512         | 512         | 512         | 82.09 % | 1024    | 512         | 512         | 512         | 512         | 512         |
| 82.25 % | 1024   | 1024        | 1024        | 1024        | 82.19 % | 1024    | 1024        | 1024        | 1024        | 1024        | 1024        |
| 84.45 % | 1536   | 1024        | 1024        | 1024        | 84.39 % | 1536    | 1024        | 1024        | 1024        | 1024        | 1024        |
| 85.05 % | 1536   | <b>1024</b> | <b>1024</b> | 1536        | 85.09 % | 1536    | <b>1024</b> | <b>1024</b> | <b>1024</b> | <b>1024</b> | 1536        |
| 85.35 % | 1536   | 1536        | 1536        | 1536        | 85.29 % | 1536    | 1536        | 1536        | 1536        | 1536        | 1536        |
| 85.65 % | 2048   | 1536        | 1536        | 1536        | 85.59 % | 2048    | 1536        | 1536        | 1536        | 1536        | 1536        |
| 86.25 % | 2560   | 1536        | 1536        | 1536        | 86.19 % | 2560    | 1536        | 1536        | 1536        | 1536        | 1536        |
| 86.65 % | 2560   | 1536        | 1536        | 2048        | 86.69 % | 2560    | <b>1536</b> | 2048        | 2048        | 2048        | 2048        |
| 86.75 % | 2560   | <b>1536</b> | 2048        | 2048        | 86.89 % | 2560    | 2048        | 2048        | 2048        | 2048        | 2048        |
| 86.85 % | 2560   | 2048        | 2048        | 2048        | 87.39 % | 3072    | 2048        | 2048        | 2048        | 2048        | 2048        |
| 87.35 % | 3072   | 2048        | 2048        | 2048        | 87.49 % | 3584    | 2048        | 2048        | 2048        | 2048        | 2048        |
| 87.55 % | 3584   | 2048        | 2048        | 2048        | 87.59 % | 4096    | 2048        | 2048        | 2560        | 2048        | 2560        |
| 87.65 % | 4096   | <b>2048</b> | <b>2048</b> | 2560        | 87.79 % | 4096    | 2560        | <b>2048</b> | <b>2048</b> | <b>2048</b> | 2560        |
| 87.85 % | 4096   | 2560        | 2560        | 2560        | 87.89 % | 4096    | 2560        | 2560        | 2560        | 2560        | 2560        |
| 88.05 % | 4608   | 2560        | 2560        | 3072        | 88.09 % | 4608    | 2560        | 2560        | 3072        | 2560        | 3072        |
| 88.15 % | 4608   | 3072        | <b>2560</b> | 3072        | 88.19 % | 5120    | 3072        | <b>2560</b> | <b>2560</b> | <b>2560</b> | 3072        |
| 88.25 % | 5120   | 3072        | <b>2560</b> | 3072        | 88.39 % | 5120    | 3072        | 3072        | 3072        | 3072        | 3072        |
| 88.45 % | 5120   | 3072        | 3072        | 3072        | 88.59 % | —       | 3584        | <b>3072</b> | <b>3072</b> | <b>3072</b> | <b>3072</b> |
| 88.65 % | —      | 4096        | <b>3584</b> | <b>3584</b> | 88.69 % | —       | 4096        | <b>3584</b> | <b>3584</b> | <b>3584</b> | <b>3584</b> |
| 88.75 % | —      | 4096        | 4096        | 4096        | 88.79 % | —       | 4096        | 4096        | 4096        | 4096        | 4096        |
| 88.85 % | —      | 4096        | —           | 4096        | 88.89 % | —       | <b>4096</b> | —           | —           | —           | —           |
| 88.95 % | —      | <b>4096</b> | —           | —           | 89.06 % | —       | —           | —           | —           | —           | —           |
| 89.06 % | —      | —           | —           | —           | —       | —       | —           | —           | —           | —           | —           |

**En negrita**, las cantidades mínimas de datos requeridas cuando son menores que en las demás estrategias. El símbolo “—” indica que no se alcanzó el desempeño especificado en la fila.

Tanto *Least Confident* como *Margin* presentan resultados similares entre sí y por lo general superiores a las demás estrategias en ACC y F1. Se destaca que ambas estrategias logran un ACC de 87.65 % con 2048 ejemplos y que *Margin* supera el 88.5 % en ACC y F1 con 3584 ejemplos.

**Tabla 4.11:** Cantidad mínima de ejemplos requeridos por los diferentes algoritmos de aprendizaje activo para alcanzar las métricas ACC y F1 en el conjunto de imágenes CIFAR-10 usando la CNN ResNet50.

| ACC (%) | Random | Least       | Confident   | Margin      | Entropy | F1 (%)  | Random      | Least       | Confident   | Margin | Entropy |
|---------|--------|-------------|-------------|-------------|---------|---------|-------------|-------------|-------------|--------|---------|
| 80.09 % | —      | —           | —           | —           | —       | 79.96 % | —           | —           | —           | —      | —       |
| 80.19 % | 512    | 512         | 512         | 512         | 512     | 80.06 % | 512         | 512         | 512         | 512    | 512     |
| 80.29 % | 1024   | 1024        | 1024        | 1024        | 1024    | 80.16 % | 1024        | 1024        | 1024        | 1024   | 1024    |
| 83.79 % | 1536   | 1024        | 1024        | 1024        | 83.66 % | 1536    | 1024        | 1024        | 1024        | 1024   | 1024    |
| 84.49 % | 1536   | 1024        | 1024        | 1536        | 84.56 % | 1536    | 1024        | 1024        | 1024        | 1536   | 1536    |
| 84.79 % | 1536   | 1536        | <b>1024</b> | 1536        | 84.76 % | 1536    | 1536        | 1536        | <b>1024</b> | 1536   | 1536    |
| 84.89 % | 1536   | 1536        | 1536        | 1536        | 84.86 % | 1536    | 1536        | 1536        | 1536        | 1536   | 1536    |
| 85.09 % | 1536   | 1536        | 2048        | 1536        | 84.96 % | 1536    | 1536        | 1536        | 2048        | 1536   | 1536    |
| 85.19 % | 2048   | 1536        | 2048        | 1536        | 85.06 % | 2048    | 1536        | 2048        | 2048        | 1536   | 1536    |
| 85.89 % | 2560   | <b>1536</b> | 2048        | 2048        | 85.86 % | 2560    | <b>1536</b> | 2048        | 2048        | 2048   | 2048    |
| 85.99 % | 2560   | 2048        | 2048        | 2048        | 85.96 % | 2560    | 2048        | 2048        | 2048        | 2048   | 2048    |
| 86.09 % | 3072   | 2048        | 2048        | 2048        | 86.06 % | 3072    | 2048        | 2048        | 2048        | 2048   | 2048    |
| 86.59 % | 3072   | 2048        | 2048        | 2560        | 86.46 % | 3072    | 2048        | 2048        | 2560        | 2560   | 2560    |
| 86.79 % | 3584   | 2048        | 2048        | 2560        | 86.76 % | 3584    | 2048        | 2048        | 2048        | 2560   | 2560    |
| 87.09 % | 3584   | 2560        | <b>2048</b> | 2560        | 86.96 % | 3584    | 2560        | <b>2048</b> | 2560        | 2560   | 2560    |
| 87.29 % | 4096   | 2560        | <b>2048</b> | 3072        | 87.26 % | 4096    | 2560        | <b>2048</b> | 2560        | 2560   | 2560    |
| 87.39 % | 4096   | 2560        | 2560        | 3072        | 87.36 % | 4096    | 2560        | 2560        | 2560        | 3072   | 3072    |
| 87.49 % | 4608   | 3072        | <b>2560</b> | 3072        | 87.46 % | 4608    | 3072        | <b>2560</b> | 3072        | 3072   | 3072    |
| 87.59 % | —      | 3072        | <b>2560</b> | 3072        | 87.56 % | —       | 3072        | <b>2560</b> | 3072        | 3072   | 3072    |
| 87.79 % | —      | 3072        | 3584        | 3072        | 87.86 % | —       | 3072        | 3584        | 3584        | 3072   | 3072    |
| 88.19 % | —      | 5120        | 3584        | <b>3072</b> | 88.16 % | —       | <b>3072</b> | 3584        | 3584        | —      | —       |
| 88.29 % | —      | 5120        | <b>4096</b> | —           | 88.26 % | —       | 5120        | <b>4096</b> | 4096        | —      | —       |
| 88.39 % | —      | 5120        | —           | —           | 88.46 % | —       | 5120        | —           | —           | —      | —       |
| 88.48 % | —      | <b>5120</b> | —           | —           | 88.47 % | —       | <b>5120</b> | —           | —           | —      | —       |

**En negrita**, las cantidades mínimas de datos requeridas cuando son menores que en las demás estrategias. El símbolo “—” indica que no se alcanzó el desempeño especificado en la fila.

Al igual que con EfficientNetB0, con ResNet50, *Least Confident* y *Margin* se muestran superiores a las otras estrategias de consulta. Por ejemplo, *Margin* supera el 87.5 % en ambas métricas con 2560 ejemplos, equivalentes a un 50 % del total de datos de entrenamiento, con una reducción aproximada de 1 % en ACC y F1.

#### 4.2.3. Selección de estrategia de consulta

Los resultados muestran que los algoritmos de aprendizaje activo requieren menos ejemplos etiquetados para alcanzar un nivel de desempeño específico en las métricas de ACC y F1, en comparación con el aprendizaje pasivo. Esta ventaja se mantiene consistentemente, excepto en casos puntuales al utilizar el 20 % o el 100 % de los datos de entrenamiento, donde el aprendizaje pasivo obtiene un rendimiento ligeramente superior al de algunas estrategias de aprendizaje activo. En términos de AULC, la estrategia *Margin* supera a las demás en todas las configuraciones evaluadas.

Respecto a las estrategias de aprendizaje activo, no se observan diferencias

significativas en su desempeño general: los valores de las métricas analizadas son notablemente similares entre ellas. No obstante, *Margin* emerge como la opción con mejor rendimiento comparativo. Además, la estrategia *Entropy* presenta una desventaja práctica relevante: su implementación en dispositivos *FPGA* utiliza demasiados recursos [65, 66, 67].

Considerando la superioridad marginal de *Margin* en AULC, la homogeneidad de resultados entre estrategias y las limitaciones técnicas de *Entropy*, se selecciona *Margin* como la estrategia de consulta para su aceleración mediante *hardware*.

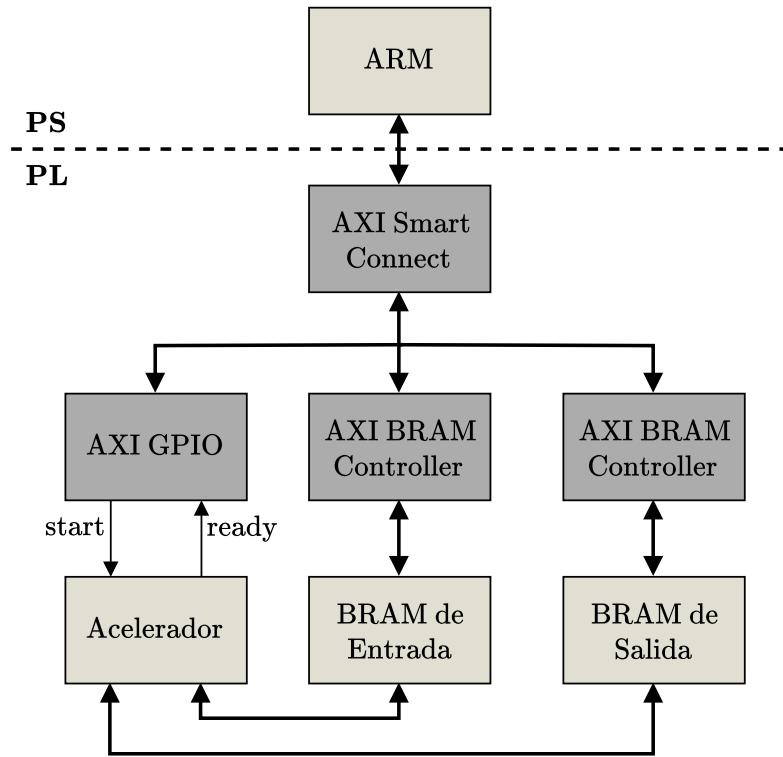
## Capítulo 5

# Arquitectura Digital en SoC

En este capítulo se presenta una descripción detallada de la arquitectura *hardware* diseñada para implementar el algoritmo *Margin Sampling* en un SoC. En primer lugar, esta sección expone la arquitectura general del sistema, que integra un sistema de procesamiento (PS) basado en un procesador ARM y una lógica programable (PL), donde se gestionan las memorias BRAM y los módulos AXI para la comunicación y el flujo de datos. Luego, se aborda el algoritmo *Margin Sampling*, explicando su flujo de operaciones desde la identificación de las dos mayores probabilidades en las predicciones del clasificador hasta la selección de instancias con márgenes menores. Después, se describe en detalle el diseño de la arquitectura digital del acelerador, analizando potenciales problemas e incluyendo las modificaciones implementadas para superar limitaciones de *timing*, logrando una frecuencia de operación de 110 MHz.

### 5.1. Arquitectura general

La arquitectura es híbrida, compuesta por un sistema de procesamiento (PS, *Processing System*) y lógica programable (PL, *Programmable Logic*), como es mostrado en la Figura 5.1. En esta estructura, PS envía los datos que serán procesados desde el procesador *ARM* hacia PL, los cuales son almacenados en la BRAM de entrada. Cuando *Acelerador* recibe la señal *start* desde PS, se da inicio al procesamiento. Una vez los datos procesados son escritos en la BRAM de salida. se activa una señal *ready* que notifica a PS que el procesamiento ha finalizado y los resultados están disponibles para su lectura.



**Figura 5.1:** Arquitectura general del sistema. Fuente: Adaptado de <https://hackmd.io/@ween168/rkZnpSxfkl>. Fecha de último acceso: 7 de Marzo de 2025 a las 08:07 hrs.

Las BRAM se generan utilizando el IP *Block Memory Generator* de Vivado y son instanciadas directamente en el RTL. Estas memorias ocupan un espacio dedicado en el FPGA, evitando el uso de recursos como flip-flops o LUTs. Ambas son configuradas como *True Dual-Port*, es decir, disponen de dos puertos independientes que acceden a una memoria compartida, permitiendo operaciones de lectura y escritura por cada uno. Se trabaja con una precisión de 16 bits; dado que cada conjunto de imágenes contiene 10 clases, se requieren 160 bits para representar cada predicción. Por ello, se selecciona un ancho de memoria de 256 bits y se ajusta el rango de direcciones de modo que la profundidad sea 8192, capaz de almacenar 5120 datos en la BRAM de entrada. En cuanto a la BRAM de salida, su ancho es de 32 bits (siendo cada índice de 13 bits) y se configura el rango de direcciones para que su profundidad sea 512, correspondientes al tamaño del *batch*. En este diseño, tanto las BRAM como los GPIO (*General Purpose Input/Output*) (*start* y *ready*) se tratan como MMIO (*Memory-Mapped Input/Output*). Esto significa que se asignan a rangos específicos del espacio de

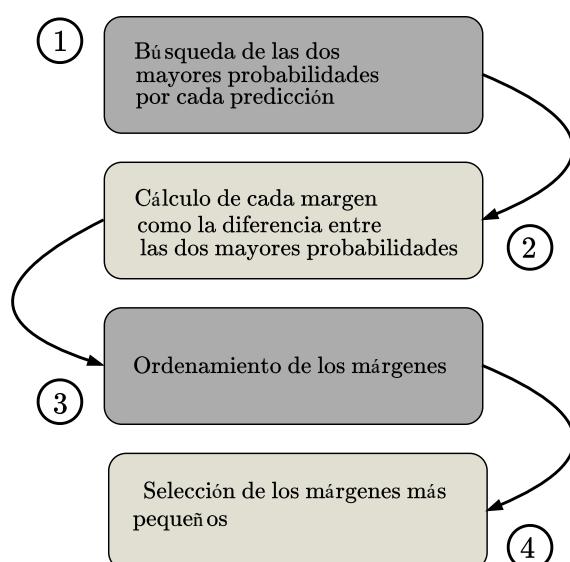
direcciones del procesador, de modo que se pueden leer y escribir datos en ellas usando operaciones de memoria estándar.

Los módulos *AXI BRAM Controller* y *AXI GPIO* se encargan de gestionar el flujo de datos de las BRAM y los GPIO respectivamente, actuando como *master* o *slave* según corresponda. El bloque *AXI Smartconnect* realiza el interconexiónado entre los demás bloques *AXI* y PS, facilitando el traspaso de datos entre PS y PL.

### 5.1.1. Descripción de Margin Sampling

Como se muestra en la ecuación 4.4, el algoritmo *Margin Sampling* consiste en seleccionar aquellas instancias cuyo margen es menor. En este contexto, el margen se define como la diferencia entre las dos mayores probabilidades asignadas por el clasificador.

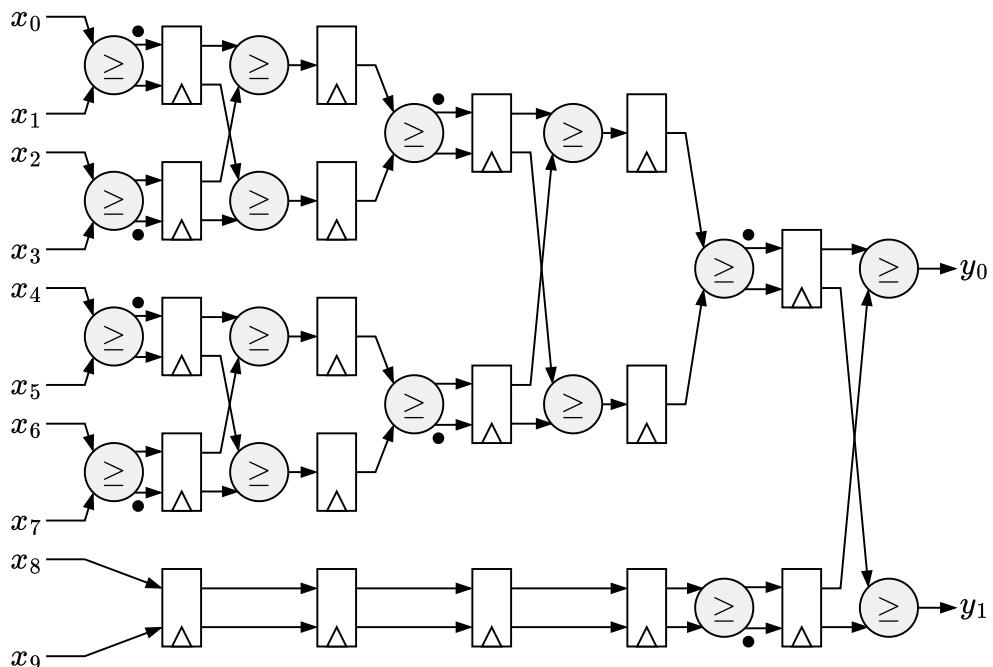
Dadas las predicciones de un clasificador de imágenes, el algoritmo se descompone en los siguientes pasos: en primer lugar, se identifican las dos probabilidades más altas de cada predicción; posteriormente, se calcula el margen restando la segunda mayor probabilidad de la mayor, lo que genera un vector unidimensional con todos los márgenes. Finalmente, se ordenan estos márgenes para seleccionar las instancias con los valores más bajos. La Figura 5.2 ilustra el flujo de operaciones del algoritmo *Margin Sampling*.



**Figura 5.2:** Serie de pasos de *Margin Sampling*. Fuente: Elaboración propia.

## 5.2. Diseño de la arquitectura digital

De acuerdo a la sección 5.1.1, dadas las predicciones de un modelo de clasificación, el primer paso en *Margin Sampling* es encontrar las dos mayores probabilidades de cada predicción. Aprovechando *paralelización* y *pipeline*, las probabilidades más grandes asignadas por el clasificador son encontradas mediante un *árbol de comparaciones binario*, como el mostrado en la Figura 5.3.



**Figura 5.3:** Árbol de comparaciones binario para encontrar las dos mayores probabilidades. El punto que acompaña a los comparadores indica la salida del mayor. Fuente: Elaboración propia.

Este módulo reduce a la mitad el número de candidatos cada dos etapas de *pipeline*. Para ilustrarlo, si se considera un caso de cuatro entradas: en la primera etapa se comparan en pares, y en la segunda los menores con los mayores obtenidos en la etapa anterior, resultando en las dos probabilidades mayores. Así, los posibles candidatos son separados en grupos de cuatro y en el caso de haber entradas restantes, estas deben pasar por registros hasta que puedan ser comparadas. Para el cálculo de la diferencia (margen), solo es necesario un *restador*.

Identificar los márgenes más pequeños en *software*, se puede lograr mediante

aplicar la función *argsort*<sup>1</sup> al vector que contiene todos los márgenes. Esta función devuelve un vector de índices ordenados de acuerdo con los valores de los márgenes. Posteriormente, se seleccionan los primeros índices, siendo la cantidad de índices extraídos igual al tamaño del *batch*. Sin embargo implementar esta estrategia en *hardware* implicaría un elevado costo en recursos. Por ejemplo, si se considera un conjunto de datos no etiquetados compuesto por 5120 datos (caso crítico en estudio), se necesitaría disponer de memoria suficiente en el FPGA para almacenar 5120 márgenes.

Ante esta limitación, se plantea almacenar únicamente una cantidad de márgenes igual al tamaño del *batch*. Una vez que la unidad de memoria se ha llenado, cada nuevo margen se almacenará solo si es menor que el mayor de los márgenes ya guardados. De este modo, al finalizar el procesamiento de todos los márgenes, se contará con el conjunto requerido sin necesidad de realizar un ordenamiento completo. El Algoritmo 2, describe el comportamiento de la estrategia de selección, en este se asume que la unidad de memoria ya ha sido llenada con los primeros datos.

---

**Algoritmo 2** Selección de índices con una unidad de memoria.

---

```
1: Input: Secuencia de datos  $d_k, d_{k+1}, \dots, d_{N_{data}-1}$ .
2: Output: Conjunto  $S$  con los  $k$  márgenes más pequeños.
3:
4: Sea:
5:  $B$ : Unidad de memoria de tamaño  $k$ .
6:  $B^{max}$ : Mayor margen almacenado.
7:
8: Selección:
9: for  $k \leq i \leq N_{data} - 1$  do
10:   if  $d_i < B^{max}$  then
11:      $B \leftarrow (B \setminus B^{max}) \cup d_i$ 
12:     Actualizar  $B^{max}$ 
13:   end if
14: end for
15:  $S \leftarrow B$ 
16:
17: return  $S$ 
```

---

Si bien lo anterior parece una solución idónea, esta estrategia requiere disponer

---

<sup>1</sup><https://numpy.org/doc/stable/reference/generated/numpy.argsort.html>. Fecha de último acceso: 3 de Marzo de 2025 a las 23:03 hrs.

del valor máximo cada vez que llega un nuevo margen. Debido a que los márgenes se calculan a través de un *pipeline*, luego del retardo inicial se obtiene un nuevo margen en cada ciclo de reloj, lo que implica la necesidad de contar con el máximo margen en cada ciclo, algo que resulta inviable.

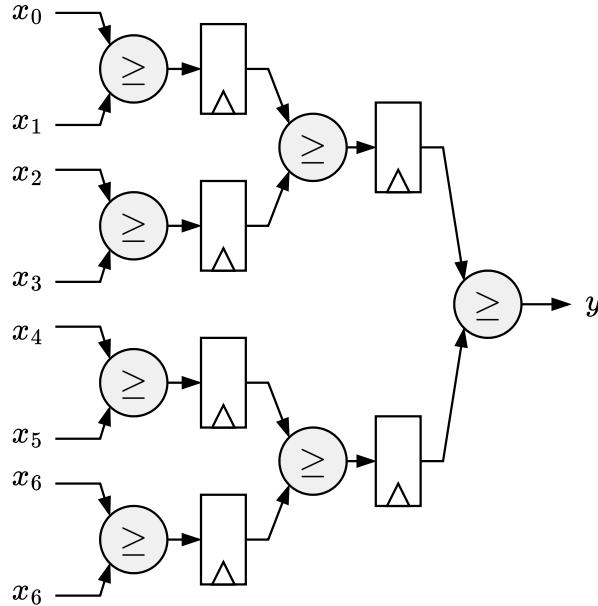
Para superar esta limitación, se propone utilizar múltiples unidades de memoria de igual tamaño, cuya capacidad combinada sea equivalente al tamaño del *batch*. De esta forma, cuando llega un margen, se compara con el máximo almacenado en una de las unidades; el siguiente margen se compara con el máximo de otra unidad, y así sucesivamente. Este funcionamiento se presenta en el Algoritmo 3. Como muestra la Figura 5.4, utilizando un *árbol de comparaciones binario* para determinar el máximo en cada unidad, se requieren  $\log_2(N)$  ciclos de reloj, donde  $N$  es el tamaño de la unidad de memoria. Por lo tanto, para un tamaño de unidad dado, es necesario disponer de al menos  $\log_2(N)$  unidades, lo que otorga el tiempo suficiente para determinar los máximos correspondientes antes de que deban compararse con un margen entrante.

**Algoritmo 3** Selección de índices con múltiples unidades de memoria.

---

```
1: Input: Secuencia de datos  $d_k, d_{k+1}, \dots, d_{N_{data}-1}$ .
2: Output: Conjunto  $S$  con los  $k$  márgenes más pequeños.
3:
4: Sea:
5:  $m$ : Número de unidades de memoria
6:  $B$ : Unidades de memoria de tamaño  $k/m$ .
7:  $B^{max}$ : Mayor dato almacenado en la unidad respectiva.
8:
9: Selección:
10:  $j \leftarrow 0$ 
11: for  $k \leq i \leq N_{data} - 1$  do
12:   if  $d_i < B_j^{max}$  then
13:      $B_j \leftarrow (B_j \setminus B_j^{max}) \cup d_i$ 
14:     Actualizar  $B_j^{max}$ 
15:   end if
16:   if  $j < m - 1$  then
17:      $j \leftarrow j + 1$ 
18:   else
19:      $j \leftarrow 0$ 
20:   end if
21: end for
22:  $S \leftarrow \bigcup_{j=0}^{m-1} B_j$ 
23:
24: return  $S$ 
```

---



**Figura 5.4:** Árbol de comparaciones binario para encontrar el valor máximo. Como  $N = 8$ , el número de ciclos que tarda en encontrar el máximo es  $\log_2 8 = 3$ . Fuente: Elaboración propia.

Por ejemplo, con un tamaño de *batch* igual a 16, lo que implica disponer de 4 unidades de memoria, cada una con capacidad para 4 datos (márgenes). Una vez llenadas las unidades con los primeros 16 datos, se obtienen los máximos respectivos: 0.4, 0.45, 0.3 y 0.2. Cuando llega el dato número 17, con un margen de 0.7, este se descarta porque 0.7 no es menor que el máximo actual (0.4) en la primera unidad, por lo que no se realiza ninguna actualización. En cambio, el dato 18 presenta un margen de 0.32, el cual es menor que el máximo de 0.45 en la segunda unidad. En consecuencia, el margen de 0.32 reemplaza el valor anterior, y se procede a re-calcular el máximo en esa unidad. Dado que cada unidad contiene 4 datos, calcular el máximo mediante un *árbol de comparaciones binario* requiere  $\log_2 4 = 2$  ciclos de reloj. Esto garantiza que, al momento de evaluar el siguiente margen en la misma unidad, el nuevo máximo ya esté disponible.

Aunque este enfoque mantiene el procesamiento en línea de los datos de entrada, el uso de múltiples unidades de memoria puede inducir discrepancias en la selección de instancias, ya que no siempre se garantiza que se escojan estrictamente aquellas con los márgenes más pequeños. Para evaluar el impacto de este posible error, se realizaron pruebas en *software* que emulan este comportamiento. A continuación, se detalla la metodología empleada y se analizan los resultados obtenidos.

### 5.2.1. Emulación en *software*

Las pruebas consisten en medir cuantas discrepancias hay en la selección de instancias entre una implementación base en *software* y una emulación del *hardware* en *software*. Además de evaluar el impacto que estas discrepancias podrían causar.

Para evaluar las discrepancias en la selección de instancias entre una implementación base en *software* y su emulación de *hardware*, se diseñó el siguiente experimento: se generan datos aleatorios que son permutados en cada iteración. Se realizan 100 iteraciones por experimento, y el experimento se repite 10 veces (1000 ejecuciones en total). En cada iteración se identifican los índices no coincidentes y se calcula su valor mínimo, máximo y promedio. Finalmente, se promedian estos valores a lo largo de todas las iteraciones. La Tabla 5.2 presenta los resultados obtenidos para tres configuraciones distintas de datos de entrada: 2560, 5120 y 10240 elementos. En cada caso, se utiliza un *batch* que equivale al 10 % del total de datos.

**Tabla 5.1:** Mínimo, máximo y promedio de índices no coincidentes para distinta cantidad de datos de entrada y distintas configuraciones de tamaño y cantidad de unidades de memoria. Fuente: Elaboración propia.

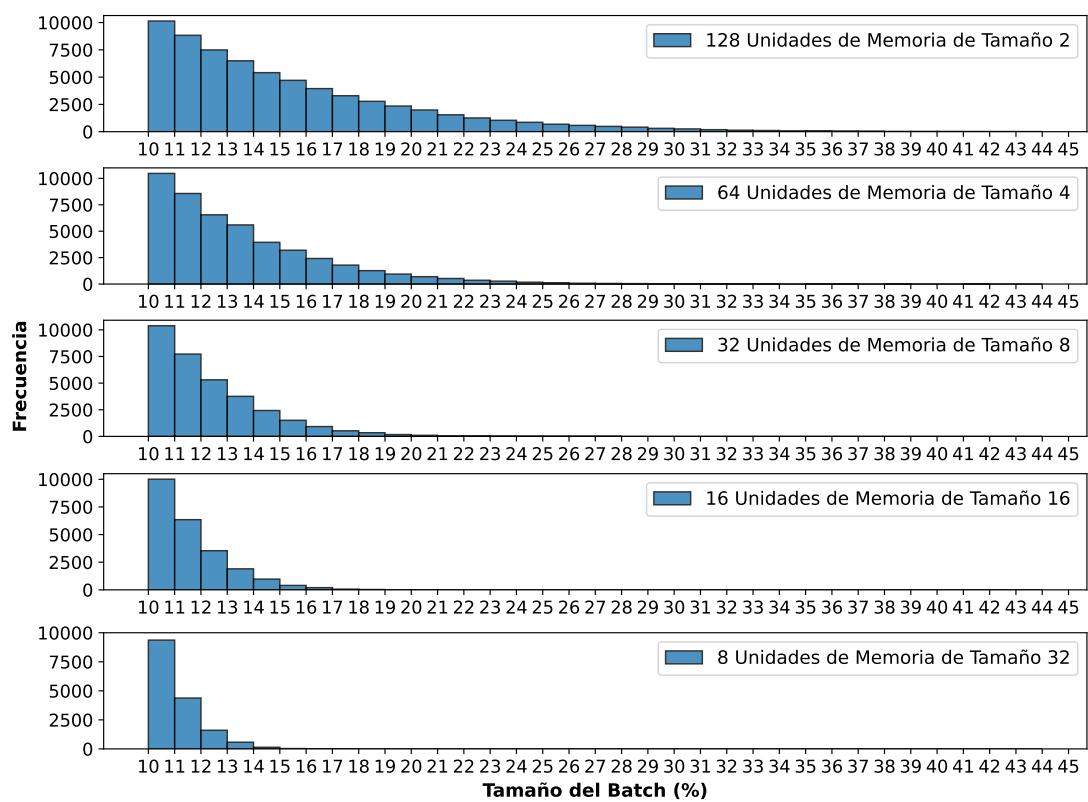
| Tamaño                         | Cantidad | Estadística |       |         |
|--------------------------------|----------|-------------|-------|---------|
|                                |          | Min         | Max   | Prom    |
| <b>Datos de entrada: 2560</b>  |          |             |       |         |
| 2                              | 128      | 54.7        | 78.1  | 65.196  |
| 4                              | 64       | 35.0        | 58.7  | 46.640  |
| 8                              | 32       | 22.8        | 45.1  | 33.119  |
| 16                             | 16       | 12.8        | 36.6  | 23.195  |
| 32                             | 8        | 5.9         | 28.3  | 15.738  |
| <b>Datos de entrada: 5120</b>  |          |             |       |         |
| 2                              | 256      | 112.5       | 147.0 | 130.525 |
| 4                              | 128      | 76.6        | 113.5 | 94.153  |
| 8                              | 64       | 50.9        | 83.8  | 66.816  |
| 16                             | 32       | 32.3        | 64.4  | 47.168  |
| 32                             | 16       | 17.7        | 50.1  | 33.133  |
| 64                             | 8        | 8.5         | 39.2  | 22.275  |
| <b>Datos de entrada: 10240</b> |          |             |       |         |
| 2                              | 512      | 238.5       | 286.0 | 261.019 |
| 4                              | 256      | 166.0       | 212.1 | 188.359 |
| 8                              | 128      | 112.0       | 157.9 | 134.444 |
| 16                             | 64       | 71.1        | 119.3 | 95.429  |
| 32                             | 32       | 45.1        | 93.2  | 67.124  |
| 64                             | 16       | 26.9        | 68.8  | 46.048  |
| 128                            | 8        | 13.7        | 62.7  | 31.615  |

**Tabla 5.2:** Estadísticas de desajuste por configuración de memoria

Los datos de la Tabla 5.2 muestran que, mientras mayor sea la cantidad de datos de entrada, mayor será el número de índices no coincidentes. También, es posible notar que, para un determinado número de datos de entrada, la cantidad de índices no coincidentes disminuye a medida que se reduce la cantidad de unidades de memoria y se aumenta el tamaño de estas.

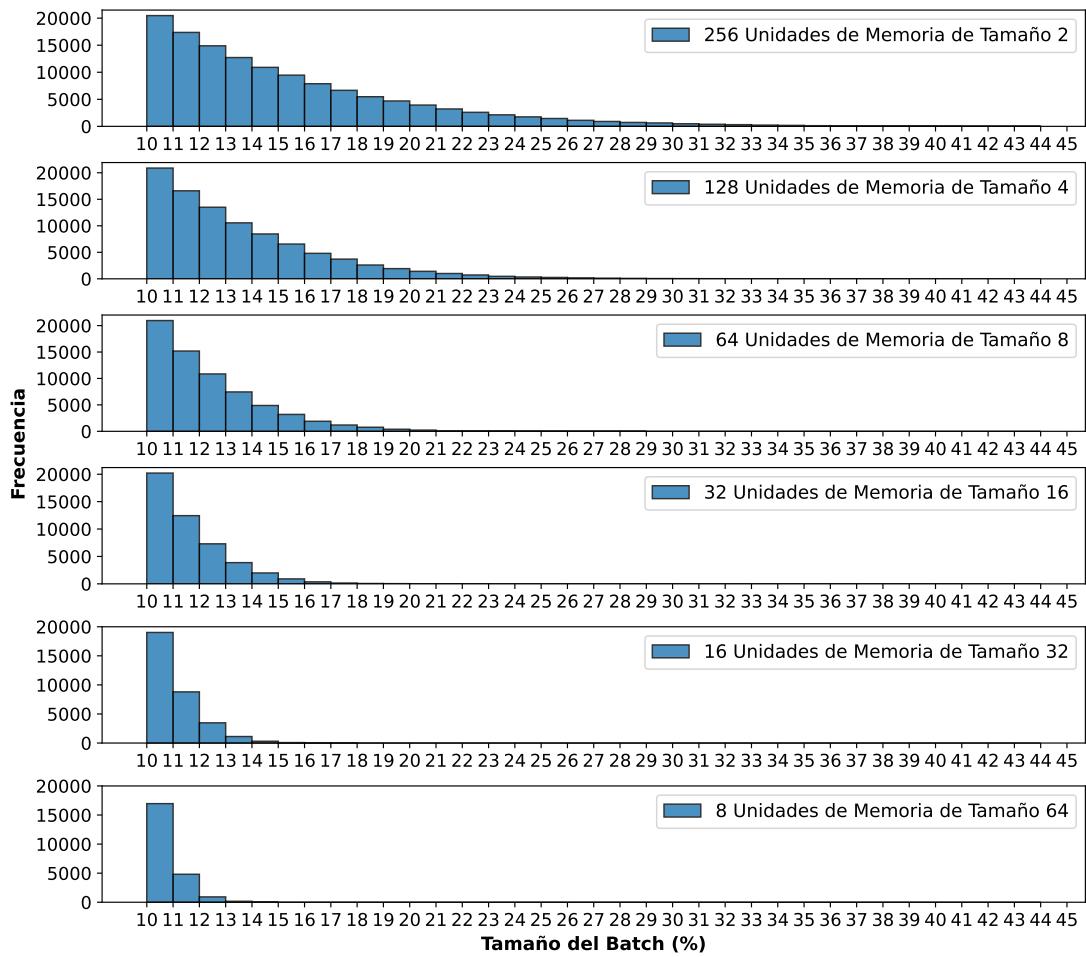
En la segunda prueba se busca evaluar el impacto de los índices no coincidentes producidos por la emulación. Concretamente, se analiza el índice que la implementación base asigna a aquellos elementos que la emulación selecciona dentro del *batch* (10 % del total de datos) pero que no son elegidos en el *batch*

de la implementación base. Se consideran todas las combinaciones posibles de número de unidades de memoria y tamaños de unidades de memoria, siempre que se pueda calcular el máximo de cada unidad antes de proceder a una nueva comparación. Los histogramas de las figuras 5.5, 5.6 y 5.7 muestran gráficamente como se distribuyen estos índices. Al igual que antes, se consideran tres casos con distinta cantidad de datos de entrada: 2560, 5120 y 10240.



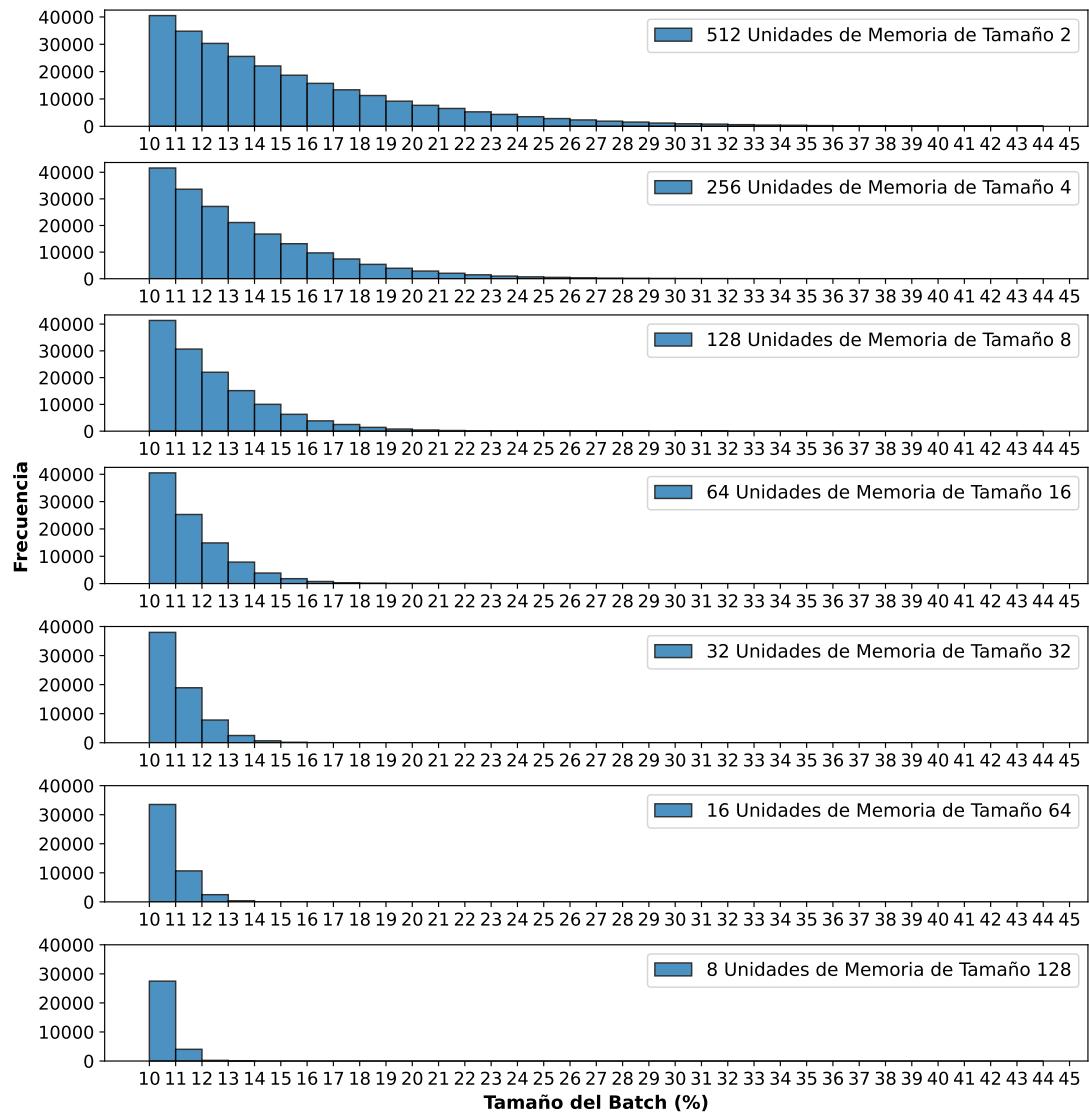
**Figura 5.5:** Índices asignados por la implementación base para los índices no coincidentes, con una cantidad de datos de entrada igual a 2560. Fuente: Elaboración propia.

La Figura 5.5 muestra que los índices no coincidentes se encuentran entre un 10 % y aproximadamente un 30 %, lo que sugiere un error cercano al 20 % cuando se emplean 128 unidades de memoria con una capacidad de 2. En contraste, al utilizar 8 unidades de tamaño 32, el error se reduce al 4 %.



**Figura 5.6:** Índices asignados por la implementación base para los índices no coincidentes, con una cantidad de datos de entrada igual a 5120. Fuente: Elaboración propia.

La Figura 5.6 presenta que cuando se emplean 256 unidades con una capacidad de 2, los índices no coincidentes se sitúan entre un 10 % y un 30 %, lo que implica un error aproximado del 20 % similar al caso con tamaño de *batch* igual a 256. Por otro lado, al reducir la cantidad de unidades a 8, pero aumentando su tamaño a 64, el error se reduce hasta un 3 %. Cabe destacar que, en este caso, existen seis combinaciones posibles de implementación, una posibilidad más respecto al caso anterior.



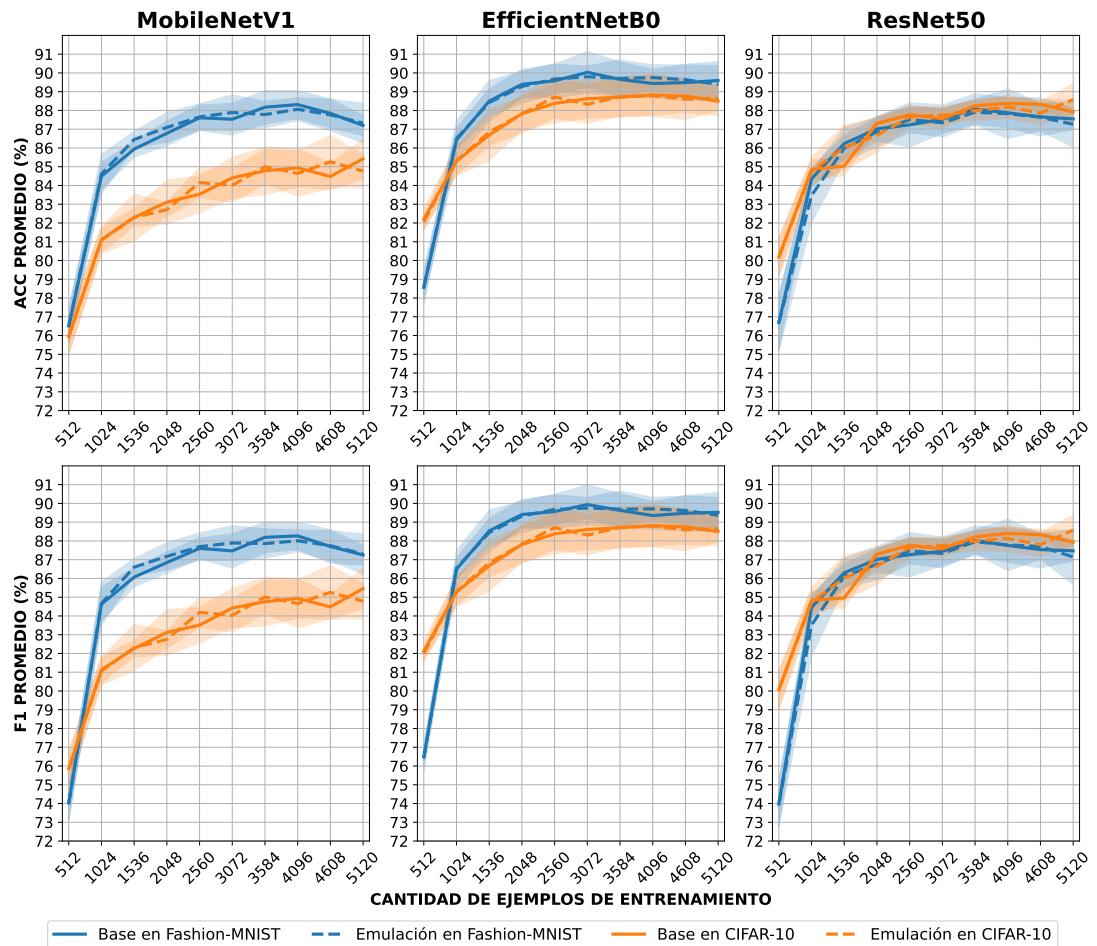
**Figura 5.7:** Índices asignados por la implementación base para los índices no coincidentes, con una cantidad de datos de entrada igual a 10240. Fuente: Elaboración propia.

Al igual que en los casos anteriores, cuando hay 10240 datos de entrada, si las unidades de memoria son de tamaño 2 (512 unidades), el error es cercano al 20 %. En la combinación con 8 unidades de tamaño 128, el error se reduce a casi un 1 %.

Por lo tanto, los resultados indican que, con unidades de memoria con capacidad igual a 2 (caso con mayor número de unidades), la mayoría de los índices no coincidentes se concentran dentro del 30 %, presentando un error de emulación *hardware* de aproximadamente el 20 %. Este error disminuye al emplear menos unidades de memoria de mayor tamaño y, además, se reduce conforme aumenta

la cantidad de datos de entrada. Por ejemplo, para 10240 datos de entrada y 8 unidades de memoria de tamaño 128, el error se reduce a cerca del 1 %.

Finalmente, se comparan las curvas de aprendizaje entre ambas implementaciones, como muestra la Figura 5.8. Dada la cantidad de datos de entrada igual a 5120, se seleccionan 8 unidades de memoria de 64 datos cada una.



**Figura 5.8:** Curvas de aprendizaje de ambas implementaciones, base (*software*) y emulación. Fuente: Elaboración propia.

El punto de mayor discrepancia corresponde a los resultados de ResNet50 en CIFAR-10 con una cantidad de datos igual a 1536 equivalente al 30 % del total. En este punto, para ambas métricas, la emulación supera en aproximadamente un 1 % el rendimiento de la implementación base. En cuanto al resto de curvas y puntos de comparación, las discrepancias no alcanzan un 1 % de diferencia. Además, si se analiza el área bajo la curva, como es mostrado en la Tabla 5.3, se destacan algunas diferencias como 0.0016 en el peor caso (AULC en ACC para ResNet50 en

Fashion-MNIST) y 0.0001 en el mejor caso (EfficientNetB0 en CIFAR-10). Esto indica que las curvas son similares en términos de AULC.

**Tabla 5.3:** AULC de la implementación base y la emulación. Fuente: Elaboración propia.

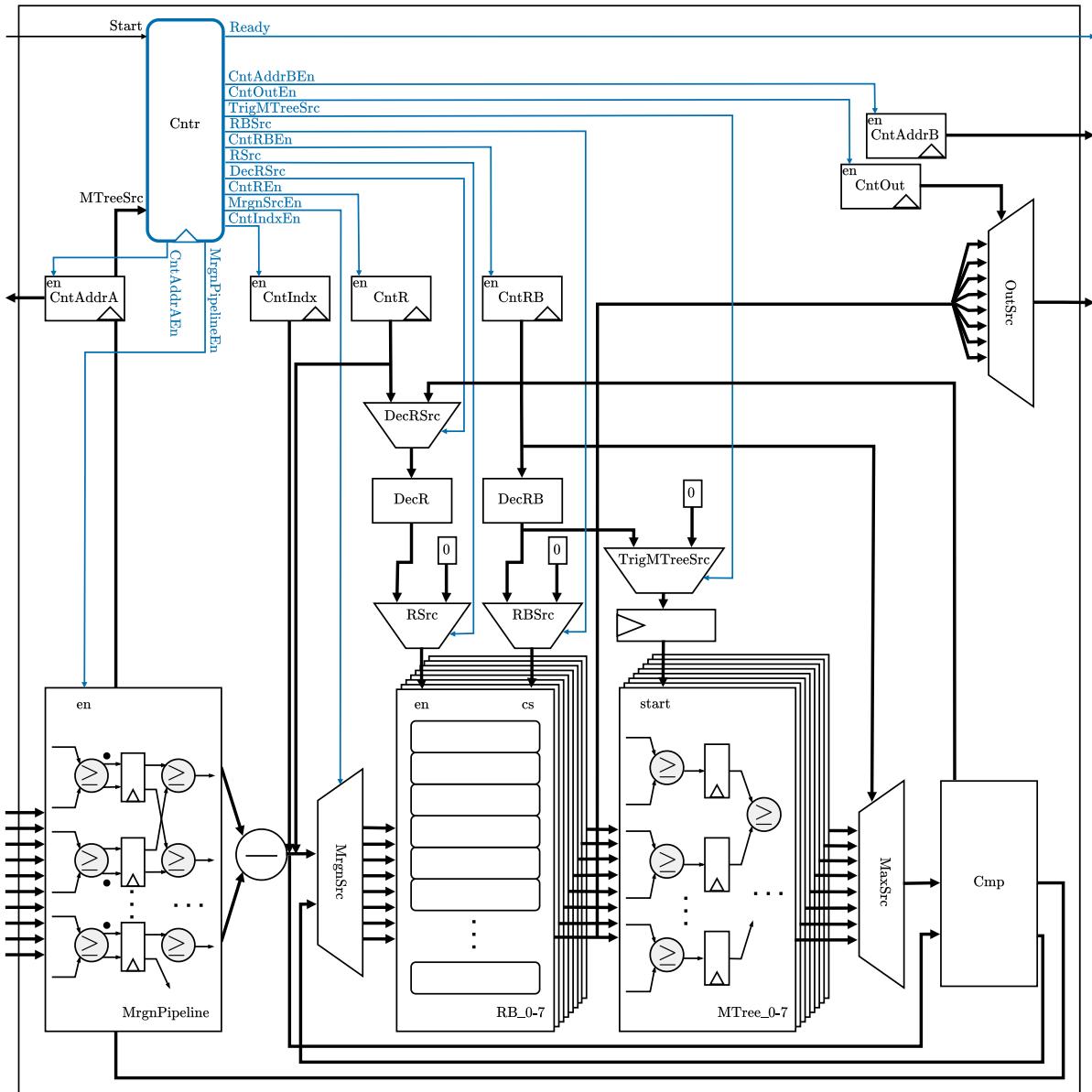
| Configuración                   | AULC          |               |               |               |
|---------------------------------|---------------|---------------|---------------|---------------|
|                                 | Curvas ACC    |               | Curvas F1     |               |
|                                 | Base          | Emulación     | Base          | Emulación     |
| MobileNetV1 en Fashion-MNIST    | 0.7785        | 0.7792        | 0.7774        | 0.7783        |
| MobileNetV1 en CIFAR-10         | 0.7493        | 0.7495        | 0.7492        | 0.7495        |
| EfficientNetB0 en Fashion-MNIST | 0.7965        | 0.7967        | 0.7954        | 0.7956        |
| EfficientNetB0 en CIFAR-10      | <b>0.7884</b> | <b>0.7885</b> | <b>0.7883</b> | <b>0.7884</b> |
| ResNet50 en Fashion-MNIST       | <b>0.7781</b> | <b>0.7765</b> | 0.7765        | 0.7752        |
| ResNet50 en CIFAR-10            | 0.7815        | 0.7812        | 0.7813        | 0.7810        |

En negrita, los valores que presentan la mayor diferencia (0.0016) y la menor diferencia (0.0001) en términos de AULC.

### 5.3. Arquitectura digital

Como se indica en la sección 4.2, la cantidad de datos de entrada es 5120. Según los resultados de la emulación de *hardware* en *software*, se determinó que la configuración que minimiza las discrepancias respecto a la implementación en *software* utiliza 8 memorias, cada una con capacidad para 64 datos.

La arquitectura, mostrada en la Figura 5.9, utiliza como unidades de memoria múltiples *registros* internos, agrupados en *bancos de registros*. Cada banco se conecta directamente a un *árbol de comparaciones binario* que se encarga de calcular el máximo. La selección de los *registros* y de los *bancos de registros* se realiza mediante contadores que generan señales de selección (equivalentes a direcciones dentro de una memoria). Estas señales pasan por un decodificador que realiza una decodificación *One-Hot*, que permite seleccionar un *registro* o *banco de registros* en específico. El dato almacenado es la concatenación del margen (necesario para la comparación), el índice de la instancia y el número de registro en la unidad de memoria, información esencial para reemplazar datos almacenados cuando sea necesario.



**Figura 5.9:** Arquitectura digital del acelerador *hardware* para *Margin Sampling*. Fuente: Elaboración propia.

El procesamiento tiene tres principales etapas marcadas por cambios en el flujo de datos y algunas señales de control. La primera etapa consiste en llenar las unidades de memoria con los primeros 512 datos. Para esto, es activado el contador  $CntAddrA$ , este se encarga de generar las direcciones de lectura desde la BRAM de entrada. Tras el retardo inicial del *pipeline* en el módulo  $MrgnPipeline$ , se obtiene un nuevo margen en cada ciclo de reloj. Una vez disponible el primer margen, se habilitan los contadores  $CntIdx$  (que asigna el índice correspondiente al dato),  $CntR$  (que determina el registro destino) y  $CntRB$  (que define el banco de registros

de almacenamiento). Los multiplexores *DecRSrc*, *RSrc* y *RBSrc* permiten el paso de la señal proveniente desde los contadores respectivos. El multiplexor *MrgnSrc*, selecciona el dato de entrada proveniente del módulo *MrgnPipeline* al cuál se le concatena tanto la cuenta de *CntIndx* como la cuenta de *CntR*. Asimismo, el multiplexor *TrigMtreeSrc* dirige la cuenta de *CntRB* (decodificada mediante *DecRB*) hacia los módulos responsables de calcular el máximo (*Mtree\_0-7*) una cantidad de ciclos de reloj igual al número de bancos de registros antes de que se hayan procesado los primeros 512 datos, de modo que una vez terminado el llenado inicial, ya esté disponible al menos el máximo de la primera unidad de memoria para su posterior comparación.

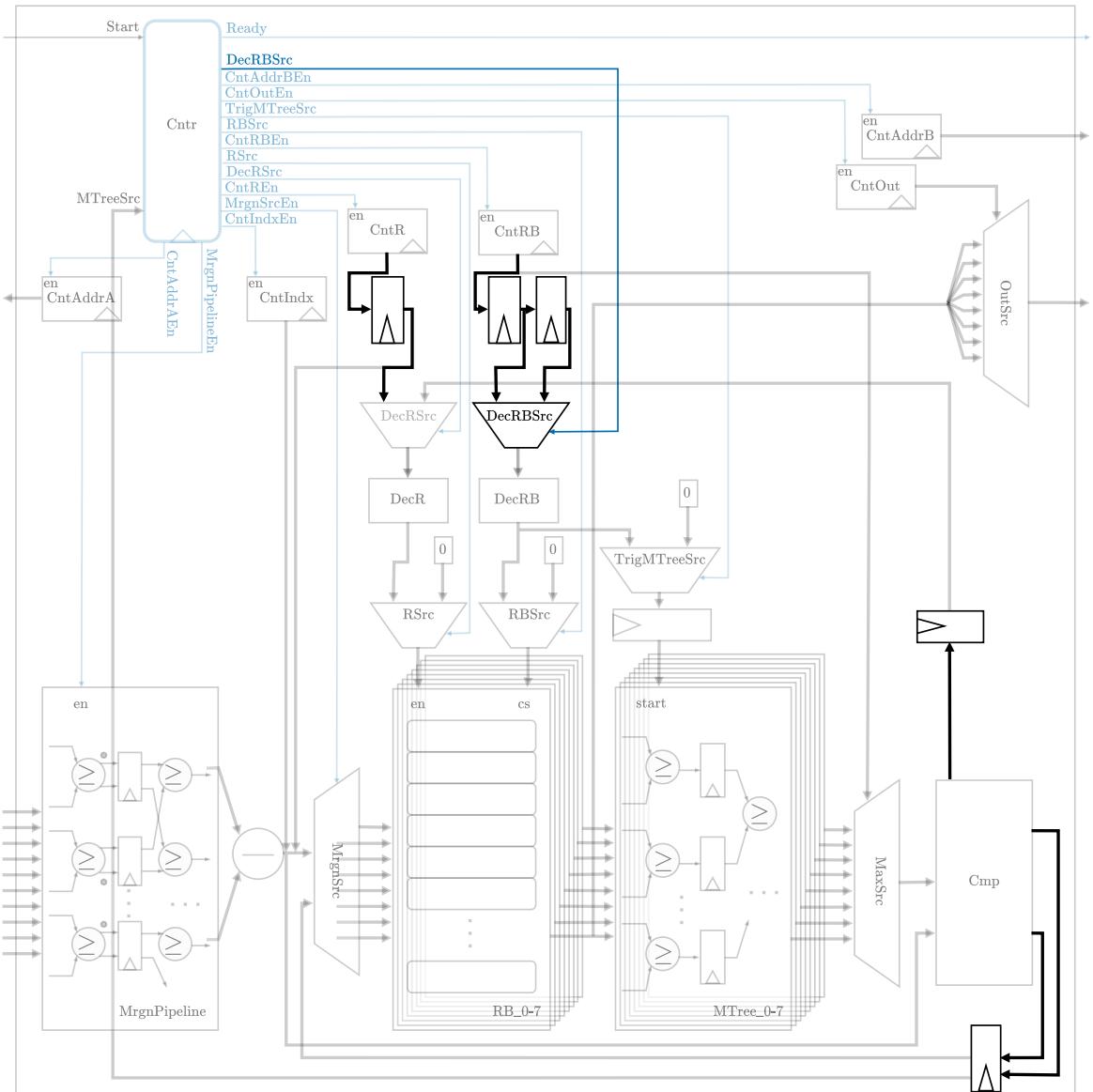
En la segunda etapa del procesamiento, el módulo de comparación *Cmp* suministra el dato de entrada a los bancos de registros, indicando además el registro en el que debe almacenarse y generando una *flag* que indica al controlador *Cntr* la necesidad de efectuar un reemplazo. En consecuencia, *MrgnSrc* y *DecRSrc* seleccionan los datos provenientes desde *Cmp*, *RSrc*, *RBSrc* y *TrigMtreeSrc* seleccionan sus entradas distintas de 0 únicamente en caso de efectuarse un reemplazo. Si no se ejecuta el reemplazo, la entrada seleccionada es 0, lo que impide la actualización de los datos en los bancos y evita el reinicio del cálculo del máximo. Es importante señalar que, en esta etapa, el selector del *MaxSrc* se determina a partir de la cuenta de *CntRB*.

La tercera y última etapa consiste en escribir los índices en la BRAM de salida. Para esto, se habilita *CntOut* que va al selector de *OutSrc* encargado de elegir cada uno de los índices. Este proceso se sincroniza con el contador *CntAddrB*, el cual genera las direcciones de escritura.

La arquitectura propuesta opera a una frecuencia máxima de 100MHz. A frecuencias superiores, los requerimientos de *timing* no se cumplen. Se ha identificado que los caminos críticos que presentan este problema van desde los contadores *CntR* y *CntRB* hasta *RB*, y también, desde *MTree* hasta *RB*. Cabe destacar que las dificultades de *timing* en estos caminos se deben principalmente a la longitud de la interconexión, y no a los recursos lógicos involucrados.

Para aumentar la frecuencia máxima de procesamiento, se implementaron modificaciones en el diseño. Dado que las dificultades de *timing* están relacionadas con la longitud de las interconexiones, se añaden registros intermedios para reducir

esta distancia, como se muestra en la Figura 5.10. Los registros en las salidas de  $CntR$  y  $CntRB$  tienen la función de acortar la longitud de las conexiones con  $RB$ . Como las señales generadas por  $CntR$  y  $CntRB$  se retrasan un ciclo debido a estos registros adicionales, ambos contadores se habilitan un ciclo de reloj antes en comparación con la implementación anterior. De manera similar, los registros en las salidas de  $Cmp$ , disminuyen la longitud del conexionado entre  $MTree$  y  $RB$ .



**Figura 5.10:** Arquitectura digital del acelerador *hardware* para *Margin Sampling*, con modificaciones para solucionar problemas de *timing*. Fuente: Elaboración propia.

Además, debido a la incorporación de estos nuevos registros en las salidas de

$Cmp$ , se requiere un retardo adicional de un ciclo de reloj en el valor del contador  $CntRB$  después de procesar el *batch* inicial. Para ello, se introduce otro registro, cuya entrada proviene del registro que retarda la señal del contador  $CntRB$ . Adicionalmente, se añade el multiplexor  $DecRBSrc$ , encargado de seleccionar la salida de  $CntRB$  retardada dos ciclos, una vez finalizado el procesamiento del *batch* inicial. Cabe destacar que las señales de control son ajustadas de tal manera que el funcionamiento mantenga la sincronización

De este modo, esta arquitectura alcanza una frecuencia máxima de operación de 110 MHz. Ambas arquitecturas fueron validadas a través de dos pruebas, que consistieron en comparar los índices resultantes de la emulación en *software* frente a la implementación en la placa de desarrollo. Para la primera se usaron datos de entrada aleatorios en cada iteración, se realizaron 50 iteraciones. Los datos corresponden a números enteros en un rango de 0 a  $2^{16} - 1$ . En la segunda prueba los datos de entrada fueron predicciones hechas por los tres modelos de clasificación (MobileNetV1, EfficientNetB0 y ResNet50) en los dos conjuntos de datos (Fashion-MNIST y CIFAR-10). El formato numérico de las predicciones es punto flotante de 32 bits (*single precision*). Estas fueron trasformadas a formato punto fijo de 16 bits, multiplicando cada número por  $2^{16}$  (equivalente a  $\ll 16$ ). Todos los índices fueron coincidentes en cada prueba para ambas arquitecturas.

## Capítulo 6

### Resultados: Implementación en SoC

En este capítulo se presentan los resultados de la implementación de la arquitectura *hardware* diseñada en el Capítulo 6 sobre la tarjeta de desarrollo PYNQ-Z2 de Xilinx. A través del IDE Vivado 2024.2 se obtiene información sobre la utilización de recursos, el consumo de potencia, tiempo de ejecución y caminos críticos. Dado que la herramienta realiza optimizaciones al implementar el diseño, los resultados de utilización de recursos lógicos pueden diferir de acuerdo al diseño. Se presenta también, una discusión de los resultados obtenidos.

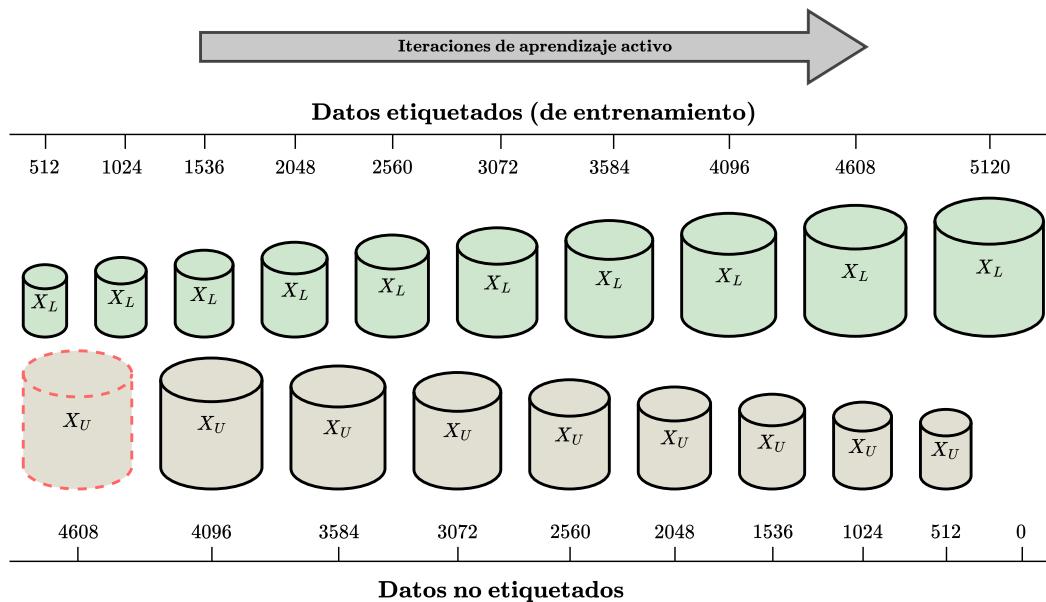
Como se mencionó en el Capítulo 6, para validar la implementación se realizaron dos tipos de prueba. La primera consistió en comparar los índices no coincidentes frente a la implementación emulada en *software* para datos de entrada aleatorios, esto fue repetido 50 veces. En la segunda prueba, también se evaluaron los índices no coincidentes en comparación a la emulación en *software*, pero esta vez se usaron predicciones de los 3 modelos de clasificación en ambos conjuntos de datos, es decir, 6 entradas diferentes. Tanto para el diseño con frecuencia máxima 100MHz como para el diseño con frecuencia máxima 110MHz, no hubo índices no coincidentes en ninguna prueba.

Si bien ambas arquitecturas fueron implementadas y validadas, los resultados que se muestran a continuación, corresponden al diseño que alcanza una frecuencia de operación máxima de 110MHz.

La implementación en *software*, para efectos de comparación, es realizada utilizando una CPU Intel Xeon en la plataforma Google Colaboratory. Esta plataforma se ejecuta en un entorno virtualizado sobre la infraestructura de Google Cloud, lo que

abstira muchos detalles del *hardware*. La instancia informa un procesador Intel Xeon a 2.20 GHz (2200 MHz) con 55 MB de caché (arquitectura Broadwell-EP, Modelo 79), diseñado para servidores y centros de datos. Aunque el consumo de energía bajo la carga teórica máxima (*Thermal Design Power*, TDP) exacto no se expone en un entorno virtual, procesadores Xeon Broadwell similares suelen tener un TDP entre 120 W y 150 W.

Como se muestra en la Figura 6.1, el caso crítico en estudio corresponde al caso en el que se presenta la mayor cantidad de ejemplos no etiquetados. Este caso está limitado por la cantidad de recursos disponibles en el FPGA.



**Figura 6.1:** Caso crítico en estudio (corresponde al caso con linea roja punteada). Fuente: Elaboración propia.

De esta forma, la cantidad de datos de entrada que son procesados por la estrategia de consulta en el caso crítico es 4608, desde donde 512 son seleccionados para su anotación.

## 6.1. Velocidad de computo

El número de ciclos de reloj que tarda la arquitectura diseñada en realizar el procesamiento está dado por la Ecuación 6.1:

$$N_{cycles} = (2 \times \log_2 N_{classes}) + N_{data} + (N_R \times N_{RB}), \quad (6.1)$$

donde  $N_{classes}$  corresponde al número de clases en las predicciones,  $N_{data}$  al número de datos de entrada,  $N_R$  a la cantidad de registros en cada banco de registros y  $N_{RB}$  al número de bancos de registros. El término  $2 \times \log_2 N_{classes}$ , está asociado al retardo inicial del *pipeline* que calcula el margen. Específicamente, la cantidad de ciclos que tarda este computo es  $(2 \times \log_2 N_{classes}) - 1$ . El producto  $N_R \times N_{RB}$  resulta en el tamaño del *batch*, que es igual a la cantidad de ciclos que toma escribir los resultados en la BRAM de salida.

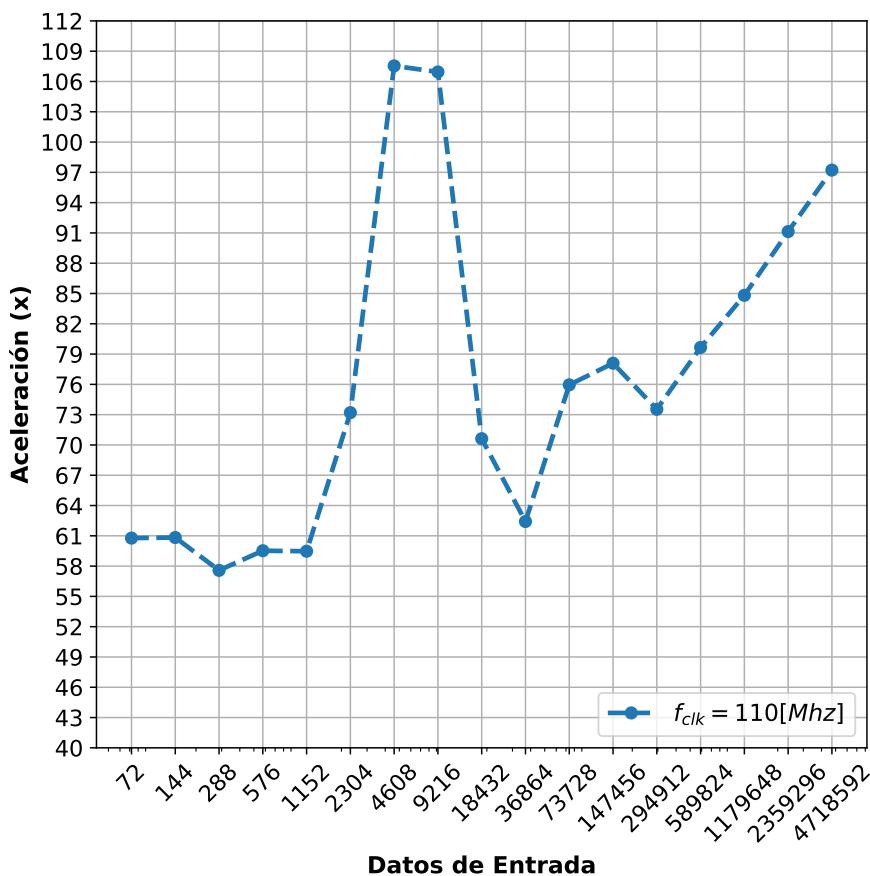
Para la arquitectura diseñada se tiene:  $N_{classes} = 10$ ,  $N_{data} = 4608$ ,  $N_R = 64$  y  $N_{RB} = 8$ . Por lo tanto, el número de ciclos del procesamiento está dado por:  $N_{cycles} = (2 \times \log_2 10) + 4608 + (64 \times 8) = 5128$ .

Con el número de ciclos, es posible encontrar la velocidad de computo como el producto entre el número de ciclos de reloj y el periodo. Dada una frecuencia de reloj  $f_{clk}$ , el periodo se encuentra como  $T_{clk} = 1/f_{clk}$ . De este modo, la velocidad de computo se expresa como muestra la Ecuación 6.2.

$$\begin{aligned} R_c &= N_{cycles} \times \frac{1}{f_{clk}} \\ &= N_{cycles} \times T_{clk} \end{aligned} \quad (6.2)$$

La mayor frecuencia de operación alcanzada es  $f_{clk} = 110 \text{ MHz}$ , por lo tanto, el periodo está dado por:  $T_{clk} = 1/110 \text{ MHz} = 9 \text{ ns}$ . De modo que la velocidad de cómputo se encuentra como:  $R_c^{HW} = 5128 \times 9 \text{ ns} = 46152 \text{ ns}$ . La velocidad en la implementación *software* base (basada en ordenamiento) es calculada como el promedio de 100 iteraciones en entradas generadas aleatoriamente, dando como resultado un tiempo de procesamiento  $R_c^{SW} = 5208980,79 \text{ ns}$ . Como se puede ver, el tiempo de procesamiento de la implementación *hardware* es inferior a la implementación *software* base. La aceleración, se puede encontrar como  $a = R_c^{SW}/R_c^{HW} = 4963433,74 \text{ ns}/46152 \text{ ns} = 107,54$ . Es decir, la implementación en *hardware* es aproximadamente 100 veces más rápida que la implementación en *software* base.

Con las ecuaciones mostradas, es posible realizar una estimación de la aceleración considerando cantidades distintas de datos de entrada, donde el *batch* es siempre un 10 % del total de datos y las unidades de memoria así como su capacidad son determinadas de tal forma que se utilizan la menor cantidad con la mayor capacidad posible. La Figura 6.2 muestra la curva de aceleración, para cantidades de datos de entrada desde  $72 ((2^3 \times 10) - 2^3)$  hasta  $4718592 ((2^{19} \times 10) - 2^{19})$ .



**Figura 6.2:** Curva de aceleración. Fuente: Elaboración propia.

Como se puede observar en la Figura 6.2, la tendencia es que la aceleración aumenta a medida que crece la cantidad de datos de entrada. En el mejor caso mostrado, la implementación en *hardware* alcanza una velocidad aproximadamente 117 veces mayor para una entrada de 5242880 datos, mientras que en el caso con la menor cantidad de datos presentada (80 datos), es 55 veces más rápida.

Estos factores de aceleración se sostienen en el hecho de que el impacto de la cantidad de datos de entrada sobre la velocidad de cómputo es lineal, ya que en la ecuación de ciclos el término correspondiente a los datos simplemente se suma al

total. Esto implica que, al aumentar el número de datos, el tiempo de procesamiento crece de forma proporcional. En contraste, en algoritmos de ordenamiento, por ejemplo, la complejidad suele crecer de manera cuadrática o logarítmica, lo que resulta en un impacto mucho mayor en el tiempo de ejecución. Aunque los resultados indican que escalar el diseño es idóneo, el incremento proporcional del tamaño del *batch*, la cantidad de bits requeridos y, en consecuencia, de los árboles de comparación para hallar el máximo, limita la escalabilidad debido a los recursos disponibles. Además, en diseños muy grandes pueden surgir problemas de ruteo, como mayores retardos y dificultades en la distribución de señales críticas (*clk*), que afectan el *timing*.

## 6.2. Utilización de recursos

La Tabla 6.1 muestra la utilización de recursos por jerarquía del sistema, que es separada en:

- AXI BRAM Controllers: Estos módulos actúan como AXI Slave, proporcionando una interface para conectar un AXI Master y BRAM. El sistema utiliza 2 de estos controladores, uno para la BRAM de entrada y otro para la BRAM de salida.
- AXI GPIO: Este módulo actúa como un AXI Slave, que permite la interacción de un AXI Master con los periféricos en PL. En el sistema, este módulo es usado para las señales que indican el inicio y término del procesamiento *start* y *ready*, respectivamente.
- AXI SmartConnect: La tarea de este módulo es la gestionar el interconexión entre múltiples AXI Master y múltiples AXI Slaves de forma independiente. En el diseño presentado, se encarga de conectar el procesador ARM (AXI Master) con los módulos AXI BRAM Controller (AXI Slave).
- Reset PS7: Módulo encargado de llevar el sistema a un estado conocido al momento de energizar, y al momento de activar la señal de *reset*.
- MSA (Margin Sampling Accelerator): Corresponde a la arquitectura digital diseñada para implementar el algoritmo *Margin Sampling*.

**Tabla 6.1:** Utilización de recursos en las jerarquías principales. Fuente: Elaboración propia.

|                      | Slice LUTs | Slice Registers | Slices  | Block RAM Tiles |
|----------------------|------------|-----------------|---------|-----------------|
| AXI BRAM Controllers | 459        | 1064            | 373     | 0               |
| AXI GPIO             | 35         | 43              | 16      | 0               |
| AXI SmartConnect     | 6902       | 9138            | 2575    | 0               |
| Reset PS7            | 17         | 33              | 12      | 0               |
| MSA                  | 16199      | 36351           | 8751    | 65              |
| Total                | 23612      | 46629           | 11727   | 65              |
| Disponible           | 53200      | 106400          | 13300   | 140             |
| Porcentaje           | 44.38 %    | 43.82 %         | 88.17 % | 46.43 %         |

MSA es la jerarquía que utiliza mayor cantidad de recursos en todos los campos, con un 74.80 % de los recursos utilizados y un 35.04 % del total de recursos disponibles en la plataforma de trabajo.

La Tabla 6.2, desglosa el uso por módulos dentro de MSA. Estos son agrupados según la función que realizan. Cabe destacar que, los recursos pueden ser compartidos entre módulos, esto explica que el total reportado, supere al total de MSA en la Tabla 6.1.

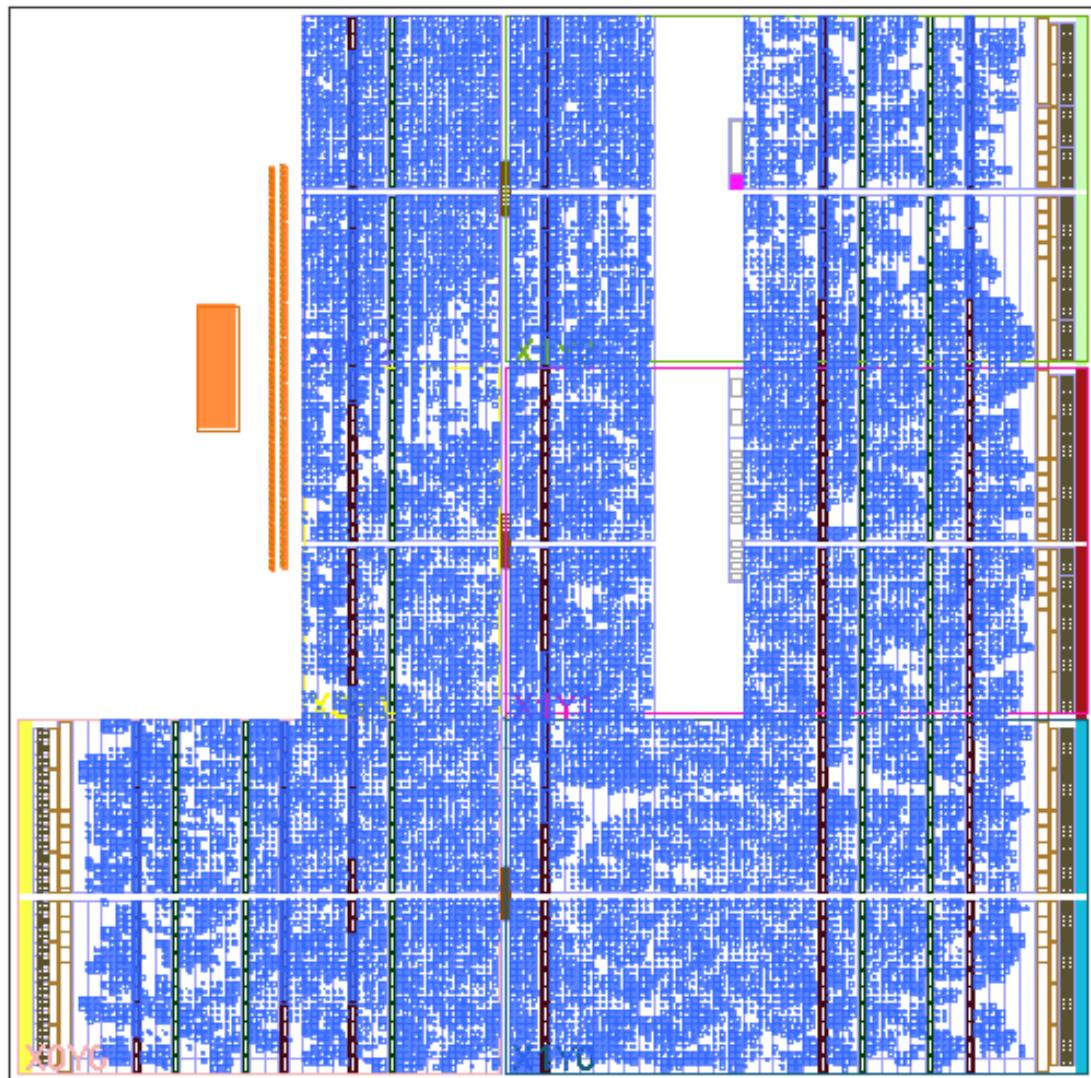
**Tabla 6.2:** Utilización de recursos de los módulos dentro de MSA. Fuente: Elaboración propia.

|                | Slice LUTs | Slice Registers | Slices  | Block RAM Tiles |
|----------------|------------|-----------------|---------|-----------------|
| Controller     | 61         | 13              | 48      | 0               |
| Counters       | 45         | 100             | 53      | 0               |
| Muxes          | 1781       | 0               | 473     | 0               |
| Margin         | 360        | 467             | 112     | 0               |
| Registers      | 641        | 208             | 517     | 0               |
| Register Banks | 4609       | 17920           | 7641    | 0               |
| Max Trees      | 8496       | 17640           | 3993    | 0               |
| Comparator     | 0          | 0               | 2       | 0               |
| BRAMs          | 221        | 2               | 146     | 65              |
| Total          | 16214      | 36350           | 12985   | 65              |
| Disponible     | 53200      | 106400          | 13300   | 140             |
| Porcentaje     | 30.48 %    | 34.16 %         | 97.63 % | 46.43 %         |

Se puede notar que los módulos que presentan mayor uso de registros son los

bancos de registros, y los árboles de comparación para encontrar el máximo, con 49.29 % y 48.52 % respectivamente. El resultado es esperado y coherente con el diseño, ya que cada valor almacenado requiere disponer de suficientes bits para representar el margen (16 bits), el índice (13 bits) y la posición (6 bits), sumando un total de 35 bits. Esta utilización de recursos podría reducirse implementando técnicas alternativas para hacer seguimiento de esta información sin necesidad de almacenar los datos directamente; sin embargo, ello implicaría incrementar la complejidad de la lógica de control.

La Figura 6.3 muestra la distribución del diseño en los recursos disponibles en el FPGA:



**Figura 6.3:** Diseño implementado. Fuente: Elaboración propia.

Se puede notar (en azul), que el diseño es distribuido utilizando casi todo el espacio disponible en el FPGA.

### 6.3. Consumo de potencia

El consumo de potencia de la implementación fue estimado en Vivado. Como se mencionó previamente, la arquitectura funciona con una frecuencia de reloj de 110 MHz en PL, mientras que el procesador ARM opera a 650 MHz en PS. La Tabla 6.3 presenta el consumo de potencia dinámica desglosado por recurso disponible en el sistema.

**Tabla 6.3:** Consumo de potencia dinámica de la implementación en SoC por recurso. Fuente: Elaboración propia.

|         | Potencia (mW) | Potencia (%) |
|---------|---------------|--------------|
| PS7     | 1256          | 73           |
| Signals | 217           | 13           |
| Clocks  | 112           | 6            |
| BRAM    | 91            | 5            |
| Logic   | 54            | 3            |
| Total   | 1730          |              |

Se observa que, el consumo total del sistema es de 1730 mW. Si se considera que la implementación *software* se ejecuta sobre un procesador con TDP de 150 W, la implementación *hardware* consume 86.7 veces menos potencia que la implementación en *software*.

El mayor consumo está asociado a PS7, es decir, al procesador ARM embebido en el SoC. Este consume un 73 % de la potencia dinámica del sistema, que es un consumo típico y esperado en esta unidad. En segundo lugar, se encuentra *Signals*, el consumo de las señales individuales dentro del sistema con un 13 %, donde 11 % corresponde a señales de datos. Para un análisis más detallado, la tabla 6.4 muestra el desglose de consumo de potencia de los módulos en la lógica programable.

**Tabla 6.4:** Consumo de potencia de implementación en SoC por módulo jerárquico.  
Fuente: Elaboración propia.

|                | Potencia (mW) | Potencia (%) |
|----------------|---------------|--------------|
| MS Accel       | 382           | 20           |
| AXI SMC        | 80            | 4            |
| AXI BRAM Cntrs | 11            | ~1           |
| AXI GPIO       | <1            | <1           |
| Reset          | <1            | <1           |
| Total          | ~475          |              |

El mayor consumo dentro de la lógica programable proviene del módulo MSHW, el acelerador *hardware* diseñado para *Margin Sampling*, que representa un 20 % del consumo total. Dentro de este módulo, los principales consumidores de potencia son el registro que almacena el resultado de la comparación a la salida del módulo de comparación, con un 6 %, y la BRAM de entrada, con un 5 %. El resto de los módulos en MS Accel tienen un consumo del 1 % o menos. Por otro lado, el módulo AXI SMC, encargado de gestionar las interconexiones entre los puertos AXI del sistema, presenta un consumo del 4 %.

Aunque los registros en los bancos de registro y en los árboles de comparación actualizan sus valores solo cuando es necesario, existen otros que permanecen habilitados de forma permanente, entre ellos el que representa el 6 % del consumo total de potencia. Para reducir este consumo, se podría implementar una lógica de control que active dichos registros únicamente cuando se requiera.

## Capítulo 7

# Conclusiones y trabajo futuro

La implementación en SoC de la arquitectura digital desarrollada en este trabajo realiza la selección de los índices más inciertos para un modelo de clasificación de imágenes, utilizando el margen entre las dos probabilidades más altas de cada predicción. En términos de velocidad de cómputo, el circuito propuesto, mediante el cómputo en línea, alcanza una aceleración ligeramente superior a 100 veces respecto a la implementación base en *software* para el caso crítico analizado (ver Figura 6.2), manteniendo además un consumo de potencia inferior a 2 W (ver Tabla 6.3).

La arquitectura es parametrizable, lo que permite ajustar sus parámetros para admitir diferentes cantidades de entradas. Sin embargo, el diseño demanda recursos de manera tal que escalarlo a un *batch* en la siguiente potencia de 2 ( $2^{10}$ ) imposibilitaría su implementación en la plataforma de desarrollo actual. La implementación presenta una utilización global elevada de los recursos, a pesar de que los componentes elementales no estén completamente saturados (ver Tabla 6.1). Esto indica que el mapeo del diseño ocupa una gran área dentro del FPGA, lo cual es una consecuencia inherente al flujo de diseño FPGA, en el que la arquitectura es mapeada en recursos fijos. En contraste, en un flujo de diseño ASIC se podría optimizar la distribución de los recursos para adecuarla mejor al diseño. Además de la escalabilidad, el uso intensivo de recursos puede limitar la posibilidad de integrar el circuito con otras funcionalidades, como un acelerador para la inferencia, lo que en el contexto actual solo sería viable para volúmenes pequeños de datos de entrada, teniendo como referencia el caso crítico en estudio.

En todas las estrategias de consulta analizadas, el desempeño supera el 70 % en área bajo la curva de aprendizaje, alcanzando cerca del 80 % en los mejores casos y utilizando menos del 40-60 % del total de ejemplos de entrenamiento en comparación al aprendizaje pasivo (ver Figura 4.1 y Figura 4.2). Esto reafirma que diseñando algoritmos de aprendizaje activo que permitan seleccionar las instancias más ambiguas se mejora el desempeño de clasificadores de imágenes basados en redes neuronales. *Margin Sampling* es ligeramente superior a las demás estrategias de consulta (ver Tabla 4.4 y 4.8). Aunque en su implementación se modificó el algoritmo para lograr un cómputo en línea y se realizaron ciertos compromisos en desempeño, las pruebas realizadas indican que las diferencias entre implementaciones son mínimas, con una diferencia máxima en el área bajo la curva de 0.0016, lo que sugiere que el método de selección propuesto puede combinarse con otras estrategias de consulta sin afectar negativamente el rendimiento de los algoritmos de aprendizaje activo (ver Figura 5.8 y Tabla 5.3).

Como trabajo futuro, se propone explorar combinaciones de unidades de distintos tamaños. Las pruebas realizadas consideraron unidades de igual tamaño, observándose que el error disminuye al utilizar la menor cantidad posible de unidades con la mayor capacidad. Sin embargo, al emplear memorias de tamaños diferentes, podría incrementarse la capacidad de ciertas unidades, reduciendo aún más el error. Por ejemplo, si se considera un *batch* de 16, la mejor opción con unidades de igual tamaño es utilizar 4 unidades de capacidad 4 cada una; en cambio, con unidades de distinto tamaño, se puede lograr la misma capacidad con 3 unidades: dos de tamaño 4 y una de tamaño 8. Al igual que en los casos anteriores, los valores máximos se calculan antes de cada nueva comparación.

Por otra parte, si bien es posible implementar arquitecturas para diferentes cantidades de datos de entrada, una misma arquitectura no podría procesar volúmenes de datos de entrada diferentes (dado por el tamaño del conjunto de datos no etiquetado). Ante esta limitación, se propone como trabajo futuro modificar la lógica de control de tal modo que la implementación sea capaz de procesar datos de entrada de distinto tamaño. Además, se sugiere explorar el escenario de aprendizaje activo *stream-based*, donde es posible seleccionar una instancia del conjunto de datos no etiquetado a medida que llega, sin necesidad de tener que procesar el conjunto en su totalidad, característica favorable en un contexto donde hay recursos limitados.

## Bibliografía

- [1] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, “Imagenet large scale visual recognition challenge,” *International journal of computer vision*, vol. 115, pp. 211–252, 2015.
- [2] H. Xiao, K. Rasul, and R. Vollgraf, “Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms,” *arXiv preprint arXiv:1708.07747*, 2017.
- [3] A. Krizhevsky, G. Hinton, *et al.*, “Learning multiple layers of features from tiny images,” 2009.
- [4] M. Elgendi, *Deep learning for vision systems*. Manning, 2020.
- [5] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Communications of the ACM*, vol. 60, pp. 84–90, 6 2017.
- [7] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.
- [8] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.
- [9] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning,” *Image Recognition*, vol. 7, no. 4, pp. 327–336, 2015.
- [10] L. Chen, S. Li, Q. Bai, J. Yang, S. Jiang, and Y. Miao, “Review of image classification algorithms based on convolutional neural networks,” *Remote Sensing 2021, Vol. 13, Page 4712*, vol. 13, p. 4712, 11 2021.
- [11] A. Tharwat and W. Schenck, “A survey on active learning: State-of-the-art, practical challenges and research directions,” *Mathematics 2023, Vol. 11, Page 820*, vol. 11, p. 820, 2 2023.

- [12] B. Settles, “Computer sciences department active learning literature survey,” *University of Wisconsin-Madison Department of Computer Sciences*, 2009.
- [13] B. Settles and M. Craven, “An analysis of active learning strategies for sequence labeling tasks,” in *proceedings of the 2008 conference on empirical methods in natural language processing*, pp. 1070–1079, 2008.
- [14] P. Dhillesararao, S. Boppu, M. S. Manikandan, and L. R. Cenkeramaddi, “Efficient hardware architectures for accelerating deep neural networks: Survey,” *IEEE access*, vol. 10, pp. 131788–131828, 2022.
- [15] D. G. Bailey, *Design for Embedded Image Processing on FPGAs*. 2023.
- [16] A. Mouri Zadeh Khaki and A. Choi, “Optimizing deep learning acceleration on fpga for real-time and resource-efficient image classification,” *Applied Sciences*, vol. 15, no. 1, p. 422, 2025.
- [17] Z. Li, F. Z. Hong, and C. P. Yue, “Fpga-based acceleration of neural network for image classification using vitis ai,” *arXiv preprint arXiv:2412.20974*, 2024.
- [18] Y. Ji, D. Wei, L. Shi, P. Liu, C. Li, J. Yu, and X. Cai, “An active learning based latency prediction approach for neural network architecture,” in *2024 4th International Conference on Neural Networks, Information and Communication (NNICE)*, pp. 967–971, IEEE, 2024.
- [19] Q. Sun, C. Bai, H. Geng, and B. Yu, “Deep neural network hardware deployment optimization via advanced active learning,” in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1510–1515, IEEE, 2021.
- [20] R. Gonzalez and R. Woods, *Digital Image Processing, Global Edition*. Pearson Education, 2018.
- [21] G. Kumar and P. K. Bhatia, “A detailed review of feature extraction in image processing systems,” *International Conference on Advanced Computing and Communication Technologies, ACCT*, pp. 5–12, 2014.
- [22] T. Ojala, M. Pietikainen, and D. Harwood, “Performance evaluation of texture measures with classification based on kullback discrimination of distributions,” in *Proceedings of 12th International Conference on Pattern Recognition*, pp. 582–585, IEEE Comput. Soc. Press, 1994.
- [23] D. G. Lowe, “Object recognition from local scale-invariant features,” in *Proceedings of the seventh IEEE international conference on computer vision*, vol. 2, pp. 1150–1157, Ieee, 1999.
- [24] H. Bay, T. Tuytelaars, and L. Van Gool, “Surf: Speeded up robust features,” in *Computer Vision–ECCV 2006: 9th European Conference on Computer Vision, Graz, Austria, May 7-13, 2006. Proceedings, Part I 9*, pp. 404–417, Springer, 2006.

- [25] P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1, 2001.
- [26] G. Csurka, C. Dance, L. Fan, J. Willamowski, and C. Bray, “Visual categorization with bags of keypoints,” in *Workshop on statistical learning in computer vision, ECCV*, vol. 1, pp. 1–2, Prague, 2004.
- [27] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)*, pp. 886–893, IEEE, 2005.
- [28] I. Wickramasinghe and H. Kalutarage, “Naive bayes: applications, variations and vulnerabilities: a review of literature with code snippets for implementation,” *Soft Computing*, vol. 25, pp. 2277–2293, 2 2021.
- [29] T. Cover and P. Hart, “Nearest neighbor pattern classification,” *IEEE Transactions on Information Theory*, vol. 13, pp. 21–27, 1 1967.
- [30] C. Cortes and V. Vapnik, “Support-vector networks,” *Machine learning*, vol. 20, pp. 273–297, 1995.
- [31] J. R. Quinlan, “Induction of decision trees,” *Machine Learning*, vol. 1, pp. 81–106, 1986.
- [32] S. L. S. SALZBERG and C. EDU, “Book review: 04.5: Programs for machine learning by j. ross quinlan. morgan kaufmann publishers, inc., 1993.,” *Machine Learning*, vol. 77, p. 176, 1994.
- [33] L. Breiman, “Random forests,” *Machine Learning*, vol. 45, pp. 5–32, 2001.
- [34] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, “An image is worth 16x16 words: Transformers for image recognition at scale,” *ICLR 2021 - 9th International Conference on Learning Representation*, 10 2021.
- [35] S. R. Dubey, S. K. Singh, and B. B. Chaudhuri, “Activation functions in deep learning: A comprehensive survey and benchmark,” *Neurocomputing*, vol. 503, pp. 92–108, 9 2022.
- [36] X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks,” in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 315–323, JMLR Workshop and Conference Proceedings, 2011.
- [37] A. L. Maas, A. Y. Hannun, A. Y. Ng, *et al.*, “Rectifier nonlinearities improve neural network acoustic models,” in *Proc. icml*, vol. 30, p. 3, Atlanta, GA, 2013.
- [38] D. Hendrycks and K. Gimpel, “Gaussian error linear units (gelus),” *arXiv preprint arXiv:1606.08415*, 2016.

- [39] Y. Tian and Y. Zhang, “A comprehensive survey on regularization strategies in machine learning,” *Information Fusion*, vol. 80, pp. 146–166, 2022.
- [40] G. Hinton, N. Srivastava, and K. Swersky, “Neural networks for machine learning lecture 6a overview of mini-batch gradient descent,” *Cited on*, vol. 14, no. 8, p. 2, 2012.
- [41] J. D. JDUCHI and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization \* elad hazan,” *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011.
- [42] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.
- [43] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” *arXiv preprint arXiv:1704.04861*, 2017.
- [44] F. Sultonov, J.-H. Park, S. Yun, D.-W. Lim, and J.-M. Kang, “Mixer u-net: An improved automatic road extraction from uav imagery,” *Applied Sciences*, vol. 12, no. 4, p. 1953, 2022.
- [45] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “Mobilenetv2: Inverted residuals and linear bottlenecks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.
- [46] M. Tan and Q. V. Le, “Efficientnet: Rethinking model scaling for convolutional neural networks,” *36th International Conference on Machine Learning, ICML 2019*, vol. 2019-June, pp. 10691–10700, 5 2019.
- [47] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Łukasz Kaiser, and I. Polosukhin, “Attention is all you need,” *Advances in Neural Information Processing Systems*, vol. 2017-December, pp. 5999–6009, 6 2017.
- [48] J. Maurício, I. Domingues, and J. Bernardino, “Comparing vision transformers and convolutional neural networks for image classification: A literature review,” *Applied Sciences 2023, Vol. 13, Page 5521*, vol. 13, p. 5521, 4 2023.
- [49] T. Scheffer, C. Decomain, and S. Wrobel, “Active hidden markov models for information extraction,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 2189, pp. 309–318, 2001.
- [50] D. D. Lewis, “A sequential algorithm for training text classifiers: Corrigendum and additional data,” in *Acm Sigir Forum*, vol. 29, pp. 13–19, ACM New York, NY, USA, 1995.
- [51] H. S. Seung, M. Opper, and H. Sompolinsky, “Query by committee,”

- Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, pp. 287–294, 1992.
- [52] S. Berardo, E. Favero, and N. Neto, “Active learning with clustering and unsupervised feature learning,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9091, pp. 281–290, 2015.
  - [53] M. Wang, F. Min, Z. H. Zhang, and Y. X. Wu, “Active learning through density clustering,” *Expert Systems with Applications*, vol. 85, pp. 305–317, 11 2017.
  - [54] W. Valenzuela, J. E. Soto, P. Zarkesh-Ha, and M. Figueroa, “Face recognition on a smart image sensor using local gradients,” *Sensors*, vol. 21, no. 9, p. 2901, 2021.
  - [55] H. Li, Y. Tang, Z. Que, and J. Zhang, “Fpga accelerated post-quantum cryptography,” *IEEE Transactions on Nanotechnology*, vol. 21, pp. 685–691, 2022.
  - [56] D. M. Harris and S. Harris, *Digital Design and Computer Architecture 2nd edition*. 2012.
  - [57] A. Abdelwahab, A. Afifi, and M. Salama, “An integrated active deep learning approach for image classification from unlabeled data with minimal supervision,” *Electronics 2024, Vol. 13, Page 169*, vol. 13, p. 169, 12 2023.
  - [58] C. A. Flores, R. L. Figueroa, and J. E. Pezoa, “Active learning for biomedical text classification based on automatically generated regular expressions,” *IEEE Access*, vol. 9, pp. 38767–38777, 2021.
  - [59] XilinxInc, “Zynq-7000 soc first generation architecture,” URL: <https://docs.amd.com/v/u/en-US/ds190-Zynq-7000-Overview>, 2012.
  - [60] K. Agrawal and A. Asati, “Improved implementation of pynq-based fft hardware accelerator,” in *2024 2nd International Conference on Device Intelligence, Computing and Communication Technologies (DICCT)*, pp. 414–418, 2024.
  - [61] S. Bbouzidi, G. Hcini, I. Jdey, and F. Drira, “Convolutional neural networks and vision transformers for fashion mnist classification: A literature review,” *arXiv preprint arXiv:2406.03478*, 2024.
  - [62] O. Nocentini, J. Kim, M. Z. Bashir, and F. Cavallo, “Image classification using multiple convolutional neural networks on the fashion-mnist dataset,” *Sensors*, vol. 22, no. 23, p. 9544, 2022.
  - [63] C.-J. Lin, C.-H. Lin, and S.-H. Wang, “Integrated image sensor and light convolutional neural network for image classification,” *Mathematical Problems in Engineering*, vol. 2021, no. 1, p. 5573031, 2021.

- 
- [64] C. A. Flores, R. L. Figueroa, and J. E. Pezoa, “Fregex: A feature extraction method for biomedical text classification using regular expressions,” in *2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pp. 6085–6088, 2019.
  - [65] J. E. Soto, P. Ubisse, Y. Fernández, C. Hernández, and M. Figueroa, “A high-throughput hardware accelerator for network entropy estimation using sketches,” *IEEE Access*, vol. 9, pp. 85823–85838, 2021.
  - [66] J. E. Soto, S. Vera, Y. Fernández, D. Yunge, C. Hernández, and M. Figueroa, “A sketch-based algorithm for network-flow entropy estimation on programmable switches using p4,” in *2023 26th Euromicro Conference on Digital System Design (DSD)*, pp. 79–86, 2023.
  - [67] C. Gallardo-Pavesi, Y. Fernandez, J. E. Soto, C. Hernández, and M. Figueroa, “A hardware accelerator for quantile estimation of network packet attributes,” in *2024 27th Euromicro Conference on Digital System Design (DSD)*, pp. 114–121, IEEE, 2024.