UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

# "Prestaciones y energía de esquemas de mapeo con precarga para aceleradores CNN sistólicos"

Trabajo Fin de Grado

**Grado en Ingeniería Informática**

**Autor**: Pau Castelló i Ferrer

**Tutores**: Julio Sahuquillo Borrás, Salvador Vicente Petit Martí

2018-2019

# Agradecimientos

Quiero expresar mi agradecimiento a todas y cada una de las personas que de una forma u otra han participado en la realización de este TFG.

En primer lugar, a todos los profesores y profesoras que durante estos cuatro años han invertido su tiempo y esfuerzo en que adquiera los conocimientos y competencias necesarias para poder realizar este trabajo.

De forma más específica agradecer la paciencia y el interés mostrado por los tutores del proyecto dado que sin sus orientaciones éste no hubiera sido posible. Destacar también la ayuda de Ricardo Marín, por ayudarnos con la realización del proyecto mediante el clúster de la universidad.

Para finalizar, quiero resaltar la labor de mi compañero Eduardo Yago Vicent por la gran capacidad de adaptación con la que hemos afrontado este proyecto, dado que ha sido crucial para su correcto desempeño.

Y por supuesto, en el aspecto personal, mi gratitud a mi familia por el apoyo que me ha brindado en todo momento para seguir adelante.

# Abstract

In this work we focus on Convolutional Neural Networks. The rise of this type of neural networks is due to its potential use both for image recognition, and for certain applications related to artificial intelligence. Research on these types of networks has increased year after year and as new uses arise the necessary computing capacity has increased to such an extent that conventional processors have difficulty carrying them out and do not offer the desired energy efficiency.

Since conventional processors are not suitable, a new type of processors called Systolic Arrays or Systolic Processors arises, which specialize in the computation of this type of neural networks. Some of the examples on which we have worked for the realization of this project are the Google TPU or the Eyeriss.

The objective of this work is to evaluate the performance and energy efficiency of these processors by varying various architectural parameters. For this we have chosen a systolic Array simulator known as SCALE-Sim master. This simulator allows to obtain performance and efficiency results for a certain processor configuration.

The results show that for the majority of applications studied, taking into account energy efficiency, an array of 64x64 PE is suitable, although rectangular (non-square) arrays may be interesting to improve some applications. On the other hand, it has been verified that the memory bandwidth can have a great impact on the execution time, so it is necessary that the systolic array incorporate this parameter in the calculation of the performance.



**Keywords:** Neural Networks, Convolutional Neural Networks, systolic arrays, architectural parameters, energy efficiency, performance.

# Resumen

En este trabajo nos centramos en la Redes Neuronales Convolucionales (en inglés, *Convolutional Neural Networks* o *CNN*). El auge de este tipo de redes neuronales se debe a su uso potencial tanto para el reconocimiento de imágenes, como para ciertas aplicaciones relacionadas con la inteligencia artificial. La investigación sobre este tipo de redes ha aumentado año tras año y a medida que surgen nuevos usos la capacidad de cómputo necesaria ha aumentado hasta tal punto que los procesadores convencionales tienen dificultades para llevarlas a cabo y no ofrecen la eficiencia energética deseada.

Visto que los procesadores convencionales no son adecuados, surge un nuevo tipo de procesadores llamados Arrays Sistólicos o Procesadores Sistólicos, los cuales se especializan en el cómputo de este tipo de redes neuronales. Algunos de los ejemplos sobre los que hemos trabajado para la realización de este proyecto son la TPU de Google o el Eyeriss.

El objetivo de este trabajo es evaluar el desempeño y eficiencia energética de estos procesadores variando diversos parámetros arquitectónicos. Para ello hemos elegido un simulador de Arrays Sistólicos conocido como SCALE-Sim master. Este simulador permite obtener resultados de rendimiento y eficiencia para una determinada configuración de procesador.

Los resultados muestran que, para la mayoría de las aplicaciones estudiadas, teniendo en cuenta la eficiencia energética, un array de 64x64 PE resulta adecuado, aunque arrays rectangulares (no cuadrados) pueden resultar interesantes para mejorar algunas aplicaciones. Por otro lado, se ha comprobado que el ancho de banda de memoria puede tener un gran impacto sobre el tiempo de ejecución, por lo que es necesario que los modelos de array sistólico incorporen este parámetro en el cálculo de las prestaciones.

**Palabras clave:** Redes neuronales, Redes Neuronales Convolucionales, arrays sistólicos, parámetros arquitectónicos, eficiencia energética, rendimiento.

# Resum

En aquest treball ens centrem en la Xarxes Neuronals Convolucionals (en anglès, Convolutional Neural Networks o CNN). L'auge d'aquest tipus de xarxes neuronals es deu al seu ús potencial tant per al reconeixement d'imatges, com per a certes aplicacions relacionades amb la intel·ligència artificial. La investigació sobre aquest tipus de xarxes ha augmentat any rere any i a mesura que sorgeixen nous usos la capacitat de còmput necessària ha augmentat fins a tal punt que els processadors convencionals tenen dificultats per a dur-les a terme i no ofereixen l'eficiència energètica desitjada.

Vist que els processadors convencionals no són adequats, sorgeix un nou tipus de processadors anomenats Arrays Sistòlics o Processadors Sistòlics, els quals s'especialitzen en el còmput d'aquest tipus de xarxes neuronals. Alguns dels exemples sobre els quals hem treballat per a la realització d'aquest projecte són la TPU de Google o el Eyeriss.

L'objectiu d'aquest treball és avaluar l'acompliment i eficiència energètica d'aquests processadors variant diversos paràmetres arquitectònics. Per a això hem triat un simulador de Arrays Sistòlics conegut com SCALE-Sim master. Aquest simulador permet obtindre resultats de rendiment i eficiència per a una determinada configuració de processador.

Els resultats mostren que per a la majoria d'aplicacions estudiades, tenint en compte l'eficiència energètica, un array de 64x64 PE resulta adequat, encara que arrays rectangulars (no quadrats) poden resultar interessants per a millorar algunes aplicacions. D'altra banda, s'ha comprovat que l'ample de banda de memòria pot tindre un gran impacte sobre el temps d'execució, per la qual cosa és necessari que els models d'array sistòlic incorporen aquest paràmetre en el càlcul de les prestacions.

**Paraules clau:** Xarxes Neuronals, Xarxes Neuronals Convolucionals (CNN), Arrays Sistòlics, paràmetres arquitectònics, eficiència energètica, rendiment.

# Tabla de contenidos

# Índice de figuras

# Índice de tablas

# 1 Introducción

## 1.1 Conceptos técnicos

Antes de comenzar con la descripción del proyecto se deben explicar ciertos conceptos que van a ser utilizados posteriormente y que son de vital importancia. Así pues, comenzamos con la definición de red neuronal artificial, la cual se define como un conjunto de nodos, llamados neuronas artificiales, conectados entre sí con el fin de transmitirse información. Esta información se transmite por la red neuronal siendo sometida a diferentes operaciones produciendo así unos valores de salida. Las redes neuronales están formadas por una capa de entrada, una capa de salida y en la mayor parte de los casos por una capa intermedia u oculta en la que se realizan ciertos cálculos de los que hablaremos en capítulos posteriores, como aparece representado en el Figura 1. Tal y como se puede observar el nombre viene dado por su gran parecido con el funcionamiento de las neuronas del cerebro humano, las cuales se encargan de recibir, procesar y transmitir información a través de señales químicas y eléctricas. [1]

Una de las mayores ventajas de las redes neuronales artificiales es su capacidad para encontrar patrones con una gran complejidad, los cuales son prácticamente imposibles de identificar mediante heurísticas.

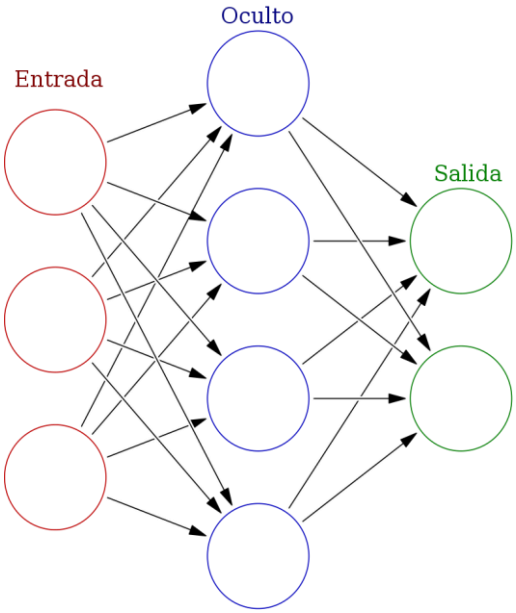



*Figura 1 Esquema de una red neuronal artificial. Fuente: [1]*

Al igual que las neuronas del cerebro humano las redes neuronales artificiales son capaces de aprender mediante entrenamiento y por ende son utilizados en gran medida en áreas dónde la programación convencional es más difícil de aplicar. Algunos de estos campos son la visión por computador, reconocimiento automático del habla o el reconocimiento de señales de audio y música, obteniendo en estos ámbitos grandes resultados. [2]

Hoy en día existen muchos tipos de redes neuronales artificiales y a continuación vamos a ver algunos de los tipos más utilizados [3]:

a) DNN → Redes neuronales profundas, conocidas en inglés como *Deep Neural Networks*.
b) PNN → Redes neuronales probabilísticas, conocidas en inglés como *Probabilistic Neural Network.*
*c)* RNN → Redes neuronales recurrentes, conocidas en inglés como *Recurrent Neural Network.*
d) CNN → Redes neuronales convolucionales, en las cuales se basa este proyecto.

Las CNN son un tipo de red neuronal artificial que presenta un especial parecido con las neuronas de la corteza visual primaria de un cerebro biológico. Asimismo, este tipo de redes neuronales son una variación de una red neuronal multicapa, también conocida como perceptrón multicapa. Este tipo de redes neuronales se caracteriza por que sus nodos intermedios están conectados con un subgrupo especializado de las capas siguientes, consiguiendo así reducir el número de neuronas necesarias y la complejidad computacional para poder ejecutarla, tal y como se puede observar en la Figura 2. [4]

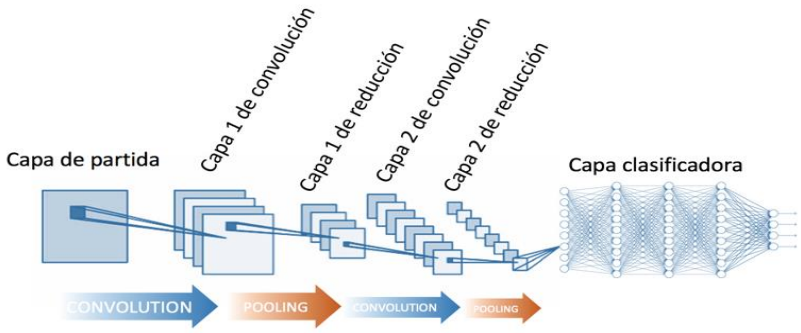



*Figura 2 Ejemplo esquemático de una red neuronal convolucional CNN. Fuente: [3]*

Este tipo de red neuronal se utiliza para el reconocimiento de visión artificial, como por ejemplo en la clasificación y segmentación de imágenes como ya hemos comentado anteriormente.

Con el paso del tiempo el desarrollo y la evolución de las redes neuronales ha provocado que su carga y complejidad aumenten de forma considerable, convirtiendo a los sistemas

convencionales inadecuados para su ejecución, por lo que se requieren nuevos sistemas que sean capaces de llevar a cabo sus requisitos de cómputo de forma eficiente.

Ante esta necesidad surge un nuevo tipo de procesadores cuyo propósito principal es el de procesar redes neuronales, estos procesadores se conocen como arrays sistólicos. Un array sistólico es una red homogénea de unidades de procesamiento de datos fuertemente acopladas llamadas nodos. Cada nodo calcula independientemente un resultado parcial, almacena el resultado y lo transmite hacia abajo. La arquitectura de los arrays sistólicos se explica de forma más detallada en la sección 2 Conceptos básicos. [7]

## 1.2 Motivación

Las redes neuronales no son una idea nueva. Datan de los años 40 y 50, cuando se empezaron a publicar los primeros diseños, conocidos como perceptrón. Sin embargo, nunca consiguieron atraer la atención de los investigadores, hasta que a finales de los 80 se desarrolló la técnica *backpropagation* o retropropagación, la cual permite retroceder a lo largo de una ruta individual de una red neuronal para revisar si las predicciones del modelo son correctas o no comparándolas con el resultado real, ajustando dichas rutas durante el proceso.

A causa de las novedades que han surgido en las últimas décadas las redes neuronales están volviendo a ser de actualidad. Por ejemplo, Google ha logrado derrotar a su propio reCAPTCHA con redes neuronales. En Stanford han conseguido generar pies de fotos automáticamente- Street View es una red neuronal convolucional que logra una precisión del 96% a la hora de reconocer números de calle en las imágenes que toman sus coches. Las redes neuronales se usan para mejorar el reconocimiento de voz de Android o para ahorrar electricidad en los centros de datos de Google. Todos estos avances nos acercan cada vez más a la idea de reproducir el funcionamiento del cerebro humano en un ordenador.

Para que estos avances puedan seguir siendo posibles, es necesario el diseño de arrays sistólicos que puedan realizar la inferencia cumpliendo con unos determinados requisitos de prestaciones y consumo energético. Para llegar a diseños adecuados, deben realizarse estudios sobre el impacto de los diversos parámetros de la arquitectura de estos arrays en el cumplimiento de los mencionados requisitos. La motivación del presente Trabajo de Fin de Grado es el desarrollo de un estudio del impacto de parámetros clave como la anchura y altura del array, o el ancho de banda de memoria principal, que permita extraer conclusiones útiles tanto para la

industria como la academia sobre el diseño de arrays sistólicos orientados a la inferencia en las redes neuronales.

## 1.3 Objetivos

El objetivo principal de este proyecto es comprender el impacto de diversos parámetros de la arquitectura de un array sistólico en las prestaciones y consumo energético de la inferencia en redes neuronales convolucionales.

Este objetivo se desarrolla en los siguientes subobjetivos:

a) Obtención de una visión general de las redes neuronales artificiales y sus conceptos básicos.
b) Comprensión de un simulador de redes neuronales mediante la realización de ejecuciones básicas que permitan entender su funcionamiento.
c) En tercer lugar, establecer los parámetros de configuración de acuerdo con los flujos de datos en los que se centra el presente TFG.
d) Análisis del impacto sobre el rendimiento de algunos parámetros arquitectónicos, más concretamente, de los que se refieren al tamaño del array y el subsistema de memoria. Este análisis se realizará desde el punto de vista del rendimiento, desde el punto de vista energético y desde el punto de vista del ancho de banda requerido por el sistema.

## 1.4 Impacto esperado

El presente TFG establece una base para la investigación en la arquitectura de los arrays sistólicos dentro del departamento DISCA, donde se ha desarrollado el proyecto.

Por otro lado, el proyecto pretende ser de ayuda para introducir el funcionamiento de la inferencia en las redes neuronales convolucionales en los arrays sistólicos actuales.

Además, para los diseñadores de este tipo de sistemas, resultarán de interés las conclusiones del análisis del impacto de los parámetros arquitectónicos estudiados.

Finalmente, este trabajo también introduce al simulador de redes neuronales convolucionales SCALE-Sim, el cual se presenta como una herramienta muy útil para el estudio y prototipado de posibles diseños de arrays sistólicos.

## 1.5 Metodología

La metodología empleada durante la realización del proyecto ha sido la siguiente:

a) Instalación del sistema operativo Linux, Python 3 y los módulos PIP necesarios para la ejecución del simulador tal y como se indica en su documentación.
b) Análisis del código fuente del simulador y realización de ejecuciones básicas con los parámetros por defecto para comprobar su correcto funcionamiento.
c) Estudio de los resultados obtenidos por el simulador con el fin de comprender la relación entre los parámetros arquitectónicos y la implementación del simulador con la intención de averiguar cómo afectan dichos parámetros a los resultados finales.
d) Realización de scripts que automaticen el proceso de ejecución de las diferentes configuraciones de tamaño de array y flujo de datos y la extracción de los resultados para su análisis.
e) Realización de cambios en los archivos de configuración variando los parámetros arquitectónicos a estudiar.
f) Análisis y estudio de los resultados. Elaboración de gráficos sobre rendimiento, consumo energético y ancho de banda para extraer las conclusiones sobre las configuraciones de arquitectura analizadas.

## 1.6 Estructura del documento

La estructura del documento es la siguiente:

1. **Introducción**: Se expone una primera toma de contacto con el contexto del trabajo, está formada por la motivación y los objetivos a cumplir. Al mismo tiempo comentamos la repercusión de este trabajo además de la metodología empleada y las colaboraciones con otros proyectos.
2. **Conceptos básicos y estado del arte:** Breve estudio sobre los conceptos sobre los que se basa el proyecto y el estado del arte de las redes neuronales artificiales.

3. **Estudio del simulador**: Se detalla en profundidad las peculiaridades y el funcionamiento del simulador SCALE-Sim.

4. **Resultados experimentales**: Se exponen y analizan los resultados que se obtienen de las configuraciones por defecto que incluye el simulador.

5. **Conclusiones**: Contiene las conclusiones que se obtienen a partir del estudio.

6. **Bibliografía**: Contiene la bibliografía empleada durante el proyecto.

## 1.7 Colaboraciones

La realización de este proyecto ha sido de forma conjunta con Eduardo Yago Vicent. Mientras que este trabajo se titula **"Prestaciones y energía de esquemas de mapeo con precarga para aceleradores CNN sistólicos"**, es decir se centra en los flujos de datos de Entrada Estacionaria y Pesos Estacionarios, mi compañero Eduardo se centra en el flujo de datos de Salida Estacionaria.

Ambos proyectos siguen la misma metodología, por lo que pueden aparecer ciertas similitudes entre ellos. Sin embargo, la diferencia entre los flujos de datos estudiados causa grandes diferencias en el trabajo realizado. El código que implementa el simulador para los diversos flujos de datos es muy diferente, así como el impacto de los parámetros arquitectónicos, lo cual afecta al análisis y las conclusiones. Es por ello que para cumplir con los requisitos temporales de los TFG en la UPV se ha decidido separar el trabajo en dos proyectos. Dado que la comparación entre ambos proyectos es inevitable, se harán referencias al otro proyecto a lo largo de este trabajo.

# 2 Conceptos básicos y estado del arte

En esta sección comentaremos los conceptos técnicos cuya compresión es necesaria para el correcto entendimiento del proyecto. Además, hablaremos sobre el estado del arte de las redes neuronales artificiales.

## 2.1 Arrays sistólicos

Debido a que los procesadores de propósito general, como las CPU, no son óptimos para ciertas aplicaciones especiales de alto rendimiento surgen los arrays sistólicos. La ejecución de aplicaciones en este tipo de procesadores se basa en identificar la función matemática clave para las prestaciones de la aplicación, y descomponer esta función en una secuencia de operaciones simples, tales como multiplicaciones y sumas. La secuencia se ejecutará sobre hardware especializado capaz de realizar una gran cantidad de operaciones simples en paralelo.

El aprendizaje profundo implica una serie de cálculos: convoluciones, multiplicaciones matriz-matriz (M-M), multiplicaciones matriz-vector (M-V), aplicación de no linealidades, cálculo de funciones de pérdida, actualizaciones de peso, agrupación máxima, etc. Como lo demostró William Dally en su tutorial de NIPS 2015 [14], las multiplicaciones M-M, M-V y las convoluciones son, con mucho, las operaciones más complejas y, por lo tanto, el objetivo principal para la optimización mediante arrays sistólicos. Afortunadamente, estas operaciones consisten en una serie de operaciones repetidas de multiplicación y acumulación (suma) múltiple. Por otro lado, las convoluciones se pueden traducir a una multiplicación matricial. En conclusión, centrarse en la optimización de la multiplicación matricial es sin duda lo más interesante.

Un array sistólico consiste en un conjunto de nodos interconectados también llamados "elementos de procesamiento" (*Processing Element* en inglés, más conocido por sus siglas *PE*), cada uno de los cuales es capaz de realizar una operación simple. La información puede fluir directamente entre los nodos de forma canalizada. Esto soluciona el problema de almacenar y cargar resultados intermedios. Las comunicaciones con el exterior del array se realizan sólo a través de las celdas fronterizas.

En resumen, cuando nos referimos a Arrays sistólicos, Matriz o Procesador sistólicos nos referimos a una red homogénea de unidades de procesamiento de datos fuertemente acopladas, comúnmente conocidos como nodos, tal y como observamos en la Figura 3.

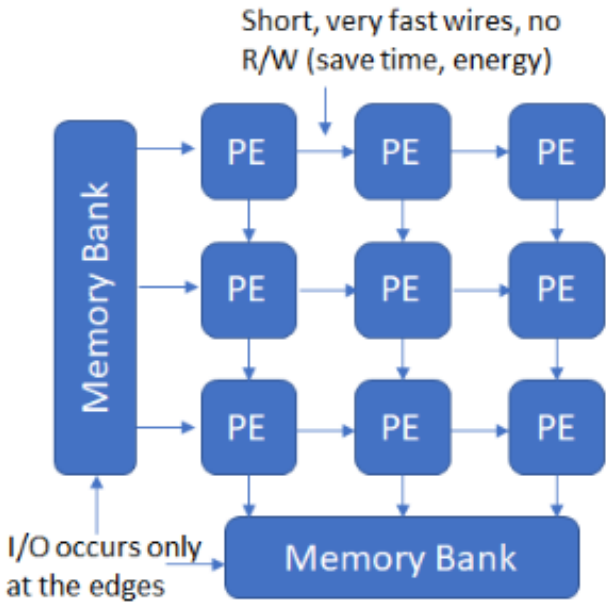



*Figura 3 Esquema simplificado sobre la arquitectura de un array sistólico.*

Cabe destacar que los PE solo pueden recibir datos desde su lado izquierdo o superior y solo puede transmitir datos hacia su lado derecho o inferior, tal y como se observa en la Figura 3.

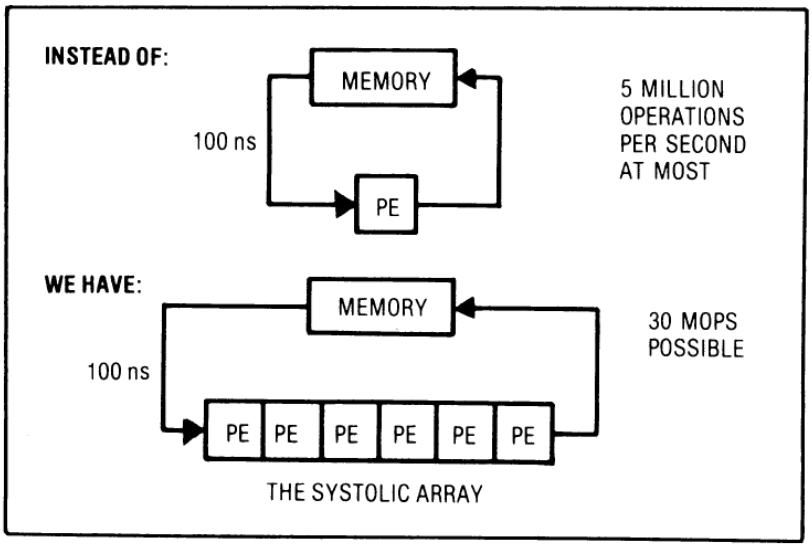



*Figura 4 Comparación a grandes rasgos entre un procesador común vs array sistólico.*

En la Figura 4 se puede observar a grandes rasgos una comparación esquemática de un procesador al uso y un array sistólico, dónde se observa claramente la diferencia entre ellos a la hora de almacenar y transmitir el resultado del cálculo de una operación. Con esta nueva arquitectura se consigue un grado mucho mayor de paralelismo que depende directamente del número de PE del array sistólico. Un aspecto importante de diseño es cómo se organiza la transmisión de los datos entre los diferentes PE's. Este aspecto se trata en la sección 3.2.1, donde se habla de ello de forma más extensa.

## 2.2 Introducción a los flujos de datos

El concepto de Flujo de datos está muy relacionado con el concepto de array sistólico y juega un papel esencial en los resultados obtenidos de la ejecución las diferentes cargas.

El Deep Learning utilizando CNN ha logrado una precisión sin precedentes en muchas aplicaciones modernas de IA. Sin embargo, el estado actual de las CNN requiere una increíble cantidad de movimiento de datos, la cual puede consumir más energía que el propio cálculo, por lo que el procesamiento de CNN no sólo debe proporcionar un alto paralelismo y rendimiento, sino que también debe optimizar el movimiento de datos de todo el sistema para lograr una alta eficiencia energética. Para abordar estos desafíos, es crucial diseñar un esquema de cómputo, llamado flujo de datos, el cual determina cuándo son introducidos los datos de entrada, cuándo se generan los datos de salida y cómo son almacenados y enviados los datos intermedios generados. En resumen, el concepto de flujo de datos se define como la política seguida para el manejo de los datos por el acelerador CNN.[7][8]

### 2.2.1 Flujos de datos y concepto de "precarga"

A grandes rasgos existen dos tipos de flujos de datos, los que no realizan una precarga de los datos como Salida Estacionaria (conocido en inglés por sus siglas *OS, Output Stationary* sobre los cuales trata el proyecto realizado paralelamente a este "Prestaciones y energía de esquemas de mapeo sin precarga para aceleradores CNN sistólicos"), tal y como se ha comentado en la sección 1.7 y los flujos de datos que sí que incluyen una precarga de los datos. En este trabajo nos centraremos en dos flujos de datos con precarga: Pesos Estacionarios (WS o *Weight Stationary*) y Entrada Estacionaria (IS o *Input Stationary*).

El cómputo de una red neuronal se basa en 3 matrices.

1. Se refiere a la matriz de entrada (*IFMAP*) y contiene todos los datos de entrada.
2. Se refiere matriz se refiere a la de pesos o filtros (*FILTER/WEIGHT*), que contiene los pesos de las conexiones entre las neuronas.
3. Se refiere a la matriz de salida (*OFMAP*), que contiene los datos que se generan al multiplicar la matriz de entrada con la matriz de pesos.

Se ha introducido un Anexo 1 dónde se explica el cómputo de una red neuronal de forma más detallada.

Antes de finalizar esta sección debemos hablar sobre el concepto de precarga. La precarga es el proceso por el cual se introduce una serie de datos, que van a ser utilizados durante la ejecución, antes de comenzar la misma. A grandes rasgos la forma de actuar es elegir una de las dos matrices IFMAP o FILTER/WEIGHT y almacenarla en el conjunto de PE de forma que cada componente de la matriz se asigna a un nodo del array. En el caso de elegir la matriz FILTER/WEIGHT el flujo de datos se denomina Pesos Estacionarios, mientras que si se precarga la matriz IFMAP el flujo es Entradas Estacionarias. Puesto que el número de PE en un array sistólico es mucho menor que el de ambas matrices, es necesario realizar precargas varias veces para poder ejecutar completamente el cómputo de una CNN.

## 2.3 Estado del arte

En esta sección se ha realizado un estudio del estado actual de las redes neuronales. Además, se han buscado TFGS relacionados en la base de datos de la ETSINF.

### 2.3.1 Redes neuronales

Hoy en día existe una gran cantidad de investigación internacional en el área de las redes neuronales artificiales. En general, cada grupo de investigación tiene motivaciones diferentes con respecto a la aplicación de este tipo de redes debido a son aplicables en un sinfín de ámbitos, algunos de estos grupos pueden ser de un carácter más técnico como programadores, físicos o matemáticos, pero también los hay de rasgos muy distintos como psicólogos, neurólogos o economistas.

Algunos de estos científicos expertos en redes neuronales artificiales están convencidos de que en un futuro no muy lejano la inteligencia artificial podría alcanzar al ser humano. Algunas de estas teorías afirman podría ocurrir en un par de décadas aproximadamente. Así lo afirmó José Cordeiro investigador de Singularity University en Sillicon Valley, “es posible que suceda entre el 2029 y el 2045”. [9]

Hoy en día disponemos una gran cantidad de máquinas inteligentes que son capaces de realizar actividades propias de humanos como por ejemplo ASIMO, el famoso robot desarrollado por Honda que es capaz de realizar una gran cantidad de actividades, tales como caminar, correr, obedecer órdenes etc. Este robot está dotado también con una serie de sensores con los que es capaz de reconocer el ambiente en el que esta para evitar choques o caídas. [9] Por otro lado, también es posible observar como otras de las famosas aplicaciones de las redes neuronales artificiales se ha estado utilizando para desarrollar aplicaciones como Google Goggles. Esta aplicación se basa en el reconocimiento de imágenes mediante el contenido de las mismas, al contrario que los buscadores tradicionales los cuales basan sus búsquedas en el nombre de la imagen, el nombre del enlace o el texto que aparece en la página donde se encuentra la imagen. [10]

Algunas aplicaciones de redes neuronales actuales o que se prevén lo sean un futuro cercano [12] son:

i. Computación afectiva: Robots capaces de ver, sentir, oler y percibir el entorno que los rodea.
ii. Predicción de valores e implementación de vehículos que sean capaces de conducir de forma autónoma.
iii. Composición de música y obras literarias.
iv. Comprensión de la información del genoma humano.
v. Autodiagnóstico medico por medio de redes neuronales.

### 2.3.2 TFGS Relacionados en la ETSINF

Al realizar una búsqueda de trabajos relacionados con arrays sistólicos o procesadores sistólicos, aceleradores de CNN o el impacto de diferentes dataflows sobre los arrays sistólicos en la base de datos de trabajos la Escuela de Ingeniería Informática nos damos cuenta de que no existe nada similar. Así pues, este trabajo y su variante realizada por mi compañero Eduardo Yago Vicent **"Prestaciones y energía de esquemas de mapeo sin precarga para aceleradores CNN sistólicos"** abren una nueva etapa en el desarrollo de este tipo de tecnologías, aportando así una vía de investigación nueva para la escuela.

Puesto que no disponemos de un array sistólico para realización de este trabajo hemos tenido que utilizar el simulador llamado SCALE-Sim del cual hablaremos en profundidad en la sección 5 Simulador SCALE-Sim. Sin embargo, hay que mencionar que tras realizar un trabajo previo con el simulador hemos detectado que no tiene en cuenta el consumo energético de los buffers ni el ancho de banda de memoria principal, lo cual afecta de manera significativa a las prestaciones. En este sentido, el presente trabajo realiza una evaluación más precisa del impacto de los parámetros arquitectónicos sobre las prestaciones y el consumo energético.

# 3 Simulador SCALE-Sim

En esta sección hablaremos sobre el simulador utilizado para obtener los datos que analizaremos posteriormente, más concretamente estamos hablando del Systolic CNN AcceLErator Simulator (SCALE Sim).

A grandes rasgos SCALE-Sim se puede definir como un simulador que permite emular el funcionamiento de un acelerador de CNN, permitiendo variar los parámetros de configuración de la arquitectura del mismo.

Existen ciertas características de este simulador que hay que destacar antes de continuar. SCALE-Sim no tiene en cuenta el ancho de banda de la memoria del sistema que está simulando, sino que cuando obtiene los resultados proporciona el ancho de banda necesario para realizar esa ejecución con los parámetros arquitectónicos proporcionados previamente. Tampoco tiene en cuenta la frecuencia de reloj a la que trabaja el acelerador. A causa de lo comentado anteriormente en secciones posteriores observaremos las diferencias entre el tiempo de ejecución proporcionado por el simulador y el tiempo de ejecución teniendo en cuenta las limitaciones hardware de un sistema real.

## 3.1 Descripción

SCALE Sim es un simulador que permite la investigación sobre la arquitectura del acelerador CNN además de poder ser utilizado para realizar estudios a nivel de sistema. Ha sido diseñado en base al array sistólico de la TPU de Google. Para su funcionamiento requiere de una topología de una red neuronal convolucional y ciertos parámetros arquitectónicos. Este simulador nos proporciona el tiempo de ejecución en ciclos, el porcentaje de utilización de los PE, los requerimientos de memoria y la cantidad de ancho de banda necesario para poder llevar a cabo la ejecución. Además, combinado con otros simuladores, como por ejemplo CACTI, también permite el cálculo de la energía consumida y el área estimada.

Existen dos puntos de vista desde los cuales se puede enforcar SCALE-Sim. En esta sección se discute SCALE-Sim desde el punto de vista de la realización de estudios a nivel de sistema.

Normalmente en aplicaciones de Machine Learning las CNN solo serían una parte del pipeline. Por lo tanto, es importante entender las características de la aplicación en el contexto de todo el SoC (System-on-Chip). Hoy en día existen simuladores de todo el sistema, como pueden ser Gem5 o GemDroid, los cuales carecen de módulos para aceleradores CNN lo que presenta un obstáculo en el estudio del comportamiento a nivel de sistema de las aplicaciones habilitadas para Machine Learning. Gracias a su diseño modular SCALE-Sim puede ser integrado con Gem5 o GemDroid para una simulación total del sistema. Esto resulta especialmente útil para los investigadores que no pretendan realizar una investigación exhaustiva sobre la microarquitectura del acelerador CNN, pero sin embargo deseen integrar un acelerador de CNN para obtener resultados significativos de caracterización a nivel de sistema.

En nuestro caso vamos a centrar nuestra atención sobre el estudio del impacto de la arquitectura sobre el acelerador CNN, puesto que es la opción del simulador que más nos interesa para nuestro proyecto.

La realización de un estudio del impacto de la arquitectura sobre el acelerador CNN permite al investigador, mediante el uso de SCALE-Sim, explorar rápidamente el espacio de diseño del acelerador. SCALE-Sim requiere que el usuario le proporcione una lista de límites para las características de la arquitectura como el tamaño del array sistólico, el tamaño de la SRAM, es decir el tamaño de las matrices (IFMAP, FILTER y OFMAP), el flujo de datos etc. Cabe destacar que en el caso de que no se especifiquen los límites comentados anteriormente SCALE-Sim asumirá los valores predeterminados. En la Figura 5 se puede observar de forma detallada el acelerador de CNN que se modela en SCALE-Sim.

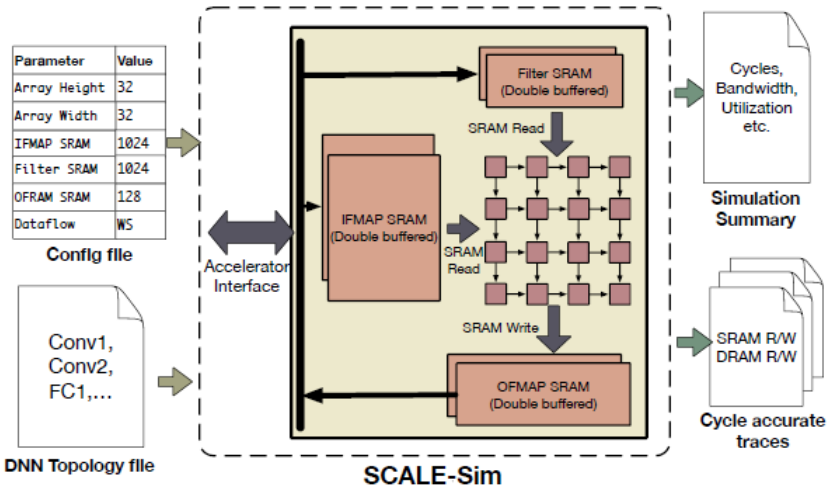



*Figura 5 Array sistólico simulado en SCALE-Sim esquema arquitectónico dónde se pueden ver además los elementos de entrada y de salida. Fuente: [11]*

De forma más concreta vamos a hablar sobre el array sistólico que ha sido modelado en el simulador SCALE-Sim, dado que es el simulador con el que se ha realizado este trabajo, más concretamente podemos visualizarlo en la Figura 6.

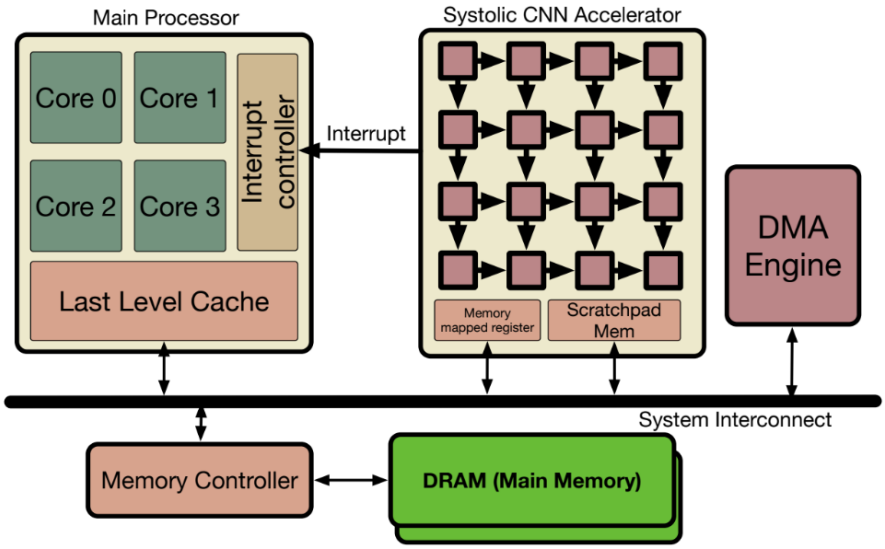



*Figura 6 Array sistólico modelado en SCALE-Sim integrado en un sistema real. Fuente: [11]*

Tal y como hemos discutido anteriormente un array sistólico está compuesto por PE's, los cuales están representados como pequeños cuadrados con flechas con las que se indica la dirección por la que los resultados son enviados a los siguientes PE.

Como muestra la Figura 6 en la que se representa el acelerador CNN dentro de un sistema real, se observa que un array sistólico cuenta con dos memorias, *Memory mapped register* y *Scratchpad Mem*.

Antes de hablar sobre la Memory mapped register cabe destacar el contexto en el que se encuentra el acelerador. Tal y como se observa en la Figura 6 éste se encuentra en un sistema dónde él no es el procesador principal, si no que su función es la de recibir una red neuronal a modo de tarea y realizar el cómputo de ésta para posteriormente guardar el resultado en la memoria y enviar una interrupción al procesador principal para que sepa que el cálculo ha finalizado. Mientras realiza el cálculo de la red neuronal el procesador puede seguir ejecutando otras tareas.

Una vez comprendido el papel que desempeña el acelerador dentro del sistema es mucho más sencillo entender que la función de la "Memory mapped register" es la de almacenar las descripciones de las tareas enviadas por el procesador principal al acelerador.

Por otro lado, la memoria "Scratchpad Mem" se caracteriza por ser una memoria SRAM de doble buffer, la cual esta subdividida en 3 partes las cuales constan de puertos de lectura/escritura independientes. Así pues, la primera partición es la correspondiente al almacenamiento de datos IFMAP, en la siguiente se almacenan los datos de FILTERS/WEIGHTS y finalmente tenemos la partición de OFMAP.

### 3.2.1 Flujos de datos utilizados en SCALE-Sim

Vamos a continuar con la definición de los flujos de datos que van a ser estudiados en este trabajo, más concretamente son aquellos que cuentan con una precarga de los datos:

En primer lugar, tenemos Weight Stationary o Peso Estacionario, el cual se define como el mapeado por el cual cada elemento de la matriz de filtros/pesos se mapea únicamente a un PE. Una vez se han mapeado los pesos en la matriz de entrada y se han realizado todos los cálculos pertinentes con los mismos se procede a sustituir los datos mapeados inicialmente por los siguientes para seguir realizando el cálculo de la CNN.

El funcionamiento es el siguiente, en cada ciclo los elementos de entrada son multiplicados con los pesos que han sido precargados y transmitidos a la matriz, y tras obtener los resultados/sumas parciales estos son almacenados en la propia matriz. El proceso de reducción se lleva a cabo mediante la transmisión de los resultados/sumas parciales a través de los PE, proceso que se lleva a cabo durante varios ciclos. Tal y como se observa en la figura 7 el proceso de mapear los datos se lleva a cabo en dos etapas. En primer lugar, se asigna un filtro para cada columna. Este proceso de asignación se inicia desde la parte superior de la matriz y finaliza cuando todos los PE tienen su elemento dado.

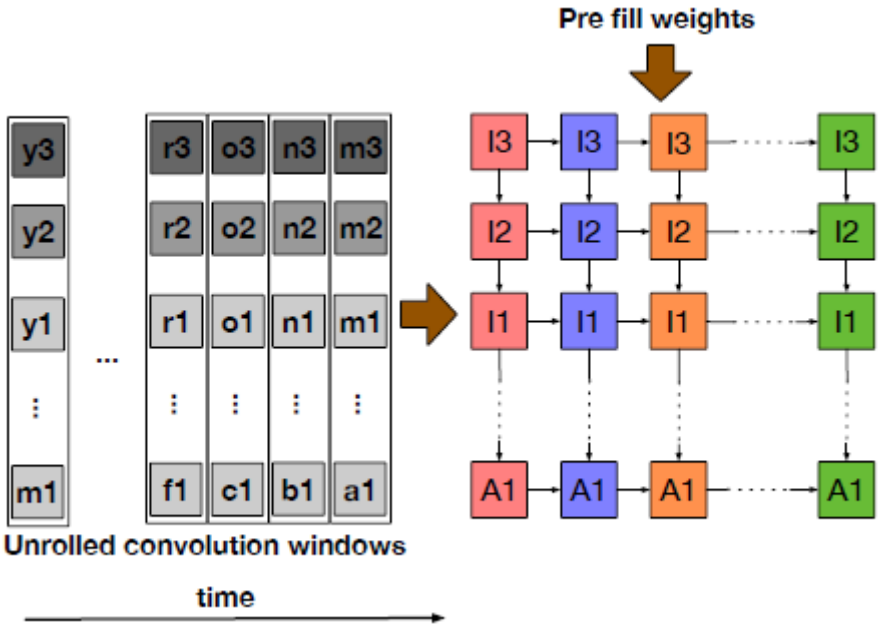



*Figura 7 Explicación esquemática del flujo de datos WS. Fuente: [11]*

Una vez se han asignado los pesos en la matriz de filtros, se introducen los elementos de entrada desde el borde izquierdo, como también se puede apreciar en la imagen. Durante esta fase las sumas parciales para un píxel de salida dado se generan en cada ciclo. Los pixeles de salida obtenidos provienen de la reducción de las sumas parciales de la misma columna. Estos pixeles de salida se obtienen en los próximos N ciclos, siendo N el número de sumas parciales generadas para cada píxel de salida. Los pesos se mantienen en la matriz de filtros hasta que todos los cálculos que requieran estos datos hayan terminado, una vez haya ocurrido esto se sustituyen por un nuevo conjunto de estos.

Cabe destacar que debido a la naturaleza secuencial de este tipo de mapeo de datos primero se realiza la asignación de pesos y posteriormente se introducen las entradas. En consecuencia, se necesita un número menor de bancos de SRAM en comparación con otro tipo de mapeos como Salida estacionaria (Output Stationary). Sin embargo, el tener que mantener las sumas parciales, correspondientes a múltiples pixeles de salida, en la matriz hasta que se reduzcan provoca un aumento en el coste de implementación.

En segundo lugar, tenemos Input Stationary o Entrada estacionaria, el cual es muy similar al dataflow comentado anteriormente. Esta gran similitud se debe a que ahora son los elementos de entrada los que están mapeados a cada uno de los PE de la matriz de filtros y los elementos de entrada transmitidos son los pesos.

Similarmente a lo que ocurre en WS el mapeo se lleva a cabo en dos etapas. Pero antes de explicar cómo funciona hay que definir que es una ventana convolucional. Una ventana convolucional se define como el conjunto de todos los pixeles de IFMAP que se necesitan para generar un solo píxel de salida/OFMAP.

Así pues, en este tipo de dataflow cada columna es asignada a una ventana convolucional. Tal y como ocurre con WS, para cada columna los pixeles correspondientes a una ventana convolucional se introducen por la parte superior de la matriz. Una vez han sido introducidos los datos de entrada, los elementos de la matriz de pesos son introducidos por la parte izquierda de la matriz.

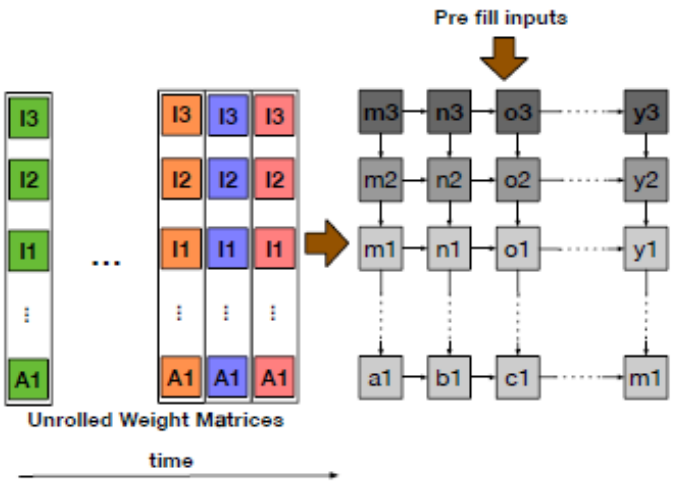



*Figura 8 Explicación esquemática del flujo de datos IS. Fuente: [11]*

De manera similar a lo que ocurre en WS, la reducción se realiza para cada columna y las ventanas convolucionales se mantienen hasta que se realicen todos los cálculos necesarios, para posteriormente ser reemplazados por elementos pertenecientes a nuevas ventanas convolucionales.

Como se puede deducir este flujo de datos también obtiene beneficios en cuanto a la cantidad de bancos de SRAM necesarios comparándolo con otros dataflows como Salida Estacionaria (OS). Al comparar el coste y el tiempo de ejecución entre WS e IS observaremos como varían en función de la carga de trabajo.

En la siguiente sección hablaremos a grandes rasgos de cada uno de los diferentes archivos utilizados por el simulador para obtener los resultados y de los archivos que el simulador nos proporciona tras la ejecución del mismo.

En cuanto a los ficheros de entrada del simulador tenemos el archivo de configuración, cuyo formato es ".cfg", donde se establecen los parámetros arquitectónicos de la configuración que deseamos simular, y por otro lado el archivo donde está representada la red neuronal sobre la que se va a ejecutar dicha configuración. Estos archivos se comentarán en la sección 3.4.

Tras introducir una topología de una CNN y los parámetros arquitectónicos comentados anteriormente, el simulador nos proporcionará los resultados, de la misma forma en que podemos observar en la Figura 9:

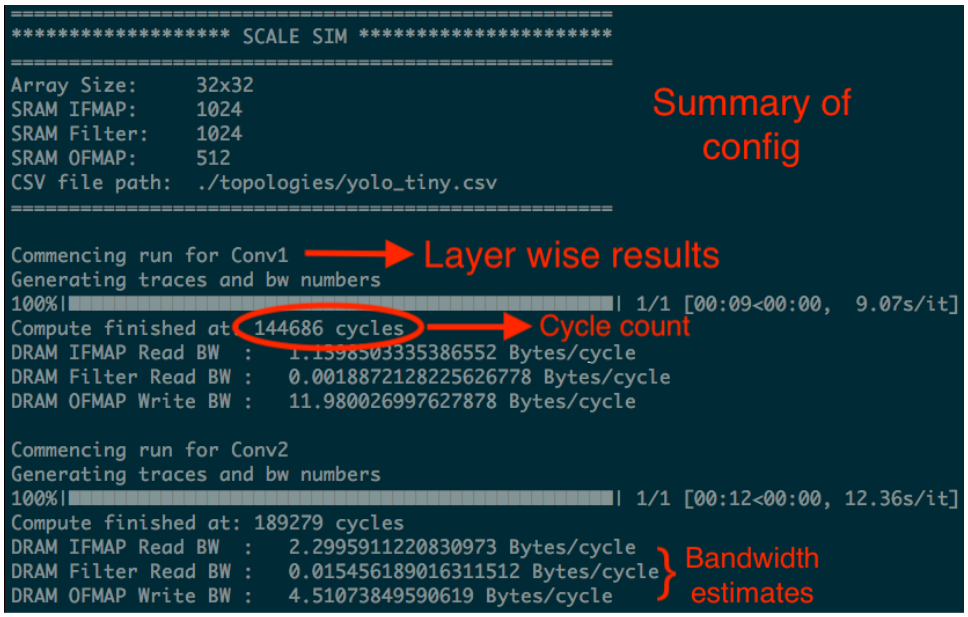
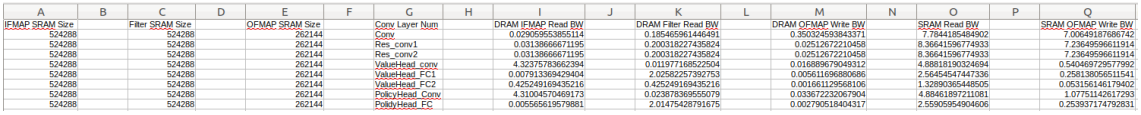



*Figura 9 Ejecución de la carga yolo_tiny.csv y sus resultados en la terminal. Fuente: [11]*

Tras la ejecución los resultados obtenidos por el simulador se almacenan en la carpeta "`outputs`". Una vez dentro de esta carpeta podemos observar los diferentes archivos generados. Al analizar los archivos de la ejecución hay que distinguir entre dos tipos.

En primer lugar, tenemos los archivos resumen generados por las trazas de lectura y escritura entre los cuales se encuentran en el fichero "`avg_bw`" donde se guardan los requerimientos medios de ancho de banda tanto para escritura como para lectura de las diferentes memorias, es

decir tanto DRAM (con sus requerimientos independientes para cada una de las tres matrices) como SRAM. En la figura 10 podemos observar una captura de dicho archivo.



| IFMAP SRAM Size | Filter SRAM Size | OFMAP SRAM Size | Conv Layer Num | DRAM IFMAP Read BW | DRAM Filter Read BW | DRAM OFMAP Write BW | SRAM Read BW | SRAM OFMAP Write BW |
|---|---|---|---|---|---|---|---|---|
| 524288 | 524288 | 262144 | Conv | 0.029059653855114 | 0.18546596144649 | 0.35032493843371 | 7.784418548490 | 7.006491876867 |
| 524288 | 524288 | 262144 | Res_conv1 | 0.03138666671195 | 0.20031827435824 | 0.02512672210458 | 8.366415967749 | 7.236495966119 |
| 524288 | 524288 | 262144 | Res_conv2 | 0.03138666671195 | 0.20031827435824 | 0.02512672210458 | 8.366415967749 | 7.236495966119 |
| 524288 | 524288 | 262144 | ValueHead_conv | 4.32375783662394 | 0.01197716852504 | 0.01688967904931 | 4.888181903246 | 0.540469729577992 |
| 524288 | 524288 | 262144 | ValueHead_FC1 | 0.00791336942940 | 2.02582257392753 | 0.00561169688068 | 2.564545474473 | 0.258138056511541 |
| 524288 | 524288 | 262144 | ValueHead_FC2 | 0.42524916943521 | 0.42524916943521 | 0.00166112956810 | 1.328903654485 | 0.053156146179402 |
| 524288 | 524288 | 262144 | PolicyHead_Conv | 4.31004570469173 | 0.02387836955507 | 0.03367223206790 | 4.884618972110 | 1.077511426172 |
| 524288 | 524288 | 262144 | PolicyHead_FC | 0.00556561957988 | 2.01475428791675 | 0.00279051840431 | 2.559059549046 | 0.253937174792831 |

*Figura 10 Archivo avg_bw.*

En segundo lugar, encontramos el archivo llamado "`cycles`" en el cual se encuentran el número de ciclos consumido por cada una de las capas de la red neuronal, así como el porcentaje de utilización de cada una de ellas.

En tercer lugar, encontramos el archivo llamado "`detail`" en el cual se encuentran las posiciones de memoria desde donde empiezan y acaban las lecturas y escrituras de DRAM y SRAM. Al mismo tiempo proporciona el tamaño de las diferentes matrices de la memoria DRAM (IFMAP, FILER y OFMAP) y de SRAM.

En cuarto lugar, encontramos finalmente el archivo "`max_bw`" dónde se recogen los anchos de banda necesarios para poder llevar a cabo la ejecución de cada una de las capas. En este último archivo cabe destacar que nos proporciona el ancho de banda máximo requerido para lectura y escritura de las memorias DRAM y SRAM que utilizaremos en las gráficas sobre el ancho de banda requerido en secciones posteriores.

Además de los archivos comentados anteriormente también nos encontramos con la carpeta "`layer_wise`" en la cual encontramos información más detallada sobre las trazas de lectura y escritura de las diferentes memorias. En cuanto a la memoria DRAM tenemos la información referente a las tres matrices de las que ya hemos hablado en la sección 2.2.1. En la misma carpeta también observamos los archivos asociados a la memoria SRAM tanto de lectura como de escritura. Por último, cabe destacar que estos cinco archivos se generan para cada una de las capas de la red neuronal que se ejecute.

## 3.4 Parámetros de la arquitectura

En la siguiente sección se comentan los parámetros básicos de la arquitectura implementada por el simulador. Así pues, para facilitar la compresión de los parámetros que van a ser utilizados observamos diferentes capturas extraídas directamente del simulador. Como se ha comentado en secciones anteriores para la ejecución de una simulación es necesario un archivo

de configuración y una topología que va a ser ejecutada en la configuración descrita anteriormente.

En la Figura 11 se observa una captura de un archivo de configuración, con los diferentes parámetros:

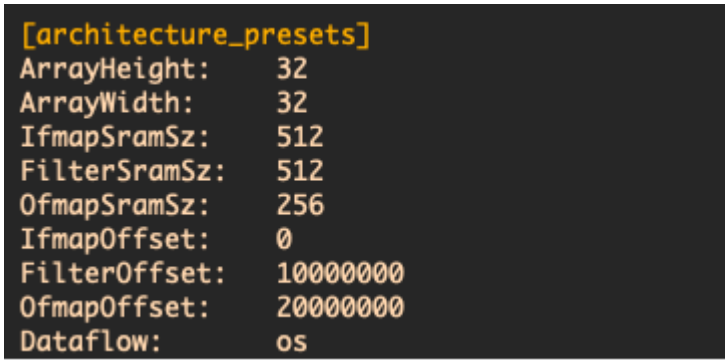



*Figura 11 Ejemplo de archivo de configuración con los parámetros arquitectónicos proporcionados al simulador.*

a) ArrayHeight → referencia al número de filas del array sistólico.
b) ArrayWidth→ referencia al número de columnas del array sistólico.
c) IfmapSramSz → referencia al tamaño del espacio de trabajo de la SRAM para IFMAP en KB.
d) FilterSramsSz→ referencia al tamaño del espacio de trabajo de la SRAM para FILTER en KB.
e) OfmapSramSz→ referencia al tamaño del espacio de trabajo de la SRAM para OFMAP en KB.
f) Ifmap/Filter/Ofmap Offset → desplazamiento para las direcciones generadas de cada una de las matrices. El valor 0 de la matriz de IFMAP se debe a que no se van a generar direcciones de los datos que se están leyendo.
g) Dataflow → en este parámetro es dónde se define el flujo de datos utilizado el cual puede ser os, ws o is.

## 3.5 Cargas soportadas

Hay que destacar que SCALE-Sim solo acepta redes neuronales convolucionales (CNN) o redes neuronales totalmente conectadas (conocidas en inglés como *fully connected*). El concepto fully connected significa que las capas por las que está formada la red neuronal están conectadas totalmente, siendo una capa totalmente conectada una que sigue la función $\mathbb{R}_m \rightarrow \mathbb{R}_n$.

| Layer name | IFMAP Height | IFMAP Width | Filter Height | Filter Width | Channels | Num Filter | Strides |
|---|---|---|---|---|---|---|---|
| Conv1 | 224 | 224 | 7 | 7 | 3 | 64 | 2 |
| Conv2red | 56 | 56 | 1 | 1 | 64 | 64 | 1 |
| Conv2 | 56 | 56 | 3 | 3 | 64 | 192 | 1 |
| Inc3a_1x1 | 28 | 28 | 1 | 1 | 192 | 64 | 1 |
| Inc3a_3x3red | 28 | 28 | 1 | 1 | 192 | 96 | 1 |
| Inc3a_3x3 | 28 | 28 | 3 | 3 | 96 | 128 | 1 |
| Inc3a_5x5red | 28 | 28 | 1 | 1 | 192 | 16 | 1 |
| Inc3a_5x5 | 28 | 28 | 5 | 5 | 16 | 32 | 1 |

*Figura 12 Ejemplo de archivo .csv donde se aprecian las diferentes capas de una red neuronal y sus parámetros*

Como vemos en la Figura 12 éste es el formato que debe seguir una red neuronal para poder ser simulada en SCALE-Sim. Analizando los parámetros de la misma observamos el nombre que se le va a dar la capa, la altura y anchura de las matrices de IFMAP y FILTER, seguidos por el número de canales, el número de matrices de filtros y el tamaño de paso o *stride*. El concepto de stride se analiza de forma más concisa en el Anexo 1.

Para las ejecuciones que se van a realizar en este trabajo se han escogido las mismas cargas que se utilizan en [11], y para una mayor facilidad a la hora del estudio de resultados se han asignado las siguientes etiquetas a cada una de las cargas tal y como se puede observar en la Tabla 1.

| Tag | Description |
|---|---|
| W1 | AlphaGoZero [18] |
| W2 | DeepSpeech2 [19] |
| W3 | FasterRCNN [20] |
| W4 | Neural Collaborative Filtering [21] |
| W5 | Resnet50 [17] |
| W6 | Sentimental CNN [22] |

*Tabla 1 Etiquetas utilizadas por las diferentes cargas. Fuente: [11]*

Cabe destacar que la carga W4 tiene un comportamiento bastante diferente al resto de las cargas tal y como veremos en la sección 6 Resultados base. Se puede observar dicha carga de forma más detallada en el Anexo 4 Cargas estudiadas.

Antes de terminar con esta sección quiero destacar que existen algunos errores en la carga W6. En primer lugar, cuando la descargas y la abres con un editor de texto te das cuenta de que la segunda línea está compuesta solo por comas, y esto genera un error en algunas configuraciones por lo que es recomendable eliminarla pues que no aporta nada, tal y como observamos en la Figura 13.

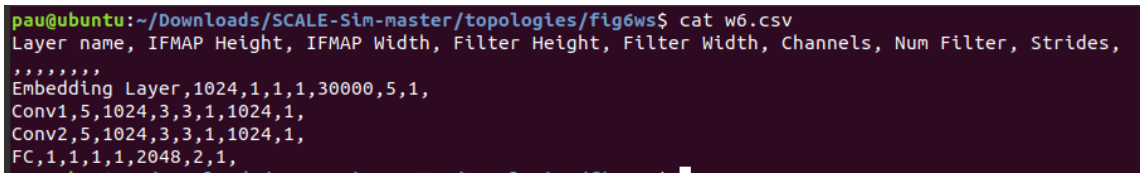



*Figura 13 W6.csv en su estado original*

Finalmente se observa en la Figura 14 como el nombre de la primera capa `Embedding Layer` no sigue el formato de nombramiento de las capas seguido en todas y cada una de las diferentes capas de cada una de las cargas, por lo que, para evitar posibles errores futuros, por ejemplo, cuando se utilicen scripts para automatizar el proceso, también hay que cambiar el nombre por `Embedding_Layer` siguiendo así el formato de las demás capas.

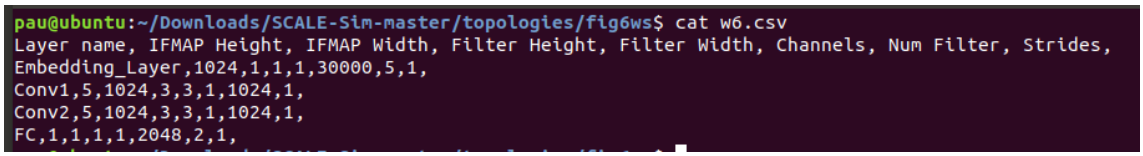



*Figura 14 W6.csv tras los cambios comentados en esta sección.*

## 3.6 Scripts utilizados

Como se ha podido observar para la obtención de resultados se necesita simular una gran cantidad de configuraciones diferentes sobre todas las cargas que van a ser estudiadas en este trabajo, las cuales hay que ejecutar en los dos flujos de datos seleccionados (WS e IS). El simulador realiza las ejecuciones de forma secuencial.

Con el fin de evitar que se pierda tiempo realizando las ejecuciones de forma manual se han creado dos scripts que automatizan dicho proceso.

Se han realizado dos scripts en bash que están centrados, uno en la creación de los archivos de configuración y otro en la ejecución de las diferentes configuraciones creadas anteriormente. Estos scripts deben ejecutarse siempre en la carpeta del simulador.

Para un correcto funcionamiento de los scripts comentados anteriormente hay que seguir una serie de pasos que van a ser descritos a continuación.

Ambos scripts comparten gran parte del código y están diseñados para recorrer las cargas que van a ser ejecutadas sobre los distintos tamaños de array sistólico y sobre los distintos flujos de datos. Para que el script sea capaz de recorrer cada una de las cargas se deben introducir dentro

de la carpeta `topologies` del simulador y se debe crear una carpeta llamada `fig6ws` en la cual deben estar las cargas desde W1 hasta W6.

Además de las cargas el simulador requiere al mismo tiempo de los archivos de configuración sobre los que va a ejecutar cada una de ellas, estos archivos se generan con el `script-configs.sh`. Antes de ejecutar dicho script deben crearse las carpetas correspondientes, dentro de la carpeta `configs`, donde van a guardarse dichas configuraciones. Estas carpetas son `fig6cfgs` y `fig6cfgsR` donde se guardarán respectivamente los archivos de configuración para las matrices cuadradas y los archivos de configuración para las matrices rectangulares. En cuanto al script encargado de generar los archivos de configuración cabe destacar que estos deben realizarse de forma separada para cada flujo de datos para que se generen de forma correcta.

Una vez se han realizado los pasos descritos anteriormente ya se puede ejecutar el `script-ejecucion.sh`. Es de gran ayuda utilizar el comando `--help` mediante el cual podremos observar de forma más detalla los parámetros que debemos introducir para su correcto funcionamiento.

Ambos scripts se pueden examinar de forma más detallada en el Anexo 2.

Además, para la sección de consumo energético también se ha realizado un script (`count.sh`, ver Anexo 3), mediante el cual se obtienen los datos que se muestran en la Figura 15.

```
Output: w6_8x8_ws
Total SRAM_IFMAP reads during execution: 17066112
Total SRAM_FILTER reads during execution: 20892528
Total SRAM_OFMAP write during execution: 31758848
```

*Figura 15 Ejemplo de salida del script count.sh*

Estos datos son necesarios para el cálculo de la energía consumida y se encuentran en los archivos correspondientes a las lecturas y escrituras de SRAM (para las matrices de IFMAP, FILTER y OFMAP) de cada una de las capas que forman cada una de las cargas que se han ejecutado.

# 4 Resultados experimentales

En esta sección comentaremos las gráficas obtenidas de los resultados de las ejecuciones de las cargas W1-W6. Estos resultados han sido obtenidos tras su ejecución en ambos flujos de datos, Pesos Estacionarios y Entradas Estacionarias.

## 4.1 Tiempos de ejecución

Vamos a empezar comentando los resultados obtenidos respecto al flujo de datos de Pesos Estacionarios y a continuación hablaremos sobre los resultados referentes al flujo de datos de Entradas Estacionarias.

### 4.1.1 Entrada Estacionaria

En primer lugar, vamos a observar detalladamente el gráfico de la Figura 16 en el cual se ha representado el tiempo de ejecución normalizado para cada una de las cargas sobre las diferentes configuraciones de tamaño del array sistólico.

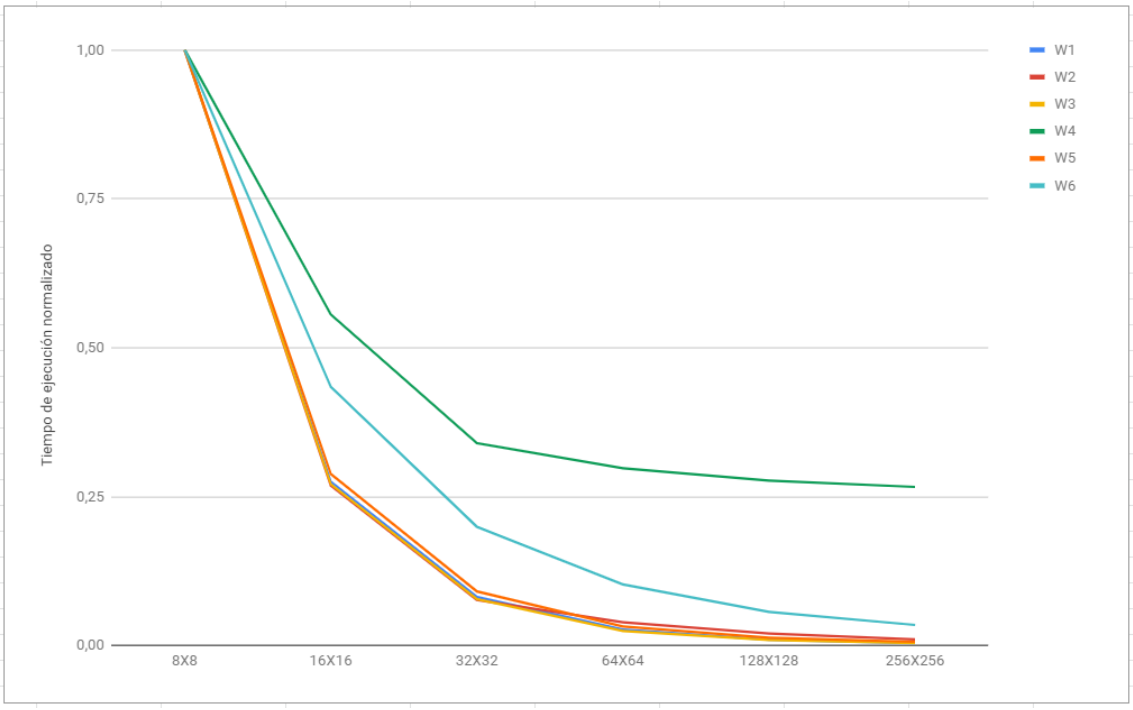



*Figura 16 Tiempo de ejecución normalizado para diferentes tamaños de array sistólico WS.*

En gráfico expuesto anteriormente se observa como al aumentar x4 el tamaño del array el tiempo de ejecución se reduce de forma significativa, pero con la intención de obtener una visión más concreta del porcentaje de mejora sobre el tamaño de array anterior se ha creado la Tabla 2:

| WS | W1 | W2 | W3 | W4 | W5 | W6 |
|---|---|---|---|---|---|---|
| 8X8 | | | | | | |
| 16X16 | 3,63 | 3,72 | 3,69 | 1,80 | 3,47 | 2,30 |
| 32X32 | 3,37 | 3,50 | 3,47 | 1,64 | 3,16 | 2,18 |
| 64X64 | 3,02 | 1,94 | 3,15 | 1,14 | 2,81 | 1,94 |
| 128X128 | 2,66 | 1,94 | 2,57 | 1,08 | 2,37 | 1,81 |
| 256X256 | 2,28 | 1,88 | 2,07 | 1,04 | 2,04 | 1,63 |

*Tabla 2 Mejora sobre el tiempo de ejecución de la configuración anterior WS.*

Siendo los valores que se encuentran en la tabla dos, el número de ciclos consumido por el tamaño de array anterior entre el número de ciclos consumidos por el tamaño de array actual. Así pues, observamos como existe una clara tendencia en la que se disminuye el tiempo de ejecución de manera significativa (alrededor de 3 veces más rápido) hasta que se produce un estancamiento. Este estancamiento se debe a que a partir de ciertos tamaños de array seguir aumentando su tamaño carece de sentido puesto que parte del mismo deja de ser utilizado tal y como veremos en la siguiente sección 6.2. Así pues, existen dos tamaños entre los cuales el porcentaje de mejora respecto al tamaño anterior sigue siendo interesante, estos tamaños son 32x32 y 64x64. A partir de este momento el aumento del tamaño del array deja de repercutir de forma tan considerable sobre el número de ciclos que necesita para finalizar el cómputo de la red neuronal, aunque existen aplicaciones, como W6, dónde tamaños superiores de array pueden tener un impacto significativo en las prestaciones. Es decir, cada aplicación tiene un comportamiento diferente debido a sus diferentes configuraciones, sin embargo, las conclusiones extraídas anteriormente son fácilmente extrapolables a la mayoría de ellas.

Cabe preguntarse por qué razón no se escoge el tamaño más grande, es decir 256x256, y esto se debe a que como observaremos en secciones posteriores el número de ciclos de cómputo de una carga no es el único factor que influye a la hora de escoger el mejor tamaño para un array sistólico. Otras de las variables a tener en cuenta son el consumo, el ancho de banda requerido y el porcentaje de utilización del array sistólico. Para una mayor profundización sobre la elección del tamaño del array sistólico se ha elaborado un gráfico del tipo EDP (Energy Delay Product) el cual se obtiene de multiplicar el número de ciclos que tarda en realizar la ejecución por la energía consumida para el tamaño de array modelado tal y como se muestra a continuación en la Figura 17. El resto de las variables, es decir el ancho de banda requerido y el porcentaje de utilización se abordarán en las próximas secciones.

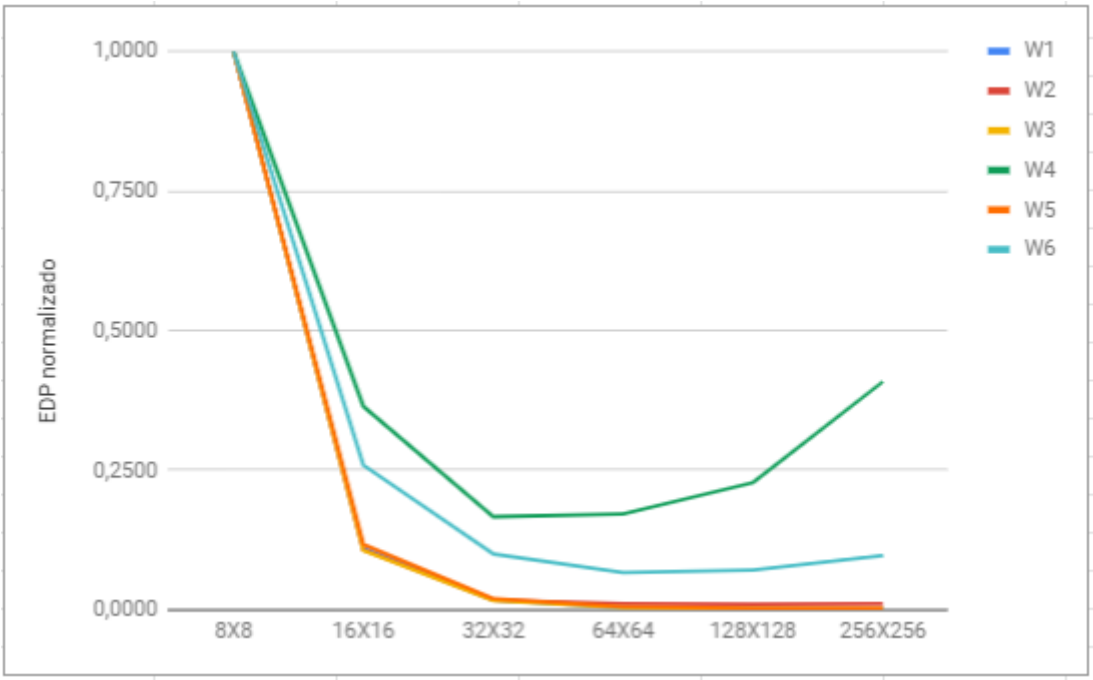



*Figura 17 EDP normalizado para diferentes configuraciones del array sistólico WS.*

En la Figura 17 se puede observar de forma mucho más clara que los tamaños de 32x32 y 64x64 son los que obtienen una mejor relación prestaciones/consumo para la mayoría de las cargas estudiadas, apoyando así las conclusiones extraídas anteriormente.

Todas las configuraciones que hemos estudiado hasta el momento han sido cuadradas, es decir el array sistólico tenía siempre la misma altura y anchura. A continuación, vamos a observar que ocurre cuando realizamos ejecuciones con arrays rectangulares. El uso de arrays rectangulares se justifica por los diferentes tamaños de las matrices que alimentan el array. Más concretamente vamos a analizar lo que ocurre cuando duplicamos la altura o la anchura del array partiendo de un array de 32x32 PE, y comparándolo con arrays de 32x64, 64x32, y 64x64 PE.

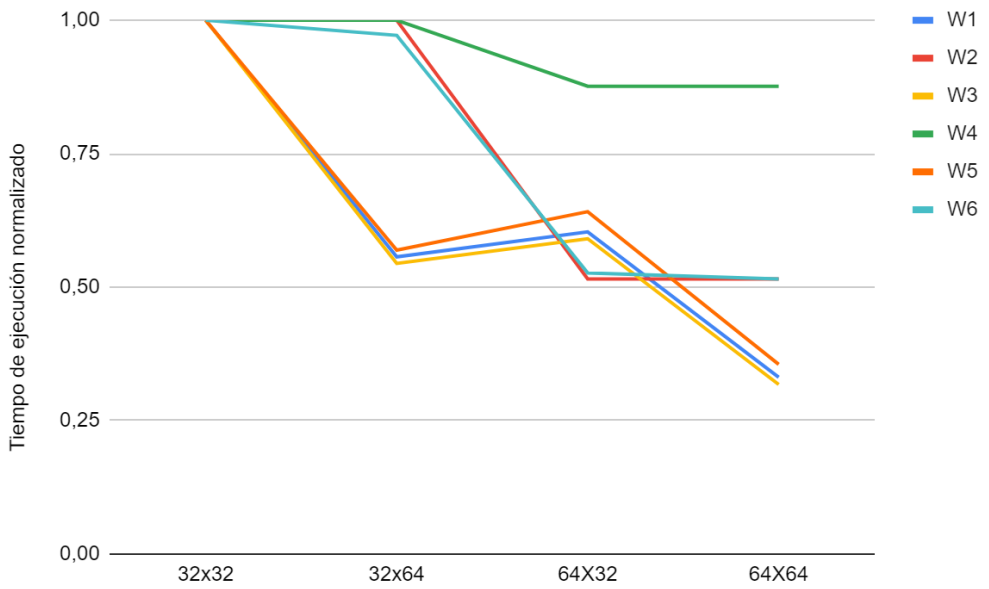



*Figura 18 Tiempo de ejecución normalizado para diferentes tamaños de array sistólico rectangulares WS.*

La Figura 18 presenta el tiempo de ejecución normalizado de las mencionadas configuraciones de array. Se observa como dependiendo de la carga a computar se obtienen resultados muy diferentes. Esto se debe a que cada una de las cargas tiene una configuración específica y diferente a las demás con el fin de que el estudio sobre las mismas sea lo más realista posible. La configuración de las cargas se presenta de forma detallada en el Anexo 4. Observamos como W2, W4 y W6 obtienen un mayor rendimiento para un tamaño de 64x32 sin embargo W3, W5 y W6 obtienen un mejor rendimiento en 32x64. La explicación para lo que acabamos de ver reside en el hecho de que las cargas que se ejecutan en el simulador están formadas por filas y columnas, estas filas y columnas se introducen en las matrices de IFMAP y FILTER respectivamente por lo que si una carga tiene un mayor número de filas que de columnas una configuración rectangular del tipo 64x32 provocará una disminución considerable en el tiempo de ejecución obtenido. Así pues, como se puede observar en el gráfico anterior observamos como las cargas con una gran cantidad de filas, es decir W2, W4 y W6, no son capaces de aprovecharse de un mayor tamaño de columnas para el flujo de datos de Pesos Estacionarios. Debido a lo comentado anteriormente sobre los resultados que estamos analizando no se puede obtener un tamaño de array rectangular que sea "general" para todas las cargas puesto que tanto si elegimos una configuración del tipo 32x64 o 64x32 siempre dependerá de la carga que vaya a ser ejecutada para mejorar o no el rendimiento.

Cabe destacar que en general la mayoría de las cargas tienen un tamaño de filas o IFMAP mucho mayor que el de columnas o FILTROS por lo que en el caso de tener que escoger una

configuración rectangular siempre se escogería una con mayor número de filas como por ejemplo 64x32.

La Figura 19 muestra el EDP, de las configuraciones de array rectangulares. Para las cargas W3, W5 y W6, el EDP baja con un mayor número de columnas. Por ejemplo, con un array de 32x64 PE, se observa una mejora muy pronunciada del EDP (disminuye en alrededor de un 50%), lo que implica que las columnas adicionales están siendo utilizadas. Al aumentar solo las filas, es decir 64x32, el EDP aumenta ligeramente, lo que significa que el incremento de consumo energético no se compensa con la mejora de prestaciones. Esto se debe a que el número de columnas de estas cargas no es tan grande como el de filas.

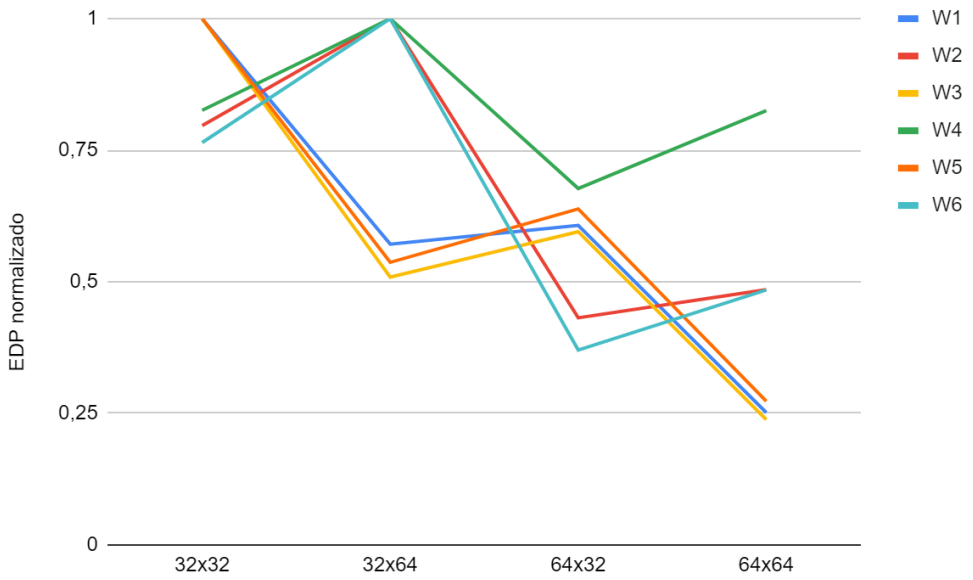



*Figura 19 EDP normalizado para diferentes configuraciones del array sistólico rectangulares WS.*

En comparación, para las cargas W2, W4 y W6 el EDP para 32x64 aumenta con respecto a 32x32, lo que significa que no se aprovechan las columnas adicionales. Por otro lado, al aumentar el tamaño de las filas nos damos cuenta de que para este caso sí que disminuye de forma considerable por lo que estas cargas cuentan con un mayor número de filas. Para finalizar con estas cargas observamos como en 64x64 vuelve a aumentar su EDP con respecto a 64x32, lo que confirma todo lo comentado anteriormente debido a que consta de un mayor número de filas que de columnas.

Estas conclusiones se corroboran en la sección 4.2.1, la cual presenta los resultados de utilización del array para configuraciones rectangulares.

## 4.1.2 Entradas Estacionarias

La Figura 20 representa el tiempo de ejecución normalizado para cada una de las cargas sobre las diferentes configuraciones de tamaño de array sistólico con el flujo de datos Entradas Estacionarias.

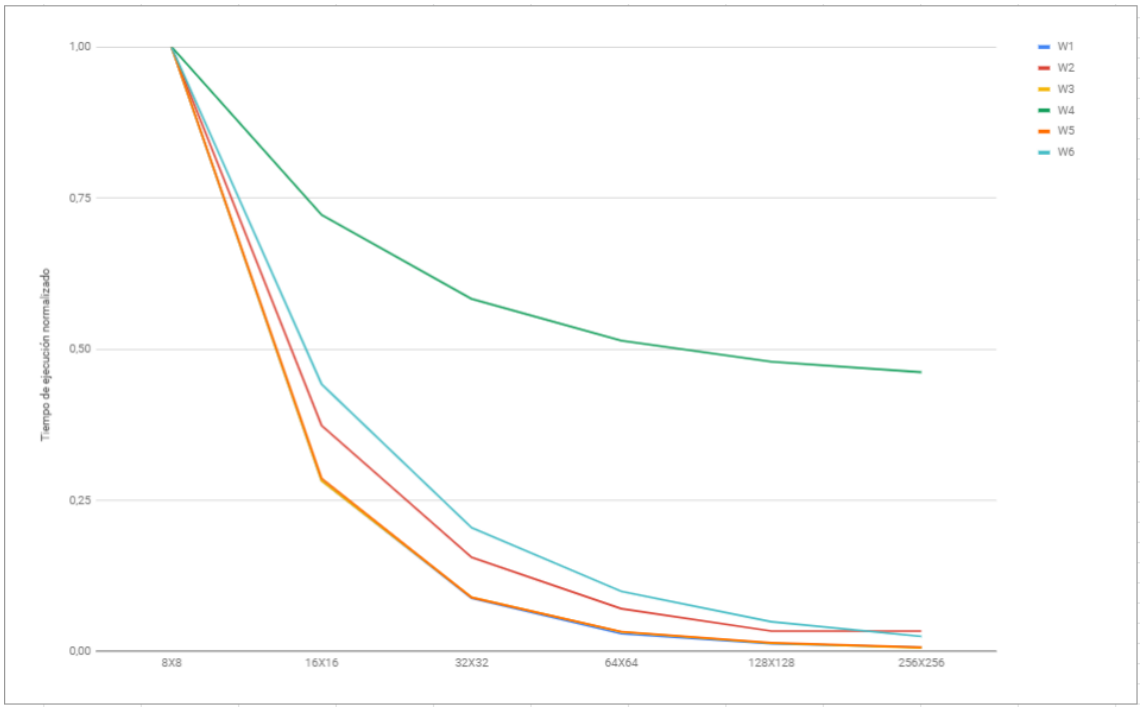



*Figura 20 Tiempo de ejecución normalizado para diferentes tamaños de array sistólico IS.*

Debido al gran parecido entre ambos flujos de datos estudiados los resultados obtenidos mantienen una gran cantidad de similitudes. Por lo tanto, las conclusiones extraídas en el apartado anterior son similares a las obtenidas en este apartado.

En la figura 20 se observa como al aumentar x4 el tamaño del array el tiempo de ejecución se reduce de forma significativa. Con la intención de obtener una visión más concreta del porcentaje de mejora sobre el tamaño de array anterior, la Tabla 3 muestra la aceleración obtenida para cada

configuración del array con respecto a la obtenida por la configuración inmediatamente anterior en número de PE.

| IS | W1 | W2 | W3 | W4 | W5 | W6 |
|---|---|---|---|---|---|---|
| 8X8 | | | | | | |
| 16X16 | 3,54 | 2,68 | 3,54 | 1,38 | 3,49 | 2,26 |
| 32X32 | 3,22 | 2,40 | 3,16 | 1,24 | 3,21 | 2,16 |
| 64X64 | 3,02 | 2,21 | 2,78 | 1,14 | 2,74 | 2,06 |
| 128X128 | 2,32 | 2,11 | 2,42 | 1,07 | 2,28 | 2,02 |
| 256X256 | 1,91 | 2,04 | 2,34 | 1,04 | 2,09 | 1,99 |

*Tabla 3 Mejora sobre el tiempo de ejecución de la configuración anterior IS.*

Se observa una clara tendencia en la que el tiempo de ejecución disminuye de manera significativa hasta que se produce un estancamiento. Tal y como se ha comentado en el flujo de datos WS este estancamiento se debe a que a partir de ciertos tamaños de array seguir aumentando su tamaño no ofrece prestaciones significativas. Al igual que ocurre con WS, existen de dos configuraciones (32x32 y 64x64 PE) en las cuales la aceleración al duplicar la anchura y altura del array resulta interesante.

Como hemos comentado en la sección anterior realizar un estudio referente al EDP resulta muy útil para confirmar las sospechas al respecto de cuál es el mejor tamaño de array. En la Figura 21 se muestra dicho gráfico:

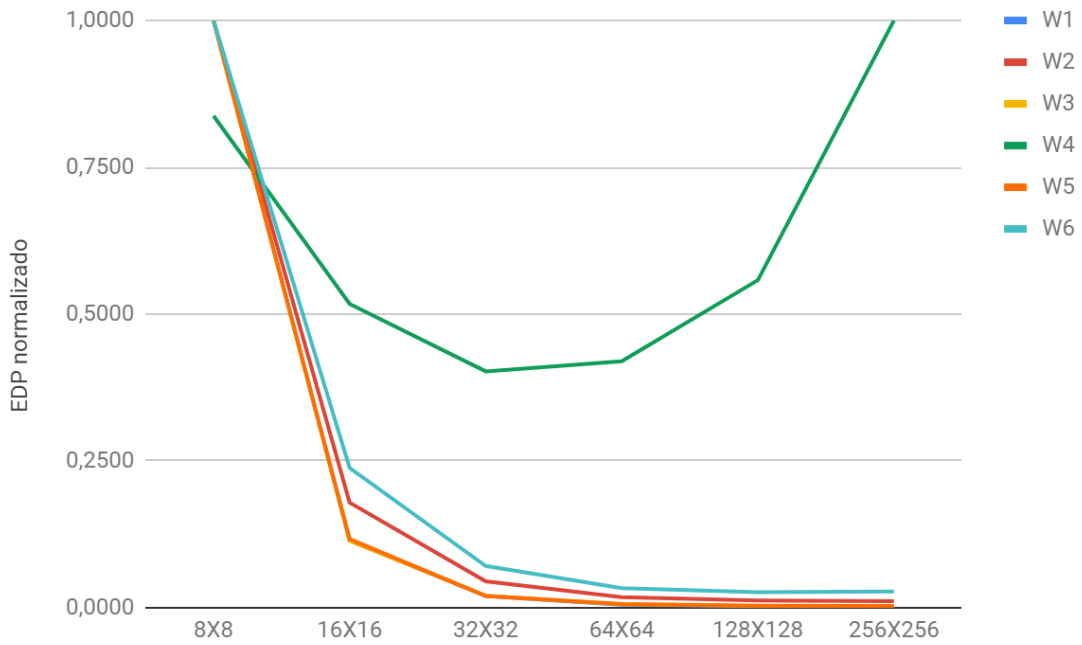



*Figura 21 EDP normalizado para diferentes configuraciones cuadradas del array sistólico IS.*

En la Figura 21 observamos como el EDP obtiene mejoras significativas hasta 64x64, tamaño a partir del cual deja de ser tan relevante dicha mejora. Por esta razón analizaremos lo que ocurre en este tamaño y el inmediatamente menor a él puesto que sabemos que las configuraciones mayores a 64x64 tienen unos requerimientos de ancho de banda bastante más altos al compararlo con el otro flujo de datos estudiado en este trabajo.

Recordemos que este grafico se refiere únicamente al EDP, es decir a la relación entre tiempo de ejecución y energía consumida y no tiene en cuenta los requerimientos de ancho de banda ni el porcentaje de utilización del array por lo que, aunque en el gráfico se observe que lo más lógico pudiera ser establecer un tamaño de 256x256, para todas las cargas excepto para W4, como veremos en las siguientes secciones esto no es realista.

Continuando con el estudio de las configuraciones rectangulares, cuyos tamaños son 32x32, 32x64, 64x32, 64x64, los cuales coinciden con los que se han estudiado en WS, obtenemos el siguiente gráfico referente al tiempo de ejecución de estas.

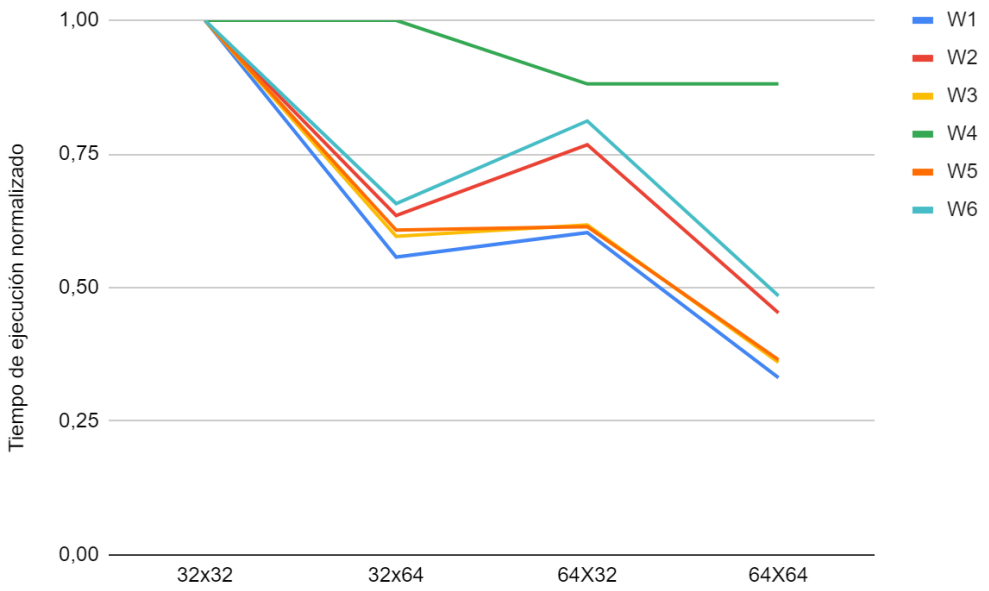



*Figura 22 Tiempo de ejecución normalizado para diferentes configuraciones del array sistólico rectangulares IS.*

Tras observar la Figura 22 nos damos cuenta de que excepto para W4, todas las cargas sufren una mejora significativa al aumentar el tamaño de las columnas. Esto se debe a que el flujo de datos IS realiza una precarga de los IFMAP en el array sistólico, recordemos que WS hace una precarga de los FILTER, y al aumentar el número de columnas la cantidad de datos que se

introducen en el array sistólico es mayor. Puesto que las cargas W2 y W6 tienen una gran cantidad de IFMAPS se ven beneficiadas por el flujo de datos actual, al contrario de lo que ocurría en WS. Al observar lo que ocurre cuando aumentamos el número de filas (64x32) observamos que todas las cargas aumentan su tiempo de ejecución debido a que el número de filtros no es lo suficientemente alto como para aprovechar todo el espacio extra.

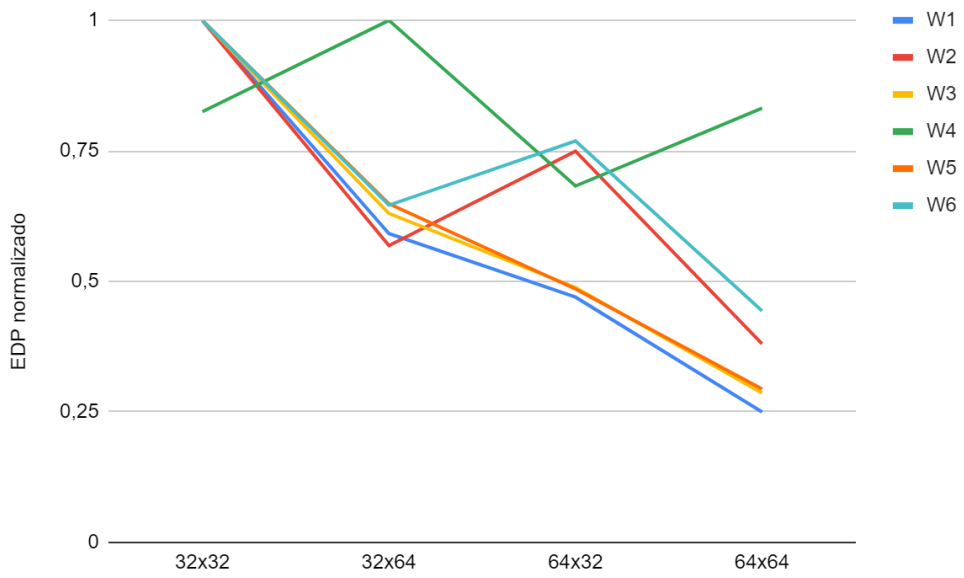



*Figura 23 EDP normalizado para diferentes configuraciones del array sistólico rectangulares IS.*

En cuanto a la Figura 23 referente al EDP de las configuraciones rectangulares observamos como se confirma lo comentado anteriormente, pero al tener en cuenta el consumo los cambios producidos por las cargas según la configuración arquitectónica se ven más acentuados en el caso de W2 y W6 mientras que para el resto de las cargas se suavizan.

Estas conclusiones pueden verse apoyadas al analizar los gráficos referentes al porcentaje de utilización que se comentarán en la sección 4.2.1.

Con la intención de obtener conclusiones con respecto a que flujo de datos obtiene un mejor tiempo de ejecución hemos realizado un gráfico de áreas comparándolos para los distintos tamaños de array estudiados, siendo el color azul el que representa a WS y el rojo a IS.

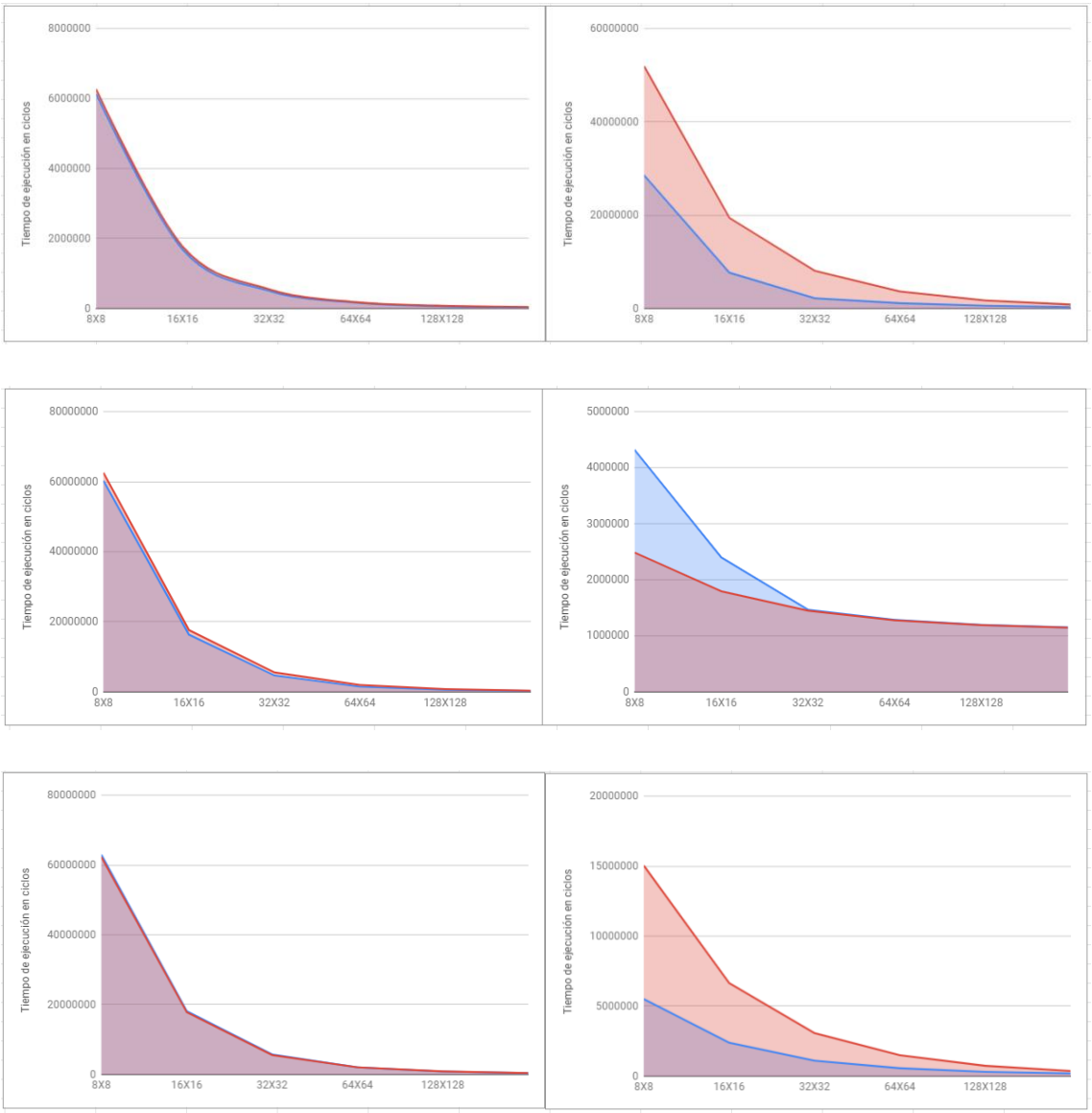



*Figura 24 Comparación entre los tiempos de ejecución para las cargas desde la W1 hasta la W6 de los flujos de datos WS e IS para los diferentes tamaños de configuración del array sistólico.*

Observamos en la Figura 24 que para W1, W3 y W5 no existe apenas diferencia. Sin embargo, para W2 y W6 el flujo de datos IS proporciona un mejor tiempo de ejecución bastante significativo el cual va disminuyendo según aumenta el tamaño de array sistólico sin llegar a converger en el tamaño máximo lo que nos incita a pensar que el tamaño del array puede seguir incrementándose para obtener mejores resultados. Mientras que para W4 se observa que WS ofrece un tiempo de ejecución bastante mejor, pero esto solo ocurre para tamaños de array inferiores a 32x32, ya que a partir de este tamaño ambos flujos de datos obtienen los mismos tiempos de ejecución.

## 4.2 Utilización del array sistólico

### 4.2.1 Configuraciones cuadradas

En el siguiente apartado comentaremos el porcentaje de utilización medio de las diferentes configuraciones sobre el array sistólico. Hemos decidido obtener el porcentaje de utilización medio debido a que el simulador nos proporciona el porcentaje de utilización de cada capa, pero no el referente a la carga completa. El porcentaje de utilización medio se ha obtenido tras multiplicar el número de ciclos de cada capa por el porcentaje de utilización de dicha capa, para posteriormente sumar estos valores y dividirlos entre el número total de ciclos de ejecución de la carga completa. Los gráficos obtenidos con los porcentajes de utilización medios contienen además el tiempo de ejecución normalizado de cada una de las configuraciones, el cual está representado mediante la línea roja que recorre el gráfico.

En las Figuras 25 y 26 observamos los gráficos referidos al porcentaje de utilización de cada uno de los tamaños de array para cada una de las cargas (están ordenadas en orden ascendiente desde W1 hasta W6).

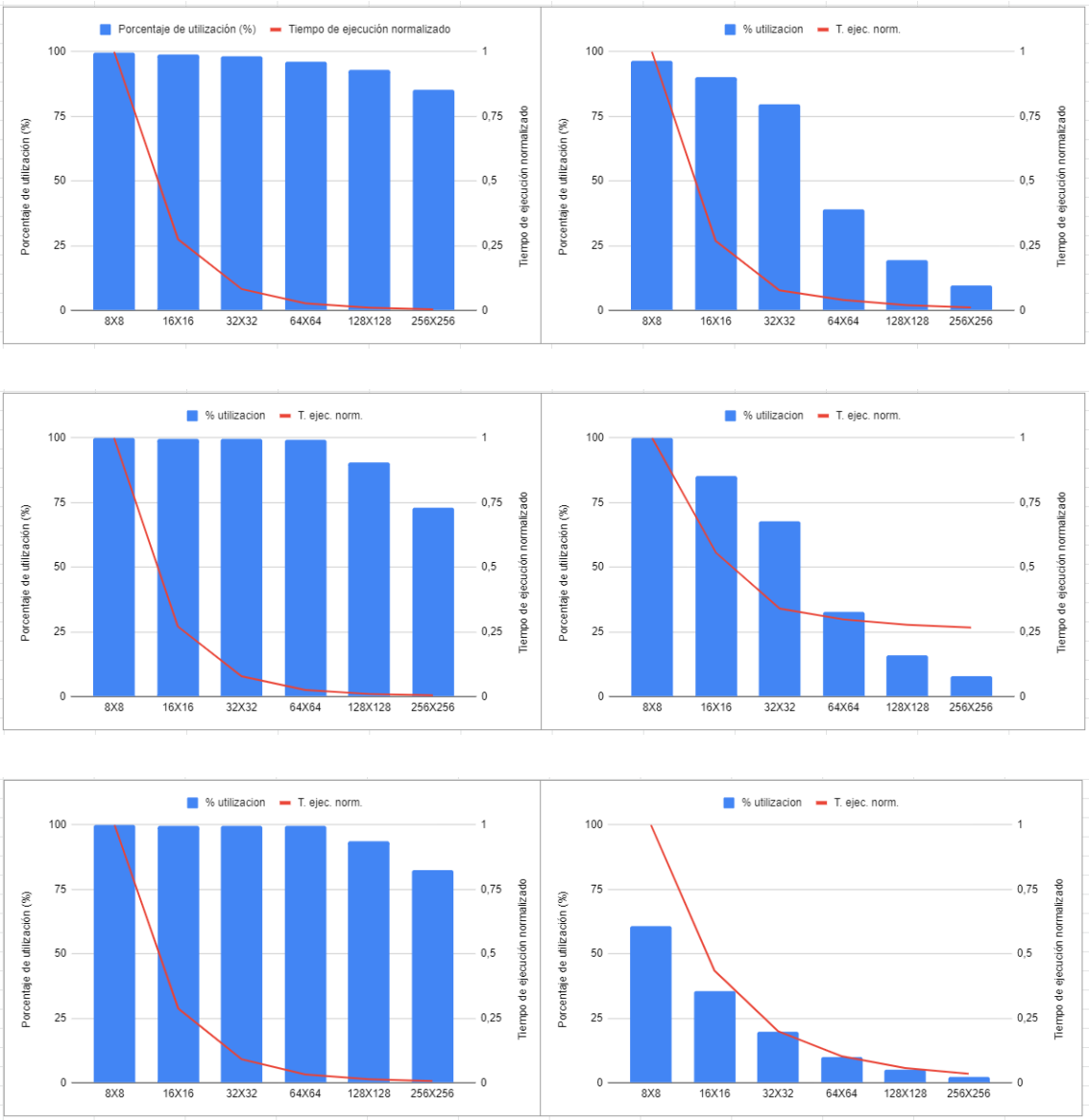



*Figura 25 Porcentaje de utilización medio de las cargas W1-W2, W3-W4, W5-W6 en conjunto con el tiempo de ejecución normalizado de las mismas WS.*

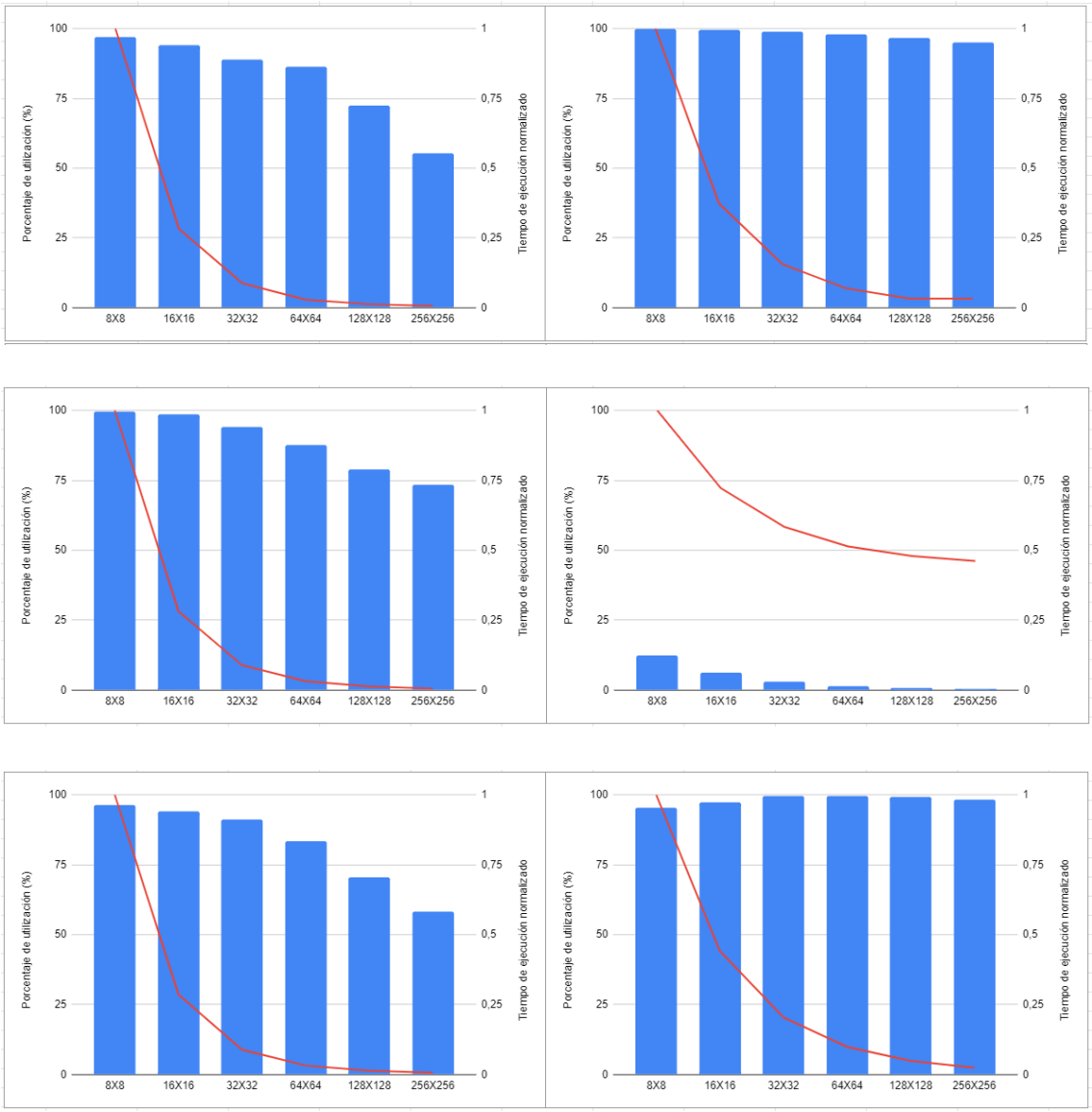



*Figura 26 Porcentaje de utilización medio de las cargas W1-W2, W3-W4, W5-W6 en conjunto con el tiempo de ejecución normalizado de las mismas IS.*

Como podemos observar el porcentaje de utilización para tamaños bajos, como 8x8 o 16x16, se mantiene a unos niveles bastante altos, la mayoría de ellos se encuentran entorno al 90%.

Sin embargo, es fácil darse cuenta de lo que hemos comentado en la sección anterior sobre los grandes tamaños de array. El porcentaje de utilización del array para tamaños de 128x128 y 256x256 disminuye considerablemente, lo cual repercute directamente sobre el consumo debido a la gran cantidad de PE que no están siendo utilizados, pero que a su vez están consumiendo energía estática.

La posibilidad de tener un gran tamaño de array que sea capaz de ofrecernos un rendimiento mayor viene acompañada de la problemática de que en algunos casos no estará siendo utilizado

al completo. Por ende, se pueden aplicar diferentes soluciones a este problema en el caso de que se quiera implementar dicho tamaño de array. Por ejemplo, podemos apagar cierta parte del array cuando no se está utilizando, o bien usar esos PE inactivos para iniciar la ejecución de la siguiente capa para realizar la ejecución en paralelo. Sobre estas ideas hablaremos de forma más extensa y teniendo en cuenta más variables en la sección 5.

Continuando con el porcentaje de utilización podemos observar las Tablas 4 y 5 en las que se muestran los porcentajes de utilización de cada una de las capas para cada una de las cargas estudiadas con la intención de que se pueda observar de forma más clara lo comentado anteriormente.

| AVG OF UTILIZA | W1 | W2 | W3 | W4 | W5 | W6 |
|---|---|---|---|---|---|---|
| 8X8 | 99,50 | 96,49 | 99,87 | 100,00 | 99,90 | 60,70 |
| 16X16 | 99,08 | 90,22 | 99,69 | 85,25 | 99,77 | 35,36 |
| 32X32 | 98,11 | 79,75 | 99,67 | 67,79 | 99,77 | 19,74 |
| 64X64 | 96,02 | 39,18 | 99,22 | 32,76 | 99,49 | 10,05 |
| 128X128 | 92,83 | 19,46 | 90,35 | 16,03 | 93,70 | 4,96 |
| 256X256 | 85,15 | 9,61 | 73,16 | 7,91 | 82,55 | 2,34 |

*Tabla 4 Porcentaje de utilización de las diferentes cargas respecto a los tamaños de array estudiados WS.*

| AVG OF UTILIZA | W1 | W2 | W3 | W4 | W5 | W6 |
|---|---|---|---|---|---|---|
| 8X8 | 96,92 | 99,75 | 99,43 | 12,50 | 96,38 | 95,34669296 |
| 16X16 | 93,96 | 99,47 | 98,43 | 6,25 | 93,94 | 97,26183613 |
| 32X32 | 88,86 | 99,04 | 94,00 | 3,12 | 91,09 | 99,47294152 |
| 64X64 | 86,30 | 98,02 | 87,59 | 1,56 | 83,55 | 99,64055642 |
| 128X128 | 72,33 | 96,67 | 78,90 | 0,78 | 70,46 | 99,18271065 |
| 256X256 | 55,39 | 95,14 | 73,30 | 0,39 | 58,38 | 98,14152678 |

*Tabla 5 Porcentaje de utilización de las diferentes cargas respecto a los tamaños de array estudiados IS*

Las celdas en amarillo representan el momento a partir del que se produce un gran descenso en el porcentaje de utilización del array sistólico.

## 4.2.2 Configuraciones rectangulares

En esta sección observaremos como las conclusiones extraídas de la sección 4.1 referentes a las ejecuciones rectangulares se corresponden con los consumos energéticos.

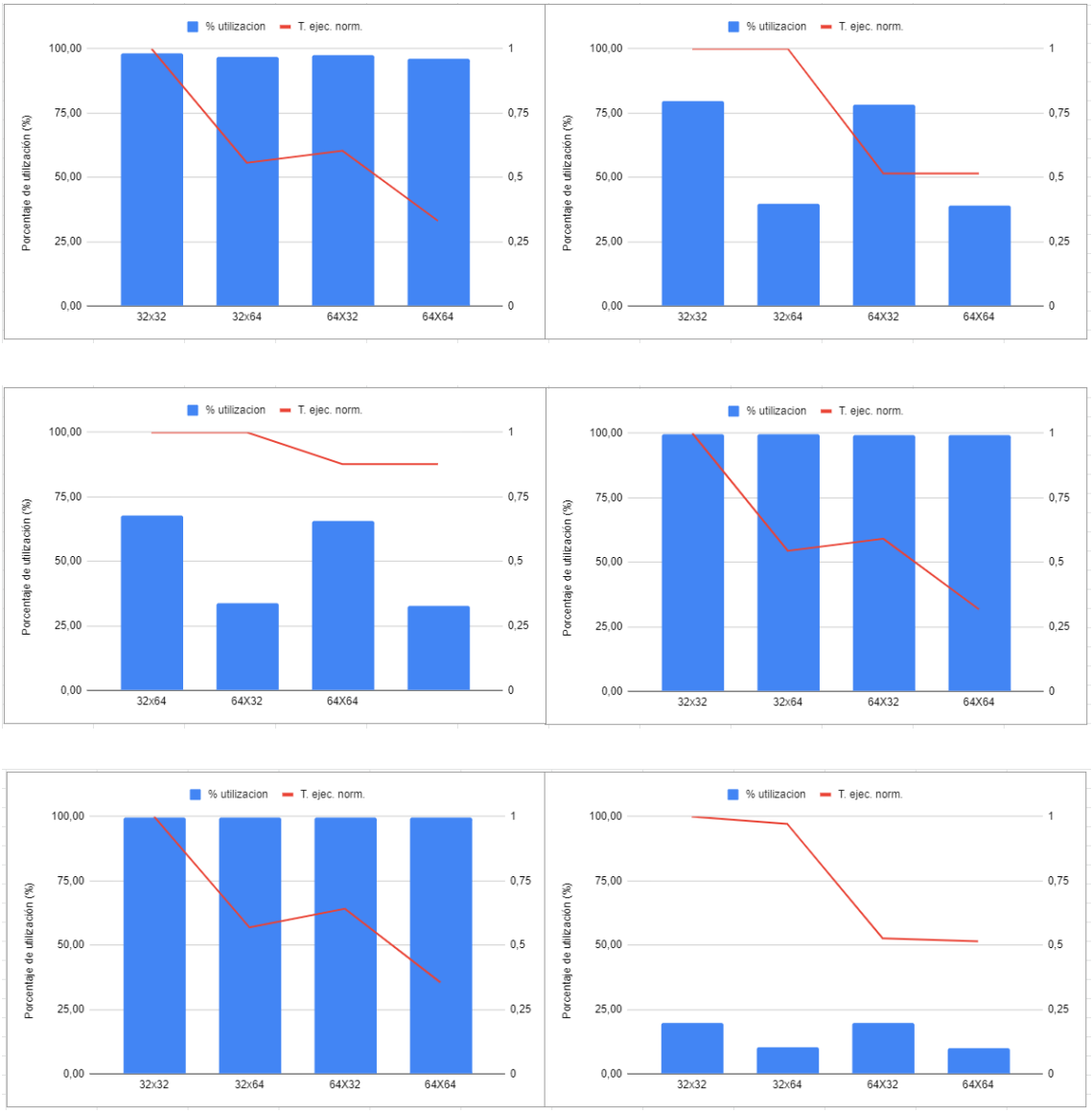



*Figura 27 Porcentaje de utilización y tiempo de ejecución respecto a los tamaños 32x32, 32x64, 64x32, 64x64 para las cargas W1-W2, W3-W4, W5-W6 WS.*

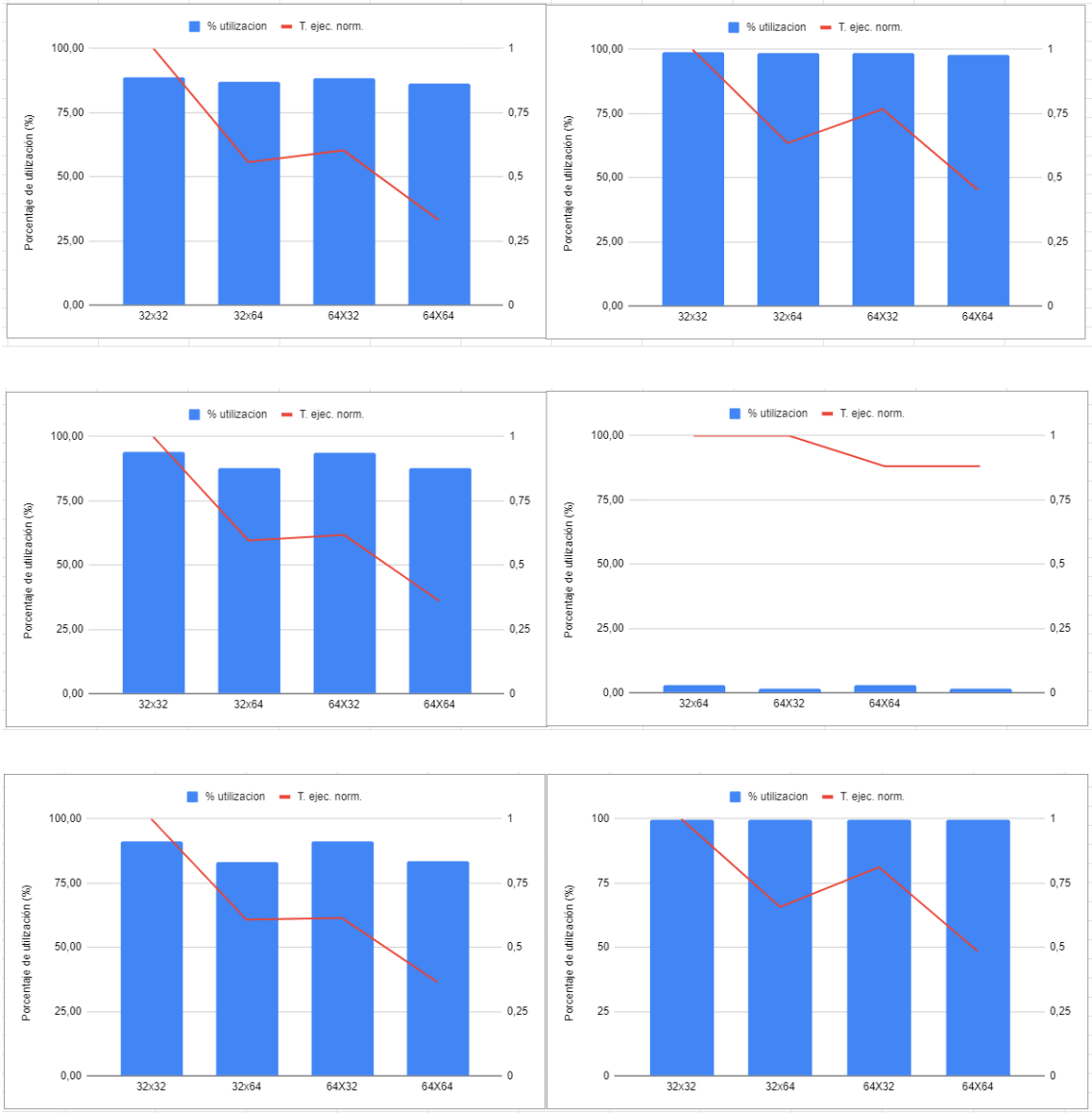



*Figura 28 Figura 29 Porcentaje de utilización y tiempo de ejecución respecto a los tamaños 32x32, 32x64, 64x32, 64x64 para las cargas W1-W2, W3-W4, W5-W6 IS.*

Observando las Figuras 27 y 28, en referencia al flujo de datos WS, tal y como habíamos comentado las cargas W2 y W6 se caracterizan por tener un gran número de filas (IFMAP), así pues, podemos observar cómo se confirma lo comentado anteriormente puesto que tras el porcentaje de utilización aumenta en 64x32 respecto a 32x64, mientras que el tiempo de ejecución disminuye de forma simultánea.

Por otro lado, al referirnos al flujo de datos IS es fácil darse cuenta de que el porcentaje de utilización medio de todas las cargas es mucho mayor que en WS debido a que realiza una precarga de IFMAPS, siendo estos por normal general bastante más grandes de los FILTERS.

## 4.3 Anchos de banda mínimos requeridos

En la siguiente sección comentaremos el ancho de banda que requeriría la implementación en un sistema real de las simulaciones ejecutadas sobre las diferentes cargas y configuraciones.

### 4.3.1 Weight Stacionary

Antes de comenzar con la explicación de los gráficos obtenidos de los resultados obtenidos gracias al simulador vamos a comentar las diferentes memorias que van a ser utilizadas sobre los mismos las cuales están resumidas en la Tabla 6.

En primer lugar, contamos con memorias tipo DDR4 en 4 versiones diferentes según su frecuencia de funcionamiento, obteniendo así diferentes anchos de banda. Estos anchos de banda se pueden observar en la Tabla 6.

En segundo lugar, tenemos una memoria tipo DDR5, la cual aún no han salido al mercado, pero lo hará de forma inminente (2020/2021), la cual proporciona anchos de banda bastante mejores que su predecesor y su precio no es tan elevado.

En tercer lugar, tenemos una memoria HBM2, la cual nos ofrece los mejores anchos de banda, pero esto repercute de forma directa en el precio de las mismas. También disponemos de su versión mejorada HBM2E que ya se encuentra en el mercado y por último de HBM3 la cual saldrá al mercado en los próximos años. [15]

| | |
|---|---|
| DDR4 2133 | 17 |
| DDR4 2400 | 19,2 |
| DDR4 2666 | 21,3 |
| DDR4 3200 | 25,6 |
| DDR5 | 51,2 |
| HBM2 | 256 |
| HBM2E | 307 |
| HBM3 | 512 |

*Tabla 6 Diferentes memorias estudiadas con sus respectivos anchos de banda en GB/s.*

Para el estudio de los resultados hemos elaborado una serie de gráficos, Figuras 29 y 30, que siguen el patrón comentado anteriormente, la primera captura se refiere a W1, la segunda a W2 y así sucesivamente. Además, estos gráficos están acompañador por una serie de líneas horizontales que se refieren a los anchos de banda de las diferentes memorias estudiadas, siendo así fácil de

observar a primera vista que configuraciones o cargas son posibles de integrar en un sistema real obteniendo los resultados esperados y cuáles no.

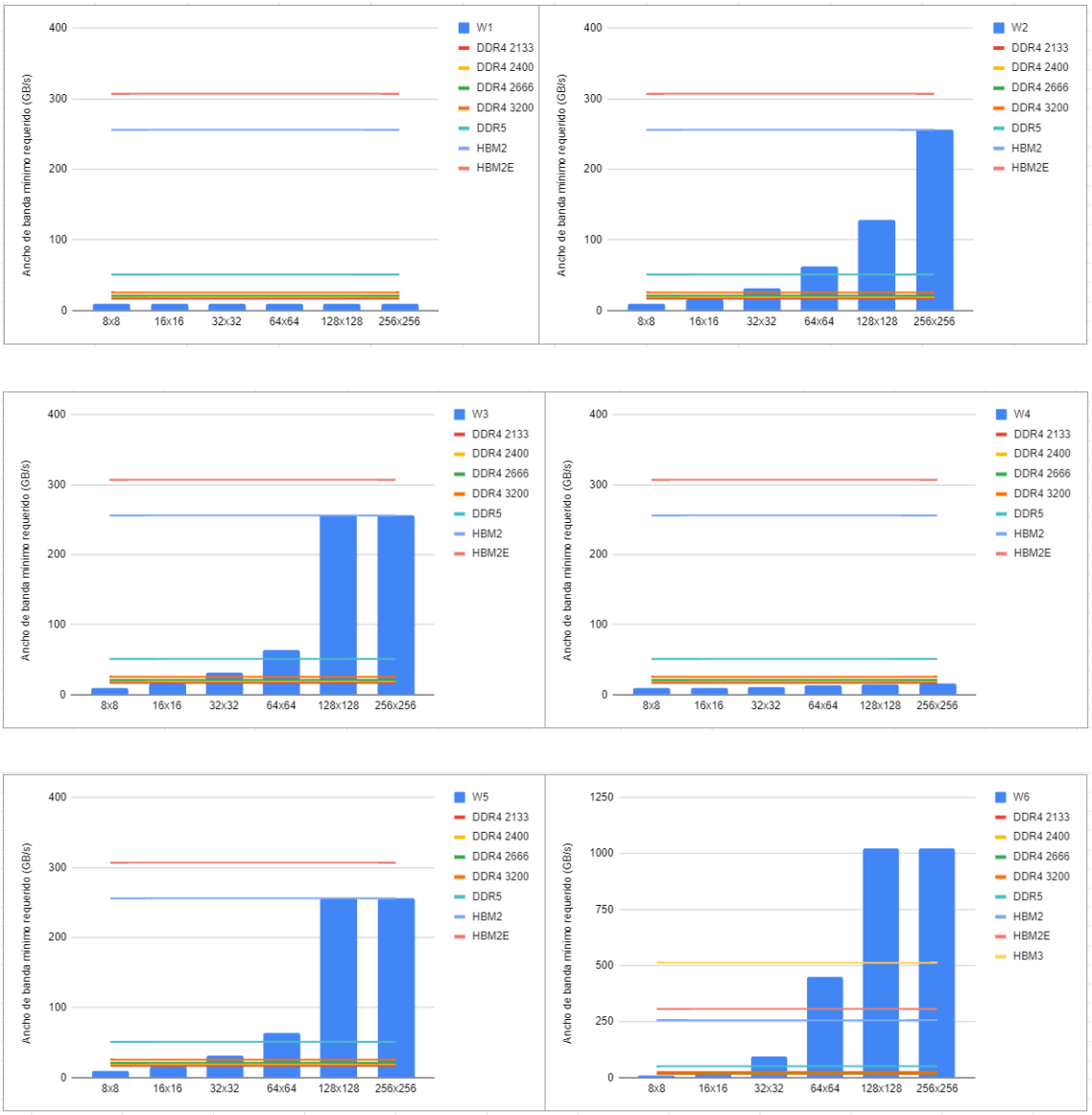



*Figura 30 – Ancho de banda máximo requerido por W1-W2, W3-W4, W5-W6 para las diferentes configuraciones de tamaño de array sistólico y los anchos de banda proporcionados por las memorias estudiadas en GB/s WS.*

Comentando de forma más específica vamos a centrarnos en los tamaños de array más interesantes hasta el momento, 32x32 y 64x64. Para 32x32 con una memoria DDR5, la cual ofrece 51,2 GB/s de ancho de banda, podríamos ejecutar todas las cargas que se observan en la imagen a excepción de W6. Por otro lado, en cuanto al tamaño de array de 64x64 nos encontramos con que únicamente podríamos ejecutar sobre una memoria DDR5 W1 y W4 obteniendo los resultados esperados, sin embargo, a excepción de W6 el resto de las cargas se encuentran ligeramente por encima del ancho de banda proporcionado por DDR5 por lo que el estudio de la inclusión de ciclos de parada podría ser interesante puesto que las memorias DDR5 suponen un

coste mucho menor que las memorias HBM. A priori W6 no podría ser ejecutada sin ciclos de parada en ninguna de las memorias que se pueden adquirir hoy en día, por lo que habría que esperar a que HBM3 saliese al mercado, lo cual puede suponer un problema dado que el coste del sistema real se incrementará considerablemente.

### 4.3.2 Input Stacionary

Ahora observemos lo que ocurre en cuanto al flujo de datos de entrada estacionaria:

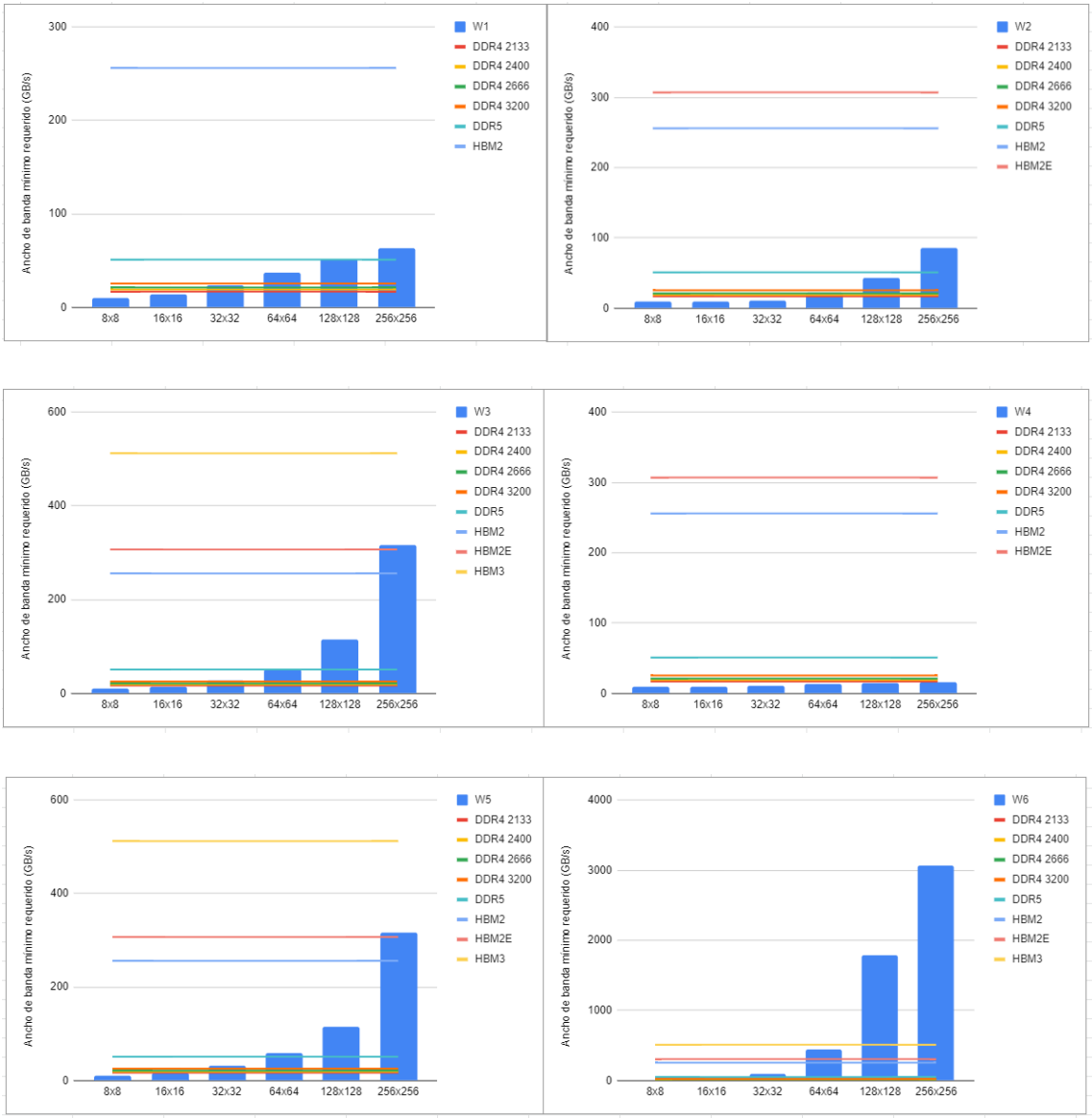



*Figura 31 Ancho de banda máximo requerido por W1-W2, W3-W4, W5-W6 para las diferentes configuraciones de tamaño de array sistólico y los anchos de banda proporcionados por las memorias estudiadas en GB/s IS.*

Para la mayoría de las cargas (W1, W2, W3, W4) para un tamaño de 32x32 o de 64x64 con una memoria del tipo DDR5 tendríamos suficiente. Vamos a observar de forma más detalla lo

que ocurre con W5 y W6 para los tamaños comentados anteriormente. Por un lado, W5 sobre una configuración de 32x32 puede ser computado en una memoria DDR5 mientras que para un tamaño de 64x64 requiere un ancho de banda de 59GB/s, sabiendo que DDR5 ofrece un ancho de banda de 51,2GB/s por lo que en la inclusión de HBM2 supondría un sobrecoste difícil de justificar debido a que el ancho de banda requerido es ínfimamente mayor que el proporcionado por DDR5.

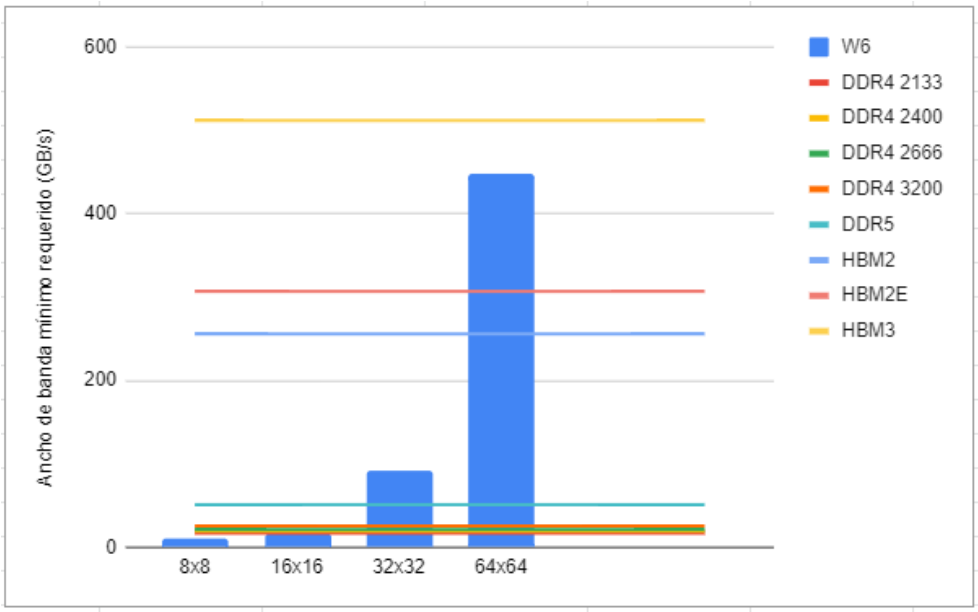



*Figura 32 Ancho de banda máximo requerido por W6 para las diferentes configuraciones de tamaño de array sistólico y los anchos de banda proporcionados por las memorias estudiadas en GB/s IS.*

Para hablar más detalladamente sobre W6 observamos en la Figura 32 se han eliminado los anchos de banda requeridos por los tamaños de 128x128 y 256x256 por ser demasiado altos. La carga W6 sobre una configuración de 32x32 requiere un ancho de banda de 92GB/s y para una configuración de 64x64 requiere un ancho de banda de 448GB/s. Con los datos comentados anteriormente nos damos cuenta de que para poder ejecutar las cargas estudiadas sobre un hardware real sería necesaria la inclusión de memorias HBM2/HBM2E que son las que podemos adquirir hoy en día, y pese a la inclusión de estas memorias si escogiéramos un tamaño de array de 64x64 algunas de las cargas a ejecutar sufrirían una bajada con respecto al rendimiento esperado.

Por lo tanto, ante la problemática comentada anteriormente para ambos flujos de datos existen dos posibilidades, una de ellas es esperar hasta que se lancen al mercado las memorias HBM3 las cuales ofrecen un ancho de banda de 512GB/s y así conseguiríamos el rendimiento esperado. Otra de las opciones sería estudiar el impacto sobre el resultado final que obtendríamos si introdujésemos ciclos de parada, cosa que el simulador no contempla, pudiendo así utilizar memorias con anchos de banda menores. Esta última solución puede observarse en la sección 4.5.

## 4.4 Consumos energéticos

Para esta sección vamos a comentar los gráficos obtenidos referentes al consumo del array sistólico para cada una de las configuraciones, Figuras 33 y 34. Así pues, los gráficos están formados por la suma de la energía estática, y la energía dinámica. En esta sección se comentarán ambos flujos de datos al mismo tiempo puesto la comparación entre ambos es lo más interesante a la hora de sacar conclusiones.

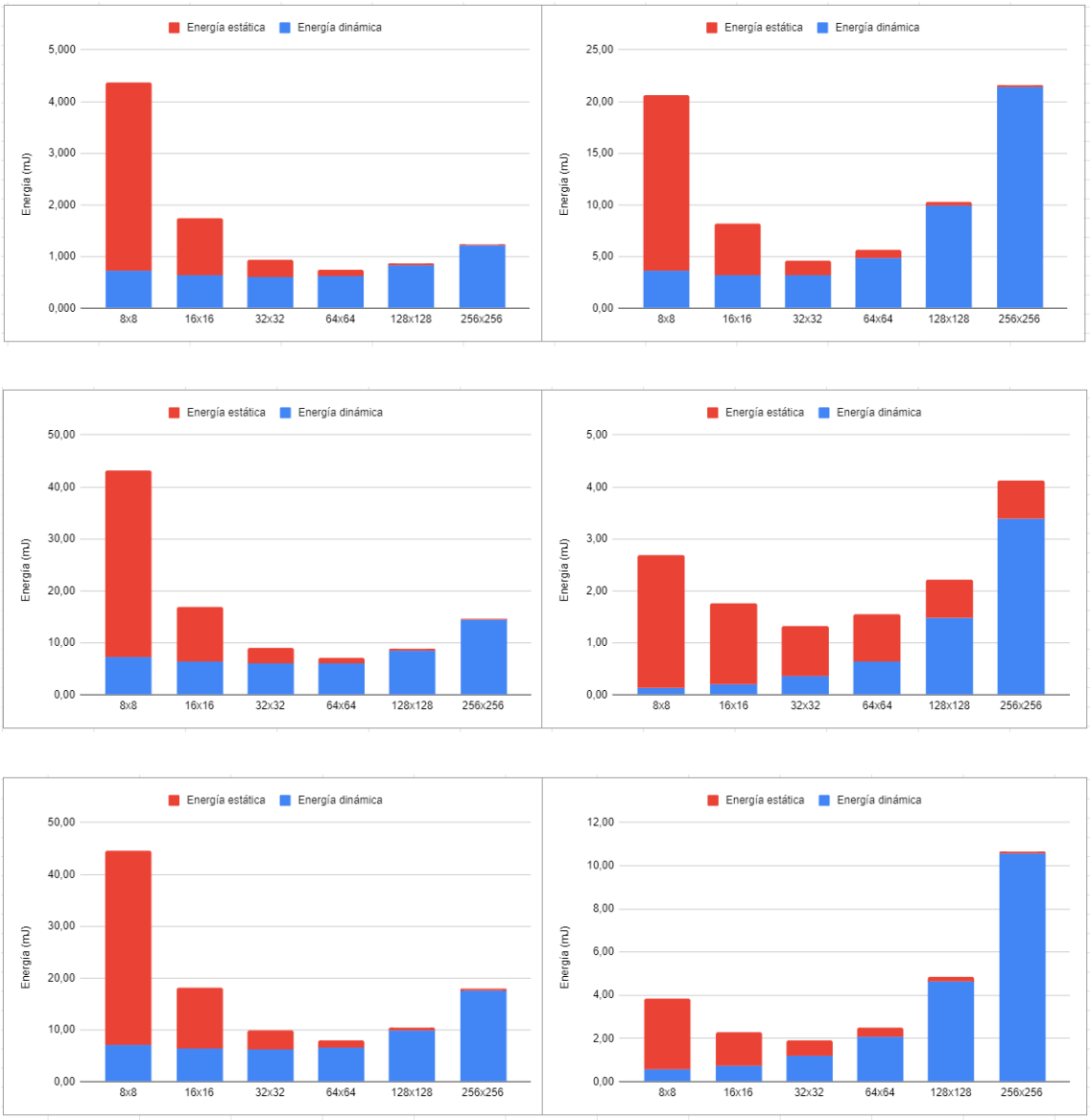



*Figura 33 Gráficos referentes al consumo energético sobre la suma del consumo estático y dinámico de los diferentes tamaños de array para las cargas W1-W2, W3-W4, W5-W6 sobre el flujo de datos WS.*

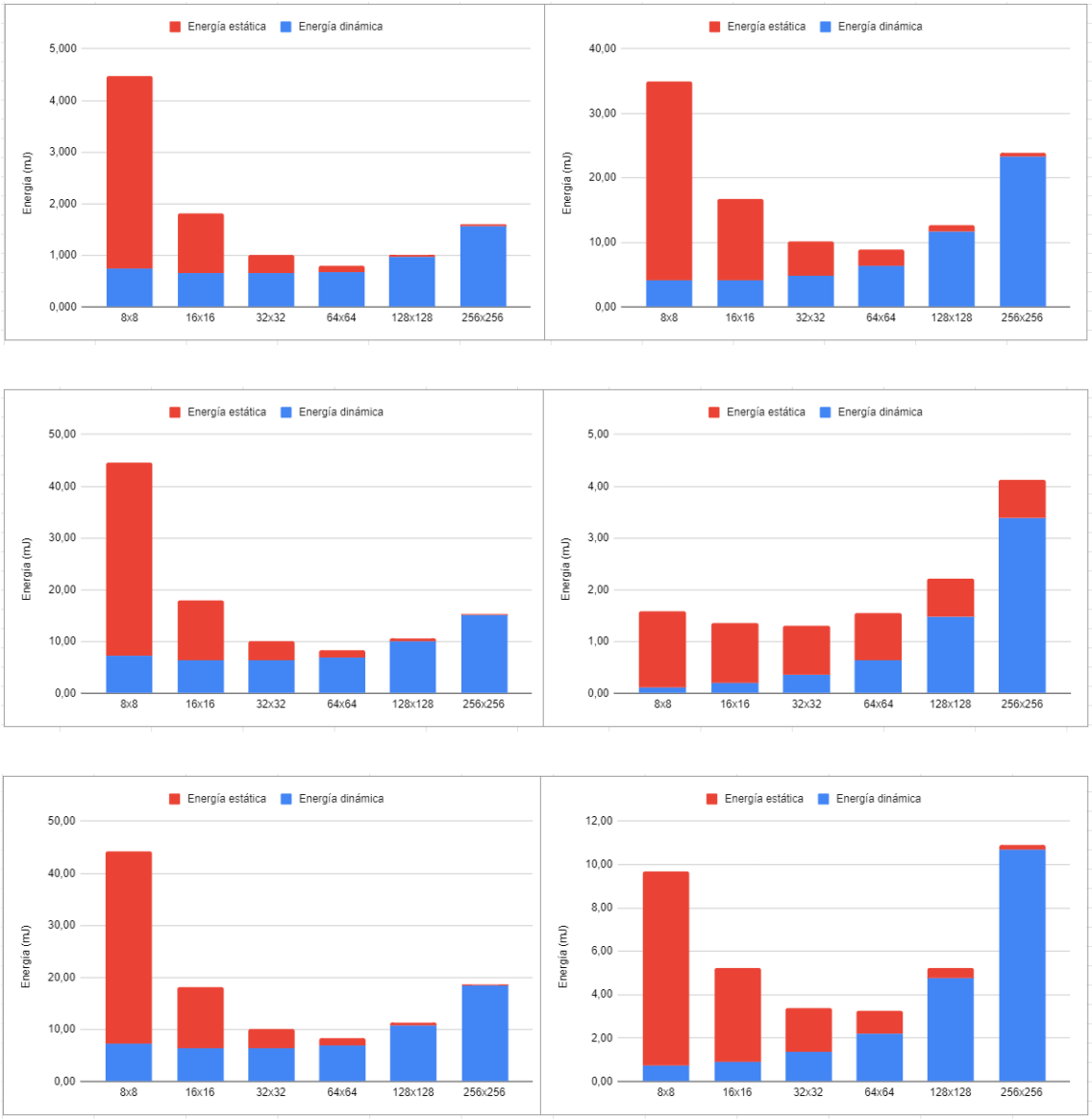



*Figura 34 Gráficos referentes al consumo energético sobre la suma del consumo estático y dinámico de los diferentes tamaños de array para las cargas W1-W2, W3-W4, W5-W6 sobre el flujo de datos IS.*

Fijándonos en las Figuras 33 y 34 observamos como se produce una "U" para todas las cargas en las diferentes configuraciones y en ambos flujos de datos, por lo que podemos confirmar lo que se ha comentado anteriormente sobre la elección del tamaño de array. Se observa como para tamaños de array bajos la energía dinámica consumida es mucho mayor que la energía estática, como es lógico, mientras que en tamaños altos de array ocurre a la inversa. Como conclusión para los datos estudiados observamos que los tamaños de array con mejores consumos energéticos son 32x32 y 64x64.

Una vez ha quedado claro cuál es el tamaño más interesante vamos a observar desde un punto de vista energético que flujo de datos proporciona un menor consumo. Para ello hemos realizado las gráficas de la Figura 35 en las que se comparan las energías consumidas sobre la misma carga para cada flujo de datos. Quedando así con el color azul el flujo de datos de WS y con el rojo el flujo de datos de IS y las cargas ordenadas de mayor a menor W1, W2…

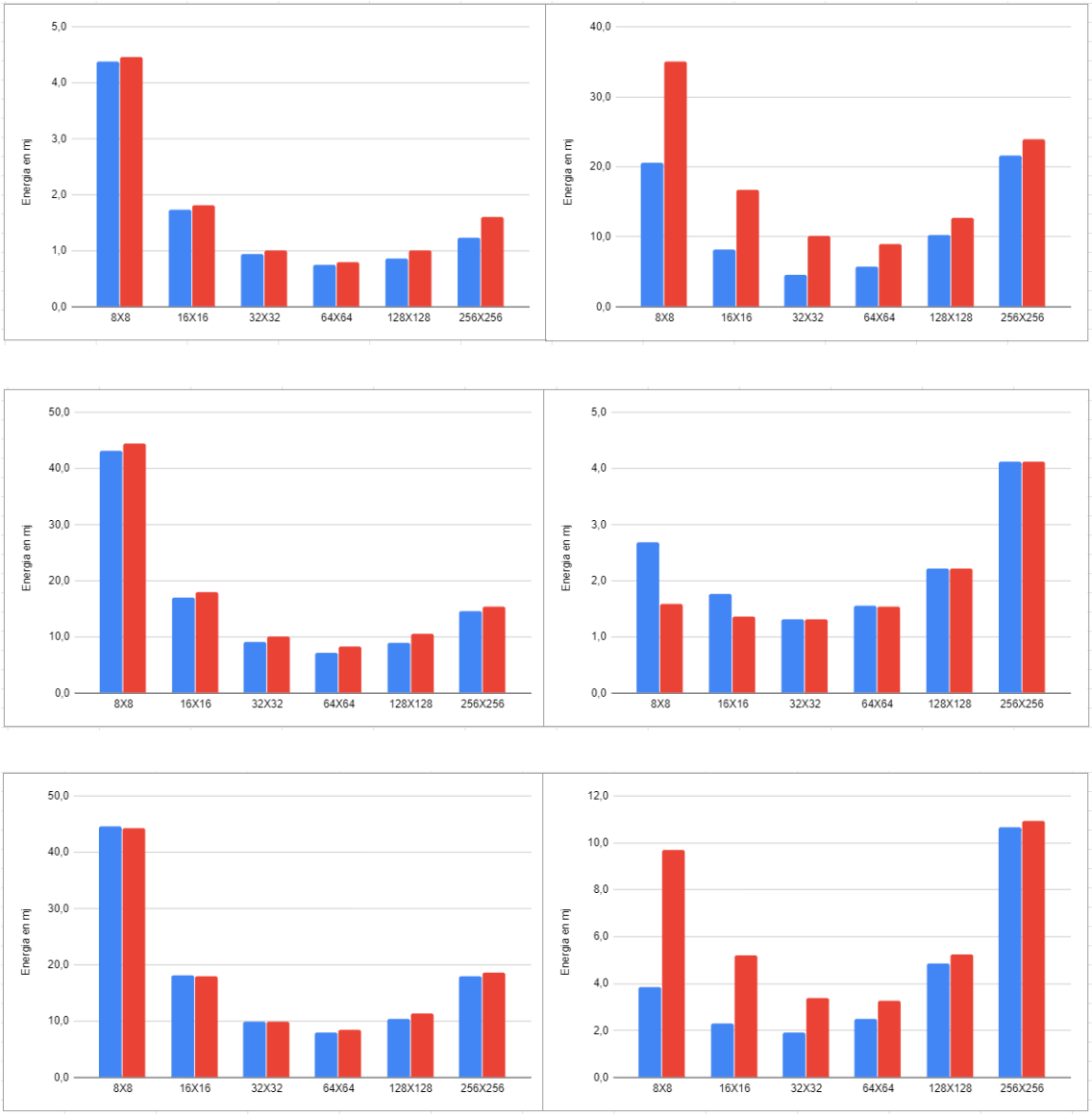



*Figura 35 Comparación sobre el consumo de cada uno de los flujos de datos sobre las cargas W1-W2, W3-W4, W5-W6.*

Podemos observar como en las cargas W1, W3, W4, y W5 el aplicar un flujo de datos u otro no afecta prácticamente en el consumo. Sin embargo, al analizar con más detenimiento W2 y W6 observamos como para todos los tamaños de array que hemos probado el flujo de datos Pesos

Estacionarios (WS) ofrece un consumo significativamente menor, sobre todo en configuraciones pequeñas del array sistólico.

En el Anexo 4 podemos observar un ejemplo de configuración del CACTI con el que se han obtenido los datos para el cálculo de los consumos anteriores.

## 4.5 Tiempos de ejecución realistas

Con la intención de ofrecer unos resultados más realistas vamos a ver lo que ocurre cuando se tienen en cuenta los requerimientos de ancho de banda. Como hemos visto anteriormente este estudio resulta de interés debido a que el coste de la implementación de memorias DDR5 o HBM2 varía mucho por lo que saber la diferencia en términos de tiempo de ejecución es crucial para decidirse entre una memoria u otra.

Para calcular el tiempo de ejecución realista hemos sumado los anchos de banda máximos referentes a las tres matrices IFMAP, FILTER y OFMAP proporcionados por el simulador. Posteriormente si este ancho de banda es mayor que el que ofrece las memorias comentadas se divide entre el ancho de banda de las mismas obteniendo el factor por el que hay que multiplicar el tiempo de ejecución ideal para obtener el tiempo de ejecución real. Cabe recordar que el simulador no implementa ciclos de parada.

En las Figuras 36 y 37 observamos el incremento del tiempo de ejecución real frente al tiempo de ejecución proporcionado por el simulador (valor 1).

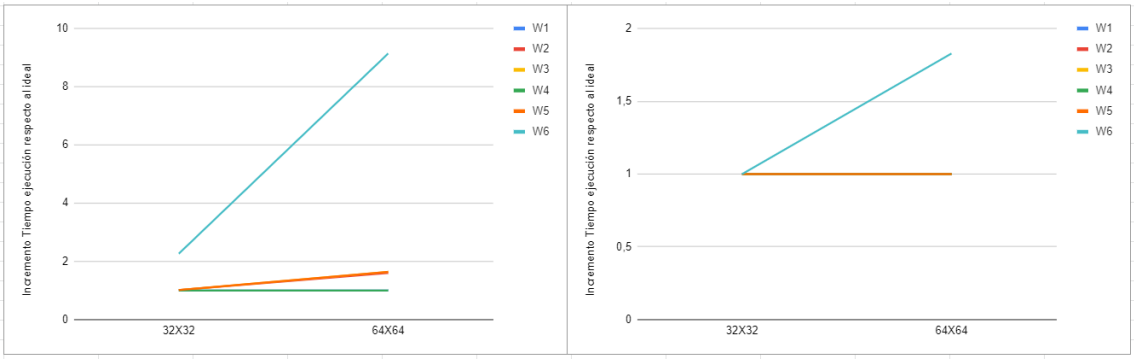



*Figura 36 Incremento del tiempo de ejecución ideal proporcionado por el simulador frente a la implementación sobre memorias DDr5 y HBM2 WS.*

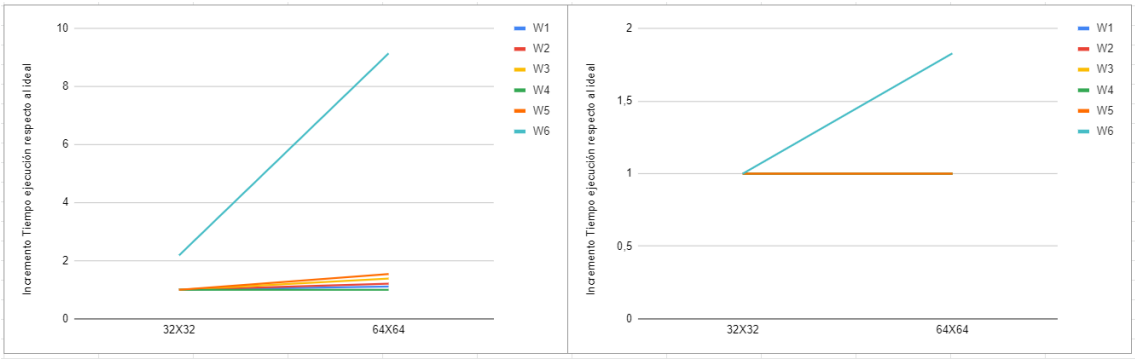



*Figura 37 Incremento del tiempo de ejecución ideal proporcionado por el simulador frente a la implementación sobre memorias DDr5 y HBM2 IS.*

Se extrae como conclusión que la mayoría de las cargas estudiadas no sufrirían apenas ningún aumento en su tiempo de ejecución a excepción de W6. De forma más concreta W6 sobre una memoria DDR5 vería como su tiempo de ejecución se duplica para configuraciones de tamaño del array sistólico de 32x32 mientras que para un tamaño de 64x64 su tiempo de ejecución se vería multiplicado nueve veces más aproximadamente. Por otro lado, en cuanto a la implementación sobre memorias HBM2 observamos como todas las cargas mantendrían sus tiempos de ejecución excepto W6 que lo vería incrementado en un 1.8%.

Tras las evidencias mostradas anteriormente es tarea de la persona que esté interesada en implementar un cierto tamaño de array sistólico si le parece suficiente o no las mejoras otorgadas por HBM2 con respecto a su diferencia de precio en el mercado comparada con DDR5.

# 5 Conclusiones

Las redes neuronales nos descubren un campo de investigación enorme que se encuentra en pleno auge y puede ser de gran utilidad tanto para el mundo de la informática como para toda la sociedad. Existen una infinidad de aplicaciones para las mismas que hoy en día son solo teorías pero que con el paso del tiempo se están llevando a la práctica con unos resultados sorprendentes. Las redes neuronales pueden contribuir en muchas otras áreas de investigación como pueden ser la neurología o la psicología, permitiendo modelar mecanismos propios del cerebro humano.

Una de las dos grandes limitaciones que rodea hoy en día a las redes neuronales recae sobre sus requerimientos a nivel de hardware puesto que están basadas en el procesamiento de la información de forma paralela. Hoy en día los procesadores existentes en el mercado no están preparados para soportar los requerimientos computacionales de las redes neuronales a nivel de paralelismo. Así pues, la investigación sobre nuevos tipos de procesadores, como los que se han comentado este trabajo, como por ejemplo la TPU de Google, sobre la cual está basada el simulador SCALE-Sim estudiado en este trabajo es crucial para que su aplicación en diferentes ámbitos se produzca lo antes posible.

La segunda limitación se refiere al aprendizaje y todo lo que éste conlleva. A grandes rasgos se puede resumir en la complejidad en cuanto al aprendizaje para grandes tareas, el elevado tiempo de aprendizaje y la gran cantidad de datos para el entrenamiento que se necesitan. Se puede encontrar información más detallada en el artículo “Introducción a las redes neuronales artificiales” de Eduardo Rivera publicado en 2007. [16]

Concluyendo con el estudio realizado en este trabajo sobre los flujos de datos con precarga sobre aceleradores CNN sistólicos tras analizar todas las variables que influyen en su rendimiento y consumo podemos decir:

a) El uso de WS o IS como flujo de datos no afecta de forma significativa a la hora de analizar el tiempo de ejecución obtenido, sin embargo, sabemos que IS ofrece un mejor tiempo de ejecución para dos de las seis cargas estudiadas.
b) En cuanto al porcentaje de utilización sí que se ha podido observar como la implementación de un flujo de datos con precarga de IFMAP (IS) ofrece resultados mucho mejores al compararlo con WS.
c) En cuanto al ancho de banda necesario para poder llevar a cabo las ejecuciones de las cargas estudiadas se ha podido observar como para los tamaños que resultan de interés hoy en día no existe una gran diferencia. Por otro lado, para tamaños del array sistólico

grandes, como pueden ser 128x128 y 256x256 sí que es bastante importante comentar que el flujo de datos WS requiere una cantidad de ancho de banda significativamente menor lo cual puede ser interesante en estudios posteriores.

d) Cómo ultima variable estudiada el consumo energético por parte de ambos flujos de datos es similar para la mayoría de las cargas, sin embargo, dos de ellas si que se benefician de la implementación de WS.

Finalmente, a efectos prácticos hoy en día resultaría más interesante la implementación de un flujo de datos con precarga de IFMAP puesto que sus resultados de tiempo de ejecución y porcentaje de utilización son claramente superiores. De cara a un estudio futuro es interesante comentar que mediante la introducción de técnicas como apagar parte del array sistólico cuando no se esté utilizando, o aún mejor, permitir que este espacio libre pueda continuar con el computo de la siguiente capa, y así ratificar que la implementación de mayores tamaños de array puede resultar muy interesante. La problemática reside en que las memorias actuales no son capaces de ofrecer los anchos de banda necesarios para estos tamaños de array, pero cuando la tecnología empleada en las memorias avance y esto no suponga un problema para su implementación en un sistema real, gracias al simulador se ha observado que el flujo de datos WS requiere mucho menos ancho de banda para configuraciones de array grandes.

Este trabajo, en conjunto con el de Eduardo Yago Vicent, se realiza sobre arrays sistólicos en el seno del grupo de investigación GAP. Se toma como una aproximación para sentar las bases de cara a una investigación posterior sobre las redes neuronales convolucionales con visión a una publicación internacional dónde se estudie el rendimiento y consumo de estas. Por último, cabe mencionar que se pueden realizar estudios futuros ampliando resultados obtenidos en este TFG, por ejemplo, indagando en la memoria DRAM y su impacto sobre el tiempo de ejecución y en lo referente al consumo energético.

## 5.1 Competencias transversales

Finalmente se comentan algunas competencias transversales que se han desarrollado durante la realización del presente trabajo.

**CT-04. Innovación, creatividad y emprendimiento.** En relación al tema tratado durante la realización de este proyecto considero que se ha innovado puesto que este trabajo y su variante son los primeros trabajos relacionados con los arrays sistólicos y su aplicación en el cómputo de redes neuronales artificiales.

**CT-06. Trabajo en equipo y liderazgo.** Este trabajo de investigación se ha realizado en conjunto con mi compañero Eduardo tal y como hemos hablado en la sección 1.7 además de contar con los tutores mencionados en ambas memorias por lo que la coordinación entre todas las partes del equipo ha jugado un papel muy importante.

**CT-07. Responsabilidad ética, medioambiental y profesional.** Puesto que lo que se busca en este trabajo es encontrar el tamaño de array que consiga un mejor rendimiento y consumo esta competencia transversal es más que obvia. Todo esto se puede ver de forma más detallada en la sección 6.

**CT-10. Conocimiento de problemas contemporáneos.** La IA y las redes neuronales artificiales están a la orden del día y haber realizado una investigación sobre ellas me ha otorgado una visión mucho más amplia de la repercusión del avance de la investigación de las mismas. El desarrollo de la IA ya está provocando grandes cambios en la sociedad actual sin embargo los cambios más grandes están aún por llegar.

# 6 Bibliografía


[1]"Red neuronal artificial", Es.wikipedia.org, 2019. [Online]. Disponible: https://es.wikipedia.org/wiki/Red_neuronal_artificial (Accessed on 08/2019)

[2] “Aprendizaje profundo”, 2019. *Es.wikipedia.org* [online], Disponible: https://es.wikipedia.org/wiki/Aprendizaje_profundo (Accessed on 08/2019)

[3] CALVO, DIEGO, 2019, Clasificación de redes neuronales artificiales - Diego Calvo. *Diego Calvo* [online]. 2019. Available from: http://www.diegocalvo.es/clasificacion-de-redes-neuronales-artificiales (Accessed on 07/2019)

[4]"¿Qué son y cuáles son los tipos de redes neuronales? | Neuronoticias", Neuronoticias, 2019. [Online]. Disponible: http://neuronoticias.com/2018-06-27/tipos-de-redes-neuronales (Accessed on 08/2019)

[5] CALVO, DIEGO, 2019, Red Neuronal Convolucional CNN - Diego Calvo. *Diego Calvo*[online]. 2019. Available from: http://www.diegocalvo.es/red-neuronal-convolucional (Accessed on 07/2019)

[6] CHIEN-PING, LU, 2019, Should We All Embrace Systolic Arrays? *Medium* [online]. 2019. Available from: https://medium.com/@CPLu/should-we-all-embrace-systolic-array-df3830f193dc

[7]"An in-depth look at Google’s first Tensor Processing Unit (TPU) | Google Cloud Blog", *Google Cloud Blog*, 2019. [Online]. Disponible: https://cloud.google.com/blog/products/gcp/an-in-depth-look-at-googles-first-tensor-processing-unit-tpu (Accessed on 08/2019)

[8] Y.-H. Chen, T. Krishna, J. Emer, y V. Sze, “Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks,” in International Solid-State Circuits Conference, ser. ISSCC, 2016

[9] ADRADE TEPÁN, EVA CRISTINA, 2019, *https://dspace.ups.edu.ec/bitstream/123456789/4098/1/UPS-CT002584.pdf* Licenciatura. Universidad Politécnica Salesiana. (Accessed on 08/2019)

[10] Google Goggles, 2019. *Es.wikipedia.org* [online], Disponible: https://es.wikipedia.org/wiki/Google_Goggles (Accessed on 08/2019)

[11] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina y T. Krishna, *SCALE-Sim: Systolic CNN Accelerator Simulator* en Nueva York: Cornell University, 2019.

[12] VARELA ARREGOCÉS, ERNESTO and CAMPBELLS SÁNCHEZ, EDWIN, 2019, *Redes Neuronales Artificiales: Una Revisión del Estado del Arte, Aplicaciones Y Tendencias Futuras*. Profesor de Matemáticas Avanzadas. Universidad Simón Bolívar.

[13] ANANDA, SAMAJDAR, ZHU, YUHAO and WHATMOUGH, PAUL, 2018, "SCALE-Sim: Systolic CNN Accelerator Simulator". *GitHub* [online]. 2018. Available from: https://github.com/ARM-software/SCALE-Sim (Accessed on 08/2019)

[14] Dally, W. (2015). *High-Performance Hardware for Machine Learning*.

[15] High Bandwidth Memory. *En.wikipedia.org* [online]. 2019. Available from: https://en.wikipedia.org/wiki/High_Bandwidth_Memory (Accessed on 08/2019)

[16] RIVERA, EDUARDO, 2007, *Introducción a las redes neuronales artificiales*.

[17] Ignacio Bagnato, J. (2019). *¿Cómo funcionan las Convolutional Neural Networks? Visión por Ordenador*. [online] Aprende Machine Learning. Available at: https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador (Accessed on 08/2019)

# 7 Anexos

## 7.1 Anexo 1 – Cómputo de una CNN

En esta sección cuando se haga referencia a píxel se entiende que un píxel es cada uno de los elementos que conforman una matriz de una red neuronal.

El cómputo de una red neuronal es un proceso algorítmico complejo el cual es beneficioso para compresión del trabajo realizado. En la Figura 38 podemos observar una representación gráfica de dicho cómputo.

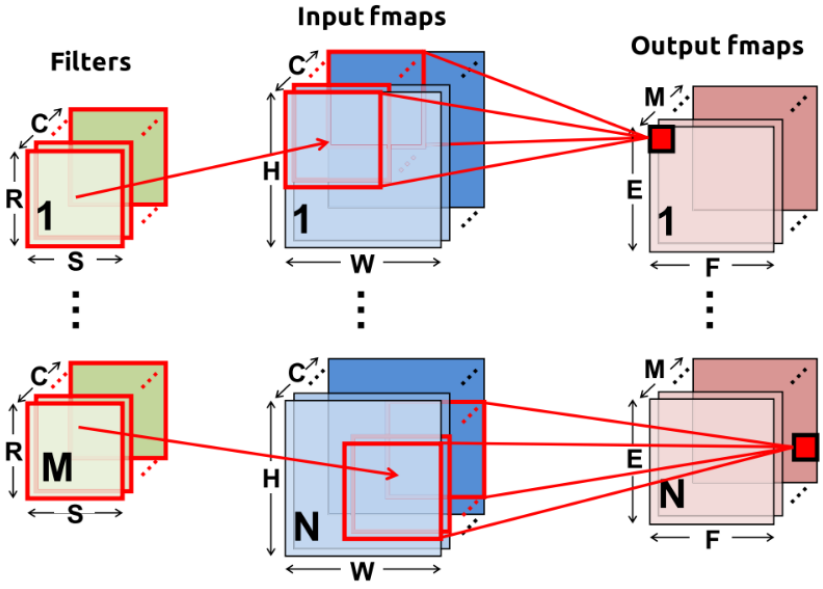



*Figura 38 Ejemplo de cálculo de una red neuronal*

Este cálculo puede representarse de forma matemática con la fórmula de la Figura 39:

$$\mathbf{O}[z][u][x][y] = \mathrm{ReLU}\left(\mathbf{B}[u] + \sum_{k=0}^{C-1}\sum_{i=0}^{R-1}\sum_{j=0}^{S-1}\mathbf{I}[z][k][Ux+i][Uy+j] \times \mathbf{W}[u][k][i][j]\right),$$

*Figura 39 Fórmula referente al cálculo de una red neuronal*

Los elementos que forman la fórmula son en primer lugar la matriz O la cual se corresponde con la matriz de salida, en segundo lugar, tenemos la matriz de sesgos B, en tercer lugar, I la cual se refiere a la matriz de entrada y para acabar observamos la matriz de filtros o pesos W.

Para ser más concreto hay que destacar las restricciones de la Figura 40:

$$0 \le z < N, \quad 0 \le u < M, \; 0 \le y < E, \; 0 \le x < F$$

$$E = (H - R + U)/U, \quad F = (W - S + U)/U$$

*Figura 40 Restricciones*

E y F se calculan mediante el álgebra matricial para la multiplicación de matrices, además observamos que el parámetro U se corresponde con el "stride" o tamaño de paso. El tamaño de paso se puede definir como el desplazamiento de los elementos que forman una matriz dada. Tras la explicación del cálculo de matrices.

Así pues, comencemos con la explicación de dicho cálculo. Se habla de subconjuntos de las matrices debido a que el tamaño de la matriz de entrada suele ser mayor que el tamaño de la matriz de salida y como es necesario que todos los datos de entrada sean computados con los elementos de la matriz de filtros/pesos tiene que hacerse en varias etapas. En primer lugar, se multiplican un subconjunto de cada una de las matrices de entrada con otro subconjunto de la matriz de filtros/pesos obteniendo así un subconjunto de la matriz de salida. Esta última matriz es sometida a una reducción obtenido así un único píxel de la matriz de salida final. Este proceso se repite hasta que todos los datos de la matriz de entrada han sido computados con la matriz de filtros/pesos.

Volviendo al concepto de "stride" comentado anteriormente, en este caso el "stride" es el que delimita los subconjuntos de las matrices utilizadas y va aumentando hasta que se han computado todos los elementos de la matriz de entrada.

Al observar las matrices implicadas en el cálculo nos damos cuenta de que hemos hablado de todas ellas con anterioridad a excepción de la matriz de sesgos B, debido a que no juega un papel clave en el cómputo de la red neuronal. Más concretamente su función es la de añadir un valor constante al resultado final del cómputo de la neurona con la intención de que este valor ajuste el umbral de activación de esta. Cabe destacar que esta matriz es semejante a la función ReLU.

Siendo más concisos la función ReLU es una función de activación, y como tal se utiliza para eliminar la linealidad en el resultado final del cómputo de la red neuronal. Por linealidad entendemos que el resultado de la función de activación solo pueda dar como resultado dos

valores, activación normalmente asociado con un 1 y no activación normalmente asociado con el valor 0. Su intención es la de evitar valores intermedios.

Existen muchos tipos de funciones similares a la ReLU que son empleadas para el cálculo de CNN, pero sin duda ésta es la más utilizada puesto que no se satura para valores de salida muy altos. Algunas de ellas son la sigmoide, la tangente hiperbólica, leaky ReLU o Softmax. Sus fórmulas se pueden observar en la Figura 41:

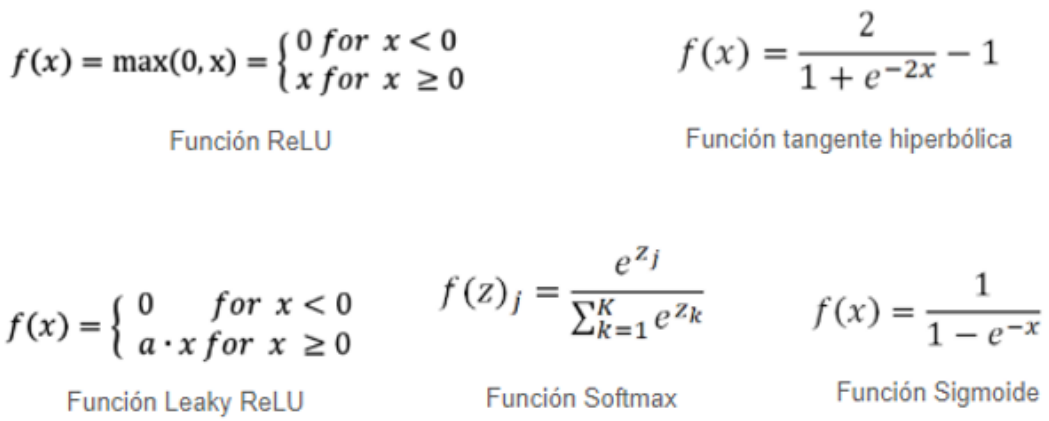



*Figura 41 Funciones similares a ReLU*

Antes de finalizar con esta sección se debe comentar que cuando se habla de matrices se habla de matrices tridimensionales, obteniendo, así como resultado una matriz de cuatro dimensiones entendiendo que esta última dimensión es el número de matrices tridimensionales.

## 7.2 Anexo 2 – “script-configs.sh” y “script-ejecucion.sh”

Ambos scripts son exactamente iguales a la hora de introducir los datos y a la hora de recorrer las diferentes variables introducidas, tales como, el flujo de datos y los parámetros arquitectónicos. Se diferencian en que “script-configs.sh” se dedica exclusivamente a la creación de los archivos de configuración mientras que “script-ejecucion.sh” se dedica a ejecutar de forma simultánea los archivos de configuración creados anteriormente sobre las diferentes topologías introducidas previamente. Se pueden en las Figuras 42,43,44,45,46,47:

```
#!/bin/bash

i=$3
j=$6
c=$(( $5 + 1))
z=$(( $8 + 1))
if [ "$2" = "all" ]; then dat=(os ws is)
else
    dat=($2)
fi

if [ "$1" == "--help" ]; then
      echo "PARAMETERS:"
    echo "\$1: [-s for squared only, by default is non squared]"
    echo "\$2: [dataFlow(os/rw/is/all)]"
    echo "\$3: [height_ini]"
    echo "\$4: [mult_fact_h]"
    echo "\$5: [h_max]"
    echo "\$6: [widh_ini]"
    echo "\$7: [mult_fact_w]"
    echo "\$8: [W_max]"
    echo "Last 3 parameters will be ignored in -s configuration."

elif [ "$1" == "-s" ]; then

    for df in ${dat[*]}
    do
        while [  $i -lt $c ];
        do

                for w in $(seq 1 1 6)
                do
                    (
                    echo "[general]"
                    echo "run_name = \"w"$w"_"$i"x"$i"_"$df"\"
                    echo

                    echo "[architecture_presets]"
                    echo "ArrayHeight:    "$i
                    echo "ArrayWidth:     "$i
                    echo "IfmapSramSz:    "512
                    echo "FilterSramSz:   "512
                    echo "OfmapSramSz:    "256
                    echo "IfmapOffset:    "0
                    echo "FilterOffset:   "10000000
                    echo "OfmapOffset:    "20000000
                    echo "Dataflow:       "$df
                    ) >> ./configs/fig6cfgs/w"$w"_"$i"x"$i"_"$df".cfg

            done
            i=$(( $i * $4 ))
        if [ "$3" == "$5" ]; then
        i=$(( $i * $4 + 1 ))
        fi

        done
    i=$3
    done
```

*Figura 42 Fragmento 1/2 "script-configs.sh"*

```
else
   for df in ${dat[*]}
   do
        while [  $i -lt $c ];
        do
            while [  $j -lt $z ];
            do
          for w in $(seq 1 1 6)
          do
            (
            echo "[general]"
            echo "run_name = \"w"$w"_"$i"x"$j"_"$df"\"
            echo

            echo "[architecture_presets]"
            echo "ArrayHeight:    "$i
            echo "ArrayWidth:     "$j
            echo "IfmapSramSz:    "512
            echo "FilterSramSz:   "512
            echo "OfmapSramSz:    "256
            echo "IfmapOffset:    "0
            echo "FilterOffset:   "10000000
            echo "OfmapOffset:    "20000000
            echo "Dataflow:       "$df
            ) >> ./configs/fig6cfgsR/w"$w"_"$i"x"$j"_"$df".cfg


          done
          if [ "$6" == "$8" ]; then
          j=$(( $j * $7 + 1))
          else
          j=$(( $j * $7 ))
          fi
            done
        j=$6
        if [ "$3" == "$5" ]; then
        i=$(( $i * $4 + 1 ))
        else
        i=$(( $i * $4 ))
        fi
        done
      i=$3
   done
fi
```

*Figura 43 Fragmento 2/2 "script-configs.sh"*

```
#!/bin/bash

i=$3
j=$6
c=$(( $5 + 1))
z=$(( $8 + 1))
if [ "$2" = "all" ]; then dat=(os ws is)
else
   dat=($2)
fi

if [ "$1" == "--help" ]; then
    echo "PARAMETERS:"
   echo "\$1: [-s for squared only, by default is non squared]"
   echo "\$2: [dataFlow(os/rw/is/all)]"
   echo "\$3: [height_ini]"
   echo "\$4: [mult_fact_h]"
   echo "\$5: [h_max]"
   echo "\$6: [widh_ini]"
   echo "\$7: [mult_fact_w]"
   echo "\$8: [W_max]"
   echo "Last 3 parameters will be ignored in -s configuration."

elif [ "$1" == "-s" ]; then

    for df in ${dat[*]}
    do
        while [  $i -lt $c ];
        do

                for w in $(seq 1 1 6)
                do
                    python3 /home/pau/Downloads/SCALE
            done
            i=$(( $i * $4 ))
      if [ "$3" == "$5" ]; then
      i=$(( $i * $4 + 1 ))
      fi

        done
    i=$3
    done
```

*Figura 44 Fragmento 1/2 "script-ejecucion.sh"*

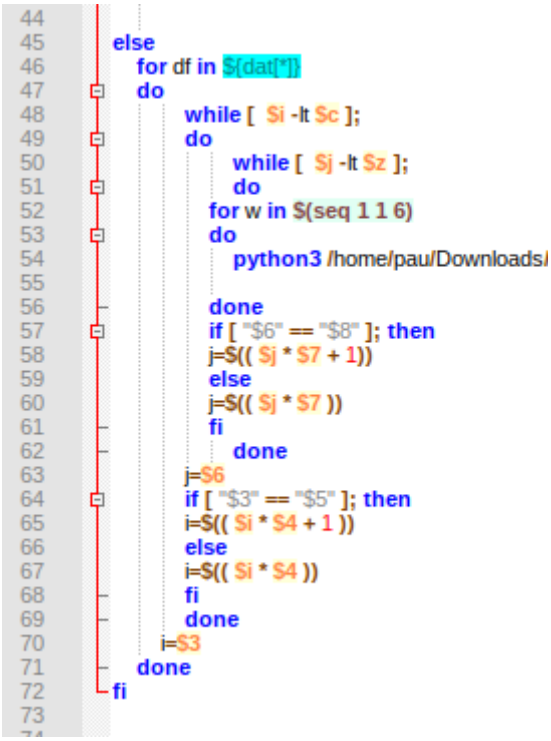



*Figura 45 Fragmento 2/2 "script-ejecucion.sh"*

Las líneas 33 y 54 que aparecen cortadas en las capturas para que se pueda observar de forma más detalla el resto de las líneas de código del script así que se comentan a continuación:

Línea 33

```
python3 /ruta/SCALE-Sim-master/scale.py -arch_config=/ruta/SCALE-Sim-master/configs/fig6cfgs/w"$w"_"$i"x"$i"_"$df".cfg -network=/ruta/SCALE-Sim-master/topologies/fig6ws/w"$w".csv
```

*Figura 46 Ampliación línea 33.*

Línea 54

```
python3 /ruta/SCALE-Sim-master/scale.py -arch_config=/ruta/SCALE-Sim-master/configs/fig6cfgsR/w"$w"_"$i"x"$j"_"$df".cfg -network=/ruta/SCALE-Sim-master/topologies/fig6ws/w"$w".csv
```

*Figura 47 Ampliación línea 54.*

En ambos fragmentos de código cabe destacar que la palabra “ruta” hace referencia al lugar dónde se encuentra el simulador en tu sistema.

```
#!/bin/bash

path=outputs
ifmapR=0
filterR=0
ofmW=0
first=0

for dir in $(find $path -type d -name w* )
 do
   dirname="${dir##*/}"
   echo "Counting in $dirname"
   for file in $(find $dir -type f -name "*sram_read.csv")
    do
      filename="${file##*/}"
      while read line_read ; do
         for line in $line_read; do
            if [ $first -gt 0 ] && [ $line != "," ]
             then
               line=${line%,}
               if [ $line -ge 10000000 ]
                then
                  let filterR++
               else
                  let ifmapR++
               fi
            fi
            first=1
         done
      first=0
      done <$file
   echo "File $filename in $dirname computed with values ifmaps: $ifmapR    filters: $filterR"
   done
   (
   echo "Output: $dirname"
   echo "Total SRAM_IFMAP reads during execution: $ifmapR"
   echo "Total SRAM_FILTER reads during execution: $filterR"
   ) >> ./resultados.out

   ifmapR=0
   filterR=0

   for file in $(find $dir -type f -name "*sram_write.csv")
    do
      filename="${file##*/}"
      while read line_read ; do
         for line in $line_read; do
            if [ $first -gt 0 ] && [ $line != "," ]
             then
               let ofmW++
            fi
            first=1
         done
      first=0
      done <$file
   echo "File $filename in $dirname computed with value ofmap: $ofmW"
   done

   (
   echo "Total SRAM_OFMAP write during execution: $ofmW"
   echo
   ) >> ./resultados.out
   ofmW=0

done
exit 0
```

*Figura 48 Código referente a "count.sh".*

w1

| Layer name | IFMAP Height | IFMAP Width | Filter Height | Filter Width | Channels | Num Filter | Strides |
|---|---|---|---|---|---|---|---|
| Conv | 19 | 19 | 3 | 3 | 17 | 256 | 1 |
| Res_conv1 | 19 | 19 | 3 | 3 | 256 | 256 | 1 |
| Res_conv2 | 19 | 19 | 3 | 3 | 256 | 256 | 1 |
| ValueHead_conv | 19 | 19 | 1 | 1 | 256 | 1 | 1 |
| ValueHead_FC1 | 1 | 1 | 1 | 1 | 361 | 256 | 1 |
| ValueHead_FC2 | 1 | 1 | 1 | 1 | 256 | 1 | 1 |
| PolicyHead_Conv | 19 | 19 | 1 | 1 | 256 | 2 | 1 |
| PolidyHead_FC | 1 | 1 | 1 | 1 | 722 | 362 | 1 |

*Figura 49 "w1.csv".*

w2

| Layer | IFMAP Width | IFMAP Width | Filter Height | Filter Width | Channels | Num Filter | Strides |
|---|---|---|---|---|---|---|---|
| DeepSpeech_conv1 | 700 | 161 | 41 | 11 | 1 | 32 | 2 |
| DeepSpeech_conv2 | 341 | 79 | 21 | 11 | 32 | 32 | 2 |
| BatchRNN1 | 672 | 2560 | 1 | 2560 | 1 | 4 | 1 |
| BatchRNN2 | 2560 | 2560 | 1 | 2560 | 1 | 4 | 1 |
| BatchRNN3 | 2560 | 2560 | 1 | 2560 | 1 | 4 | 1 |
| FC | 1 | 2560 | 1 | 2560 | 1 | 29 | 1 |

*Figura 50 "w2.csv".*

w3

| Layer name | IFMAP Height | IFMAP Width | Filter Height | Filter Width | Channels | Num Filter | Strides |
|---|---|---|---|---|---|---|---|
| Conv1 | 224 | 224 | 7 | 7 | 3 | 64 | 2 |
| CB2a_1 | 56 | 56 | 1 | 1 | 64 | 64 | 1 |
| CB2a_2 | 56 | 56 | 3 | 3 | 64 | 64 | 1 |
| CB2a_3 | 56 | 56 | 1 | 1 | 64 | 256 | 1 |
| CB2s | 56 | 56 | 1 | 1 | 64 | 256 | 1 |
| IB2b_1 | 56 | 56 | 1 | 1 | 256 | 64 | 1 |
| IB2b_2 | 56 | 56 | 3 | 3 | 64 | 64 | 1 |
| IB2b_3 | 56 | 56 | 1 | 1 | 64 | 256 | 1 |
| IB2c_1 | 56 | 56 | 1 | 1 | 256 | 64 | 1 |
| IB2c_2 | 56 | 56 | 3 | 3 | 64 | 64 | 1 |

*Figura 51 "w3.csv".*

y sigue...

w4

| Layer name | IFMAP Height | IFMAP Width | Filter Height | Filter Width | Channels | Num Filter | Strides |
|---|---|---|---|---|---|---|---|
| Neural Collaborative Filtering(Recommendation) | | | | | | | |
| MF_Embedding_user | 1 | 1 | 1 | 1 | 138000 | 8 | 1 |
| MF_Embedding_item | 1 | 1 | 1 | 1 | 138000 | 8 | 1 |
| MLP_Embedding_user | 1 | 1 | 1 | 1 | 138000 | 32 | 1 |
| MLP_Embedding_item | 1 | 1 | 1 | 1 | 138000 | 32 | 1 |
| MLP_FC1 | 1 | 1 | 1 | 1 | 64 | 32 | 1 |
| MLP_FC2 | 1 | 1 | 1 | 1 | 32 | 16 | 1 |
| MLP_FC3 | 1 | 1 | 1 | 1 | 16 | 8 | 1 |
| Predict_FC | 1 | 1 | 1 | 1 | 16 | 1 | 1 |

*Figura 52 "w4.csv".*

w5

| Layer name | IFMAP Height | IFMAP Width | Filter Height | Filter Width | Channels | Num Filter | Strides |
|---|---|---|---|---|---|---|---|
| Conv1 | 224 | 224 | 7 | 7 | 3 | 64 | 2 |
| CB2a_1 | 56 | 56 | 1 | 1 | 64 | 64 | 1 |
| CB2a_2 | 56 | 56 | 3 | 3 | 64 | 64 | 1 |
| CB2a_3 | 56 | 56 | 1 | 1 | 64 | 256 | 1 |
| CB2s | 56 | 56 | 1 | 1 | 64 | 256 | 1 |
| IB2b_1 | 56 | 56 | 1 | 1 | 256 | 64 | 1 |
| IB2b_2 | 56 | 56 | 3 | 3 | 64 | 64 | 1 |
| IB2b_3 | 56 | 56 | 1 | 1 | 64 | 256 | 1 |
| IB2c_1 | 56 | 56 | 1 | 1 | 256 | 64 | 1 |
| IB2c_2 | 56 | 56 | 3 | 3 | 64 | 64 | 1 |
| IB2c_3 | 56 | 56 | 1 | 1 | 64 | 256 | 1 |
| CB3a_1 | 56 | 56 | 1 | 1 | 256 | 128 | 2 |
| CB3a_2 | 28 | 28 | 3 | 3 | 128 | 128 | 1 |
| CB3a_3 | 28 | 28 | 1 | 1 | 128 | 512 | 1 |
| CB3s | 56 | 56 | 1 | 1 | 256 | 512 | 2 |
| IB3b_1 | 28 | 28 | 1 | 1 | 512 | 128 | 1 |
| IB3b_2 | 28 | 28 | 3 | 3 | 128 | 128 | 1 |
| IB3b_3 | 28 | 28 | 1 | 1 | 128 | 512 | 1 |
| IB3c_1 | 28 | 28 | 1 | 1 | 512 | 128 | 1 |
| IB3c_2 | 28 | 28 | 3 | 3 | 128 | 128 | 1 |

*Figura 53 "w5.csv".*

y sigue..

w6

| Layer name | IFMAP Height | IFMAP Width | Filter Height | Filter Width | Channels | Num Filter | Strides |
|---|---|---|---|---|---|---|---|
| Embedding_Laye | 1024 | 1 | 1 | 1 | 30000 | 5 | 1 |
| Conv1 | 5 | 1024 | 3 | 3 | 1 | 1024 | 1 |
| Conv2 | 5 | 1024 | 3 | 3 | 1 | 1024 | 1 |
| FC | 1 | 1 | 1 | 1 | 2048 | 2 | 1 |

*Figura 54 "w6.csv".*

-size (bytes) 524288

-block size (bytes) 8

-associativity 1

-exclusive read port 1

-exclusive write port 1

-UCA bank count 1

-technology (u) 0.032

-output/input bus width 64

-operating temperature (K) 350

-tag size (b) "default"

-access mode (normal, sequential, fast) - "fast"

-cache type (SRAM - only data array <ex:register files, buffers etc.>, SRAM_CACHE - tag & data array, DRAM_CACHE) - "SRAM"

-design objective (weight delay, dynamic power, leakage power, cycle time, area) 100:100:0:0:0

-deviate (delay, dynamic power, leakage power, cycle time, area) 20:100000:100000:100000:1000000

-NUCAdesign objective (weight delay, dynamic power, leakage power, cycle time, area) 100:100:0:0:0

-NUCAdeviate (delay, dynamic power, leakage power, cycle time, area) 10:10000:10000:10000:10000

-Optimize ED or ED^2 (ED, ED^2, NONE): "ED"

-Cache model (NUCA, UCA)  - "UCA"

-NUCA bank count 0

-wire signalling (fullswing, lowswing, default) - "default"

-Print level (DETAILED, CONCISE) - "DETAILED"