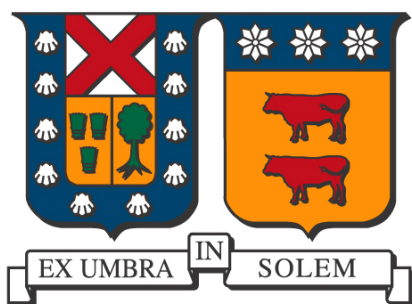


Nota

La siguiente tesis se realizó bajo el marco del convenio de doble título con Politécnico de Milan, siendo desarrollada en este ultimo. Para facilitar la reproducción de los resultados se habilitó el siguiente repositorio: <https://github.com/NicolasLanderos/Gated-CNN> el cual contiene el código desarrollado durante el proyecto.

De acuerdo a lo estipulado en el convenio se presenta a continuación una version traducida del resumen ejecutivo, junto al texto completo de la tesis en su idioma original inglés.



POLITECNICO
MILANO 1863

**SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE**

RESUMEN EJECUTIVO DE LA TESIS

Diseño de mecanismos de microarquitectura para reducir efectos de envejecimiento en memorias on-chip de aceleradores de redes neuronales convolucionales

PROGRAMA DE DOBLE TITULACIÓN: INGENIERÍA CIVIL ELECTRÓNICA // LAUREA MAGISTRALE IN
COMPUTER SCIENCE AND ENGINEERING

Autor: NICOLÁS LANDEROS MUÑOZ

Profesores Guía: PROF. GONZALO CARVAJAL, PROF. DAVIDE ZONI

Asesor: PROF. ALEJANDRO VALERO

Año académico: 2023-2024

1. Introducción

Los aceleradores de redes convolucionales neuronales (CNN por sus siglas en inglés) se han consolidado como un dispositivo que completa las plataformas informáticas, desde aquellas de alto rendimiento hasta los sistemas embebidos. A lo largo de los años, tanto el mundo académico como la industria han propuesto numerosas organizaciones arquitectónicas para hacer frente al cálculo convolutivo masivo en el proceso de inferencia de las redes. Sin embargo, al igual que los procesadores de propósito general, los aceleradores de CNN se implementan utilizando transistores CMOS, y suelen tener elevados requisitos de almacenamiento on-chip de memoria estática de acceso aleatorio (SRAM) con el fin de almacenar en caché los parámetros de las CNN consistentes en pesos y activaciones de las distintas capas.

Dos de los principales efectos adversos que aceleran el desgaste de los transistores CMOS son la inestabilidad de temperatura de polarización negativa (NBTI) y la inyección de portador caliente (HCI). Estos efectos aumentan la ten-

sión umbral del transistor (V_{th}) a lo largo de la vida útil de un circuito digital, por lo que incrementan el retardo de conmutación del transistor, provocando fallos permanentes cuando las rutas críticas superan el tiempo de ciclo del procesador.

Por su diseño, las celdas SRAM son particularmente sensibles a los efectos de fallo NBTI y HCI, pues envejecen continuamente. NBTI degrada los transistores PMOS cuando un determinado valor lógico se almacena durante un largo periodo, mientras que HCI deteriora no sólo los transistores inversores de bucle NMOS cuando el valor lógico almacenado cambia, sino también los transistores de paso NMOS cuando se accede a su contenido. En general, estas situaciones están estrechamente relacionadas entre sí, lo que significa que combatir únicamente la NBTI podría agravar la HCI como efecto secundario, y viceversa.

El enfoque del estado del arte para la mitigación del envejecimiento en las memorias on-chip SRAM de los aceleradores CNN se centra únicamente en NBTI sobre memorias de pesos [2]. En cambio, este trabajo aborda no sólo

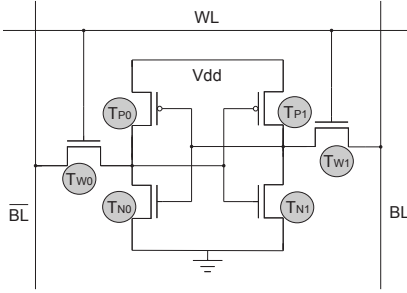


Figure 1: Implementación de una celda SRAM 6T, distinguiendo entre transistores del bucle inversor PMOS (T_{Pi}) y NMOS (T_{Ni}), así como transistores de paso NMOS (T_{Wi}).

NBTI sino también HCI en memorias on-chip de activaciones. Para ello, esta tesis realiza dos contribuciones principales:

- Se presenta un estudio exhaustivo de caracterización del ciclo de trabajo, cambios lógicos (flips) y patrones de acceso que las aplicaciones CNN actuales inducen a cada celda de memoria de activaciones.
- Se propone un novedoso mecanismo de microarquitectura que tiene en cuenta el envejecimiento, denominado Gated-CNN, el cual explota técnicas de power-gating y rotación de direcciones adaptado a los requisitos específicos de memoria y a las localidades temporales/espaciales de las aplicaciones CNN, además de la organización y gestión de memoria de los aceleradores CNN actuales.

2. Metodología

Esta sección presenta el modelo de envejecimiento junto con el acelerador CNN modelado

2.1. Modelo de Envejecimiento

La figura 1 representa la implementación de una celda SRAM típica formada por 6 transistores (celda 6T). Los cuatro transistores etiquetados como T_{Pi} o T_{Ni} forman un bucle inversor que almacena el valor lógico de la celda, mientras que los dos transistores T_{Wi} restantes actúan como transistores de paso. El fenómeno NBTI afecta principalmente a los transistores PMOS cuando se aplica un '0' lógico a sus puertas. En una celda 6T, el bucle invertido hace que T_{P0} y T_{P1} presenten una degradación NBTI complementaria, lo que significa que cuando un transistor

está bajo tensión, el otro se recupera parcialmente del efecto NBTI. Por ende, para mitigar la degradación NBTI, se requiere de un ciclo de trabajo equilibrado.

$$dV_{th_{NBTI}} = A_{NBTI} \times t_{ox} \times \sqrt{C_{ox} \times (V_{dd} - V_{t0})} \times \left(1 - \frac{V_{ds}}{\alpha_{NBTI} \times (V_{dd} - V_{t0})}\right) \times e^{\frac{V_{dd}}{t_{ox} \times E_{NBTI}} - \frac{E_a}{k \times T}} \times t_{stress}^{0.25} \times \left(1 - \sqrt{\frac{etha \times t_{rec}}{t_{stress} + t_{rec}}}\right) \quad (1)$$

La ecuación 1 muestra la fórmula estándar para calcular la degradación V_{th} (dV_{th}) de un transistor PMOS debida al fenómeno NBTI [7]. Los parámetros t_{stress} y t_{rec} denotan la cantidad de tiempo (en segundos) que el transistor está bajo los modos de estrés y recuperación, respectivamente. En una celda 6T, T_{P0} y T_{P1} acumulan tiempo t_{stress} cuando la celda almacena un '0' y un '1' lógicos, respectivamente.

HCI afecta principalmente a los transistores NMOS cuando se produce una transición de valor lógico en sus puertas. En una celda 6T, los transistores T_{N0} y T_{N1} se ven afectados si el valor lógico almacenado se invierte como consecuencia de una operación de escritura. Por otra parte, cada acceso a la celda induce desgaste HCI en los transistores T_{W0} y T_{W1} . Al contrario que en NBTI, no hay fase de recuperación. Por lo tanto, HCI se minimiza cuando se reduce el número de accesos a una celda y las escrituras no cambian el valor almacenado.

$$dV_{th_{HCI}} = A_{HCI} \times \alpha_{HCI} \times f \times e^{\frac{V_{dd} - V_{t0}}{t_{ox} \times E_{HCI}}} \times \sqrt{t} \quad (2)$$

La contribución del efecto HCI a dV_{th} de un transistor NMOS se calcula utilizando la ecuación estándar $dV_{th_{HCI}}$ [6]. El parámetro t se refiere a la cantidad de tiempo (en segundos) en que la puerta transiciona de '0' a '1' y viceversa. En una celda 6T, los transistores T_{N0} y T_{N1} acumulan tiempo t cuando el valor almacenado cambia, mientras que los transistores T_{W0} y T_{W1} lo hacen cuando la línea de palabras (WL) pasa de alto a bajo y viceversa.

Power-gating es una técnica bien conocida para reducir drásticamente el consumo de energía de las estructuras de memoria. Esta técnica ofrece la oportunidad de mitigar los efectos del envejecimiento mencionados.

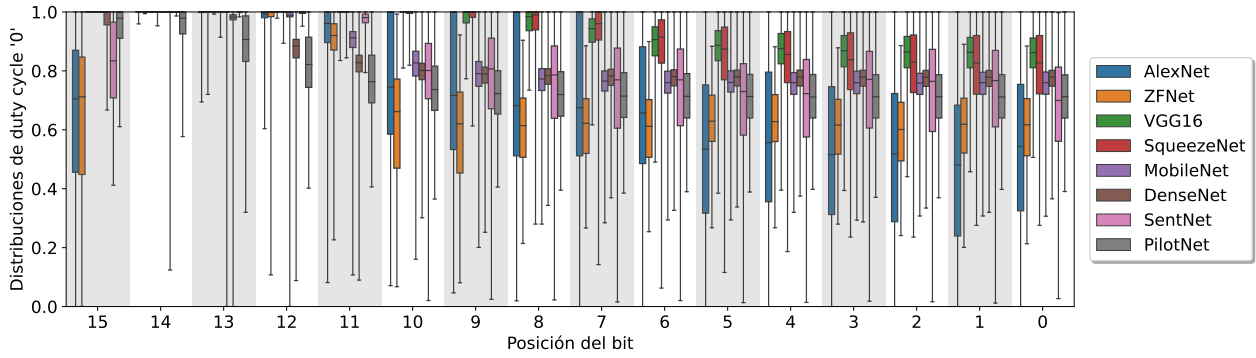


Figure 2: Distribuciones del ciclo de trabajo ‘0’ en las celdas agrupadas por posición de bit de las palabras activas de 16 bits.

Una configuración de power-gating que tiene en cuenta el envejecimiento consiste en el uso de un transistor de reposo (sleep transistor) NMOS de alto V_{th} que conecta la celda 6T a tierra. De este modo, el terminal de tierra de la celda se conecta a una tierra virtual. Cuando el transistor de reposo conduce corriente (estado activo), la celda funciona como de costumbre, pero con una tensión de tierra igual a la tierra virtual. Por el contrario, cuando el transistor de reposo está apagado, la celda está desconectada de la tierra y ambos transistores T_{P0} y T_{P1} permanecen parcialmente bajo recuperación de NBTI al mismo tiempo, ya que ambos nodos de la celda mantienen un ‘1’ lógico [1]. Por supuesto, mientras la celda permanezca apagada, no sólo se reducen las distribuciones de altos ciclo de trabajo para combatir NBTI, sino que tampoco se accede ni se voltea el contenido de la celda, lo que evita el desgaste por HCI. A diferencia de los transistores NMOS de la celda, el transistor de reposo NMOS es resistente a HCI, ya que se implementa utilizando un dispositivo de alto V_{th} [4, 5].

2.2. Acelerador CNN para Inferencia

Para estudiar el efecto del envejecimiento en los aceleradores de CNN actuales, así como contramedidas microarquitectónicas para reducir su impacto, modelamos una vista abstracta de un acelerador genérico en software. El modelo propuesto se basa en los modelos del estado del arte tanto del mundo académico como de la industria para acelerar el proceso de inferencia en las CNN.

La figura 4 muestra la organización del hardware del acelerador de referencia, el cual con-

sta de una matriz de elementos de procesamiento (PE) de 8×8 , un par de búferes de entrada/salida (E/S) de 2 MB para el almacenamiento de activaciones, un búfer de pesos de 2 MB, despachadores para cada búfer y una unidad de control.

Los búferes de E/S se dividen en bancos y sub-bancos, de forma similar a como se hace en [3]. En concreto, cada búfer consta de ocho bancos de 256 KB, y cada banco se compone de ocho sub-bancos de 32 KB.

3. Caracterización

La figura 2 muestra la distribución del ciclo de trabajo ‘0’ para cada celda de un búfer de activaciones, esto es, la fracción de ciclos en la cual cada celda almacena el valor lógico ‘0’ con respecto al total de ciclos; Para facilitar el análisis, se agrupan las celdas en función a la posición del bit dentro de la palabra de 16 bits, representada en little-endian. Las distribuciones incluyen los ciclos de trabajo ‘0’ para cada celda activa que almacena bits útiles. Es decir, la figura no incluye el efecto de los ciclos de trabajo que provocan máxima degradación del NBTI por estar en celdas de memoria inactivas. Los benchmarks consisten en ocho aplicaciones CNN ampliamente utilizadas con distintas demandas de cálculo y memoria. Las activaciones se representan utilizando un tipo de datos de punto fijo de 16 bits y el número de bits de la parte entera depende del benchmark, sin utilizar más que el mínimo necesario para un rendimiento óptimo. Aparte de AlexNet y ZFNet, se puede observar un ciclo de trabajo ‘0’ sesgado en el bit de signo (15º bit) para cada benchmark. El ciclo de trabajo ‘0’ también está sesgado en la mayoría de

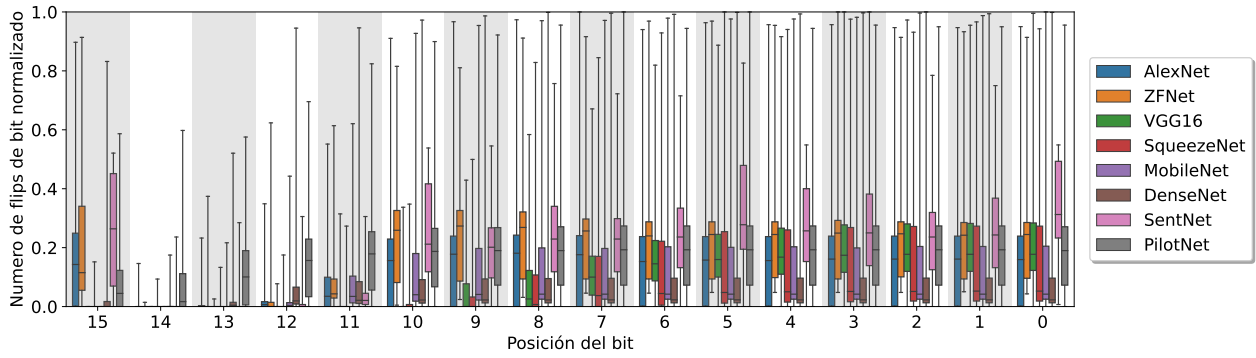


Figure 3: Flips normalizados por posición del bit con respecto a la celda del bit con el número de flip más alto.

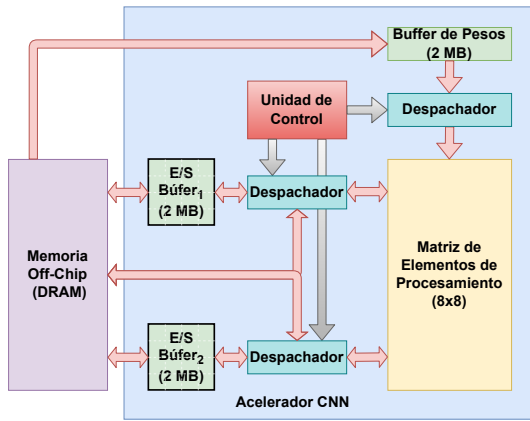


Figure 4: Visión general del acelerador CNN modelado

los bits enteros, y en particular en los más significativos, lo que pone de manifiesto que la mayoría de las activaciones son próximas a cero. En contraste con los bits de signo y enteros, los bits fraccionarios muestran una distribución del ciclo de trabajo más amplia, empujando la mediana hacia el ciclo de trabajo ideal del 50%. Sin embargo, en pruebas como VGG16 y SqueezeNet, la mediana del ciclo de trabajo '0' supera el 90% en todas las posiciones de bits. En general, el desgaste de NBTI, se aprecia no sólo en las celdas de signo y de bits enteros, sino también en muchas celdas que almacenan bits fraccionarios. Las técnicas anteriores de detección del envejecimiento para arquitecturas de CPU que trabajan con una granularidad de palabra serían ineficaces en aceleradores CNN. La figura 3 muestra el número normalizado de flips para cada celda de bit activa con respecto a la celda del peor caso, aquella que acumula el

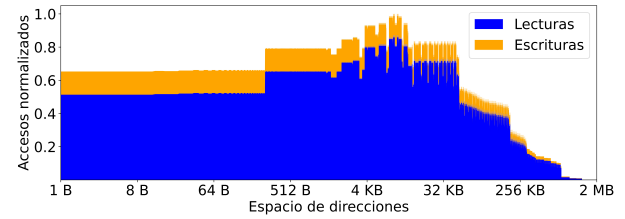


Figure 5: Número de accesos de DenseNet en todo el espacio de direccionamiento, normalizado con respecto a la dirección de memoria con el mayor número de accesos. El eje X se muestra en escala logarítmica con fines ilustrativos.

número de flips más alto para cada benchmark. Como era de esperar, los bits de signo y enteros, en los que el ciclo de trabajo suele estar muy sesgado, muestran un menor número de flips en comparación con los bits fraccionarios, en los que el ciclo de trabajo suele estar menos sesgado y se observa una mayor cantidad de flips. El patrón de flips muestra que pocas celdas sufren de un elevado número de flips, ya que todas las casillas se mantienen por debajo de un 50% de la distribución de flips. Sin embargo, esas casillas están repartidas entre todos los bits fraccionarios.

La figura 5 traza el número normalizado de accesos de lectura/escritura a cada byte de un búfer de E/S con respecto al byte con el número de acceso más alto. Los resultados se limitan a DenseNet. Las demás CNN muestran patrones de acceso similares.

La mayoría de los accesos están sesgados hacia las palabras de orden inferior, ya que cada capa de una red se almacena a partir de la dirección 0x0. Además, teniendo en cuenta todos los benchmarks estudiados, el número medio de

operaciones de lectura es 201 veces mayor que el de escritura, debido a la reutilización de la activación que presentan las convoluciones. Por lo tanto, los transistores de paso T_{Wi} están mucho más expuestos a la degradación por HCI que los transistores de bucle inversor T_{Ni} .

4. Propuesta: Gated-CNN

Esta sección presenta el diseño Gated-CNN propuesto, mostrando una visión general del enfoque. La funcionalidad detallada de los módulos puede encontrarse en el documento de la tesis, así como consideraciones detalladas sobre los sobrecostos de temporización, potencia, energía y área.

Basándose en el estudio de caracterización previo, Gated-CNN pretende minimizar los efectos de envejecimiento tanto de NBTI como de HCI en cada transistor de las celdas de los búferes de E/S de los aceleradores CNN. La idea clave de Gated-CNN es combinar un esquema de rotación de direcciones de bancos con un mecanismo de power-gating de bancos para distribuir no sólo los patrones de flips y accesos, sino también los ciclos de apagado, con el fin de equilibrar la distribución de los ciclos de trabajo en todos los bancos.

El componente Gated-CNN está acoplado a un búfer de E/S. Es decir, se necesitan dos componentes Gated-CNN para el acelerador CNN básico. Además, el enfoque propuesto está diseñado para un búfer de E/S con 8 bancos. Sin embargo, podría ser rediseñado para soportar un número diferente de bancos con cambios menores.

La figura 6 representa la arquitectura de los cuatro módulos principales de Gated-CNN. El módulo *banco efectivo* calcula el id de banco efectivo en el que se encuentra la activación solicitada en el búfer de E/S. Además, envía el id del banco inicial de la capa almacenada actualmente al módulo *rotación inter-bancos*. Este módulo identifica los bancos inicial y final necesarios para el siguiente paso de inferencia (la siguiente capa que se almacenará en el buffer) y envía esta información al módulo *generador de bitmap power-gating*. En función de dichos bancos, el generador de power-gating bitmap calcula un bitmap que indica qué bancos deben activarse/desactivarse en el siguiente paso de inferencia. Este bitmap se envía al módulo *actu-*

alización de bitmap. Este módulo actualiza el estado de activación/desactivación de los transistores de reposo de los bancos en tiempo de ejecución, ocultando la penalización por latencia de activación de bancos.

5. Evaluación Experimental

Esta sección cuantifica la reducción de los efectos de envejecimiento de los transistores que aporta la técnica Gated-CNN. En primer lugar, se presenta un análisis de los ciclos de trabajo, los flips y los patrones de acceso en las celdas de memoria con mayor degradación por envejecimiento. Luego, se cuantifica la reducción del envejecimiento con el modelo de envejecimiento descrito en la Sección 2.1.

5.1. Análisis de ciclo de trabajo

La figura 7a gráfica, para todos los benchmarks estudiados, el ciclo de trabajo '0' de las celdas correspondientes al peor caso, es decir, aquellas con el ciclo de trabajo '0' más largo. Gated-CNN incorpora un estado *apagado* para reflejar la cantidad de tiempo que una celda está apagada. El símbolo de la cruz en cada barra se refiere al valor medio del ciclo de trabajo '0' entre todas las celdas. BL corresponde a base-line un enfoque básico sin ningún esquema de mitigación del envejecimiento y GC al enfoque Gated-CNN. Al igual que en el estudio de caracterización anterior, asumimos de forma conservadora que los resultados de referencia sólo se refieren a las celdas activas que almacenan contenidos útiles. Por el contrario, Gated-CNN utiliza todas las celdas y, en consecuencia, todas ellas se incluyen en los resultados.

Como era de esperar, el peor ciclo de trabajo '0' para el baseline es del 100% en todos los benchmarks. Además, el ciclo de trabajo '0' medio también está muy sesgado, como ya se ha mencionado. Por otro lado, al apagar los bancos inactivos y distribuir uniformemente los ciclos de apagado entre ellos, Gated-CNN reduce el peor ciclo de trabajo '0' en un 50% o más para todos los benchmarks salvo AlexNet que lo hace en un 44%.

En general, Gated-CNN reduce el ciclo de trabajo '0' más alto en un 71%. Teniendo en cuenta todas las celdas, la reducción media del ciclo de trabajo '0' es del 85%.

Para realizar un análisis completo del efecto

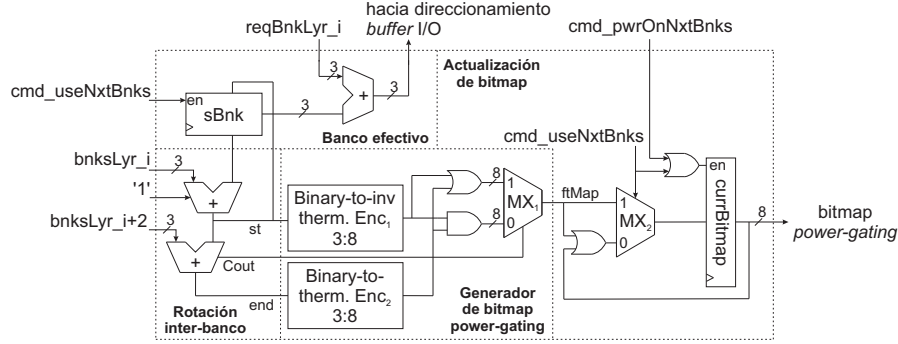
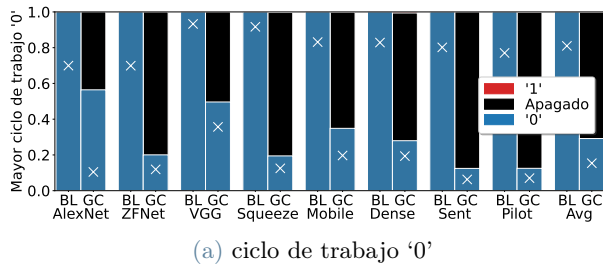
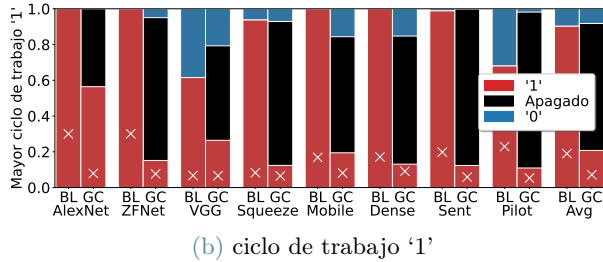


Figure 6: Diseño propuesto de Gated-CNN que consta de cuatro módulos principales: *banco efectivo*, *rotación inter-bancos*, *generador de bitmap power-gating* y *actualización de bitmap*.



(a) ciclo de trabajo '0'



(b) ciclo de trabajo '1'

Figure 7: celdas con mayor ciclos de trabajo para cada benchmark. El símbolo cruz representa los ciclos de trabajo medios entre todas las celdas. BL y GC se refieren al baseline y Gated-CNN.

NBTI, también debe tenerse en cuenta la distribución del ciclo de trabajo '1' más largo. La figura 7b muestra los resultados. A diferencia de las celdas del peor caso en el baseline con un ciclo de trabajo '0' del 100% (Figura 7a), las celdas con el ciclo de trabajo '1' más largo para el baseline no alcanza el 100% en todas las pruebas. Esta observaciones confirma la presencia de más ceros que unos en las activaciones. Para el diseño Gated-CNN, el porcentaje del estado de apagado en la distribución es el mismo que en el análisis anterior. Al igual que en la línea de base, el ciclo de trabajo máximo '1' se reduce aún más con la presencia de ceros.

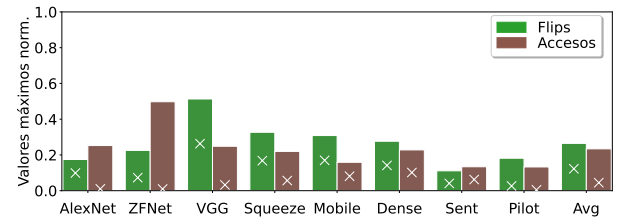


Figure 8: Valores máximos normalizados de flips y accesos de Gated-CNN con respecto al baseline. La cruz indica el promedio normalizado de cambios y accesos de bits de todas las celdas.

5.2. Análisis de accesos y flips

La figura 8 muestra el peor caso de flips y accesos en las celdas para el diseño Gated-CNN normalizadas con respecto al peor caso del baseline. Estos resultados sólo se refieren a las celdas activas que almacenan bits útiles en ambos enfoques.

Al rotar los datos entre los bancos, Gated-CNN reduce sustancialmente los valores máximos de flips y accesos en comparación con el baseline. La mayor reducción de flips y accesos se observa en SentimentalNet. Esto se debe principalmente a que todas las capas caben en un único banco, lo que impide que se produzcan flips en los bancos restantes. Otros benchmarks obtienen valores máximos más altos debido a que las capas más grandes ocupan varios bancos.

En general, los valores máximos más altos de volteo y acceso se reducen de media en un 74%. Si se tienen en cuenta todas las celdas, las reducciones medias alcanzan el 88% y el 96%, respectivamente.

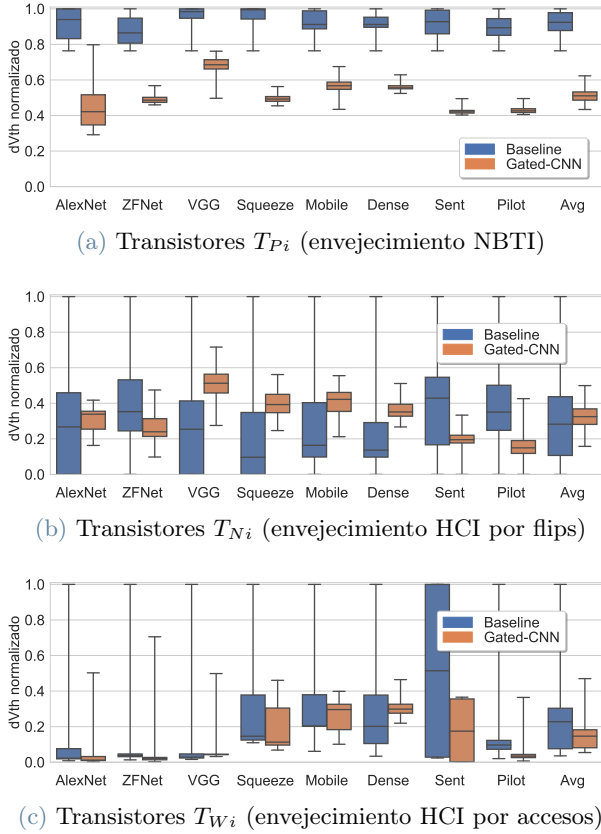


Figure 9: dV_{th} para cada transistor dentro de la celda, normalizada con respecto al máximo dV_{th} tras una vida útil de 3 años, distinguiendo entre distintos tipos de transistores (efectos del envejecimiento)..

5.3. Degradación de la tensión de umbral

La figura 9a muestra la distribución dV_{th} del baseline y Gated-CNN normalizadas con respecto al máximo dV_{th} después de 3 años en el baseline. Los resultados representados se refieren a la degradación NBTI que afecta a los transistores T_{Pi} . Como se observa, el enfoque propuesto reduce la tensión NBTI en todos los transistores, ya que las cajas y los bigotes se sitúan por debajo de los del baseline todos los benchmarks. Además, las alturas de las cajas referidas a Gated-CNN son más cortas que las del baseline en la mayoría de los puntos de referencia. Es decir, el enfoque Gated-CNN propuesto consigue una degradación NBTI más homogénea en todos los transistores en al menos el 50% de las celdas.

El ahorro de dV_{th} inducido por NBTI para los transistores T_{Pi} del peor caso oscila entre el 20 (AlexNet) y el 51% (SentimentalNet). La reduc-

ción media de dV_{th} para todos estos transistores es del 49%.

La figura 9b representa la distribución normalizada dV_{th} causada por el efecto HCI en los transistores T_{Ni} . A diferencia del efecto NBTI, algunas casillas de Gated-CNN se mantienen por encima de las del baseline como consecuencia de la dispersión de los flips en los transistores NMOS. En otras palabras, HCI se distribuye de forma más uniforme entre los transistores, lo que podría implicar percentiles de distribución por encima de los del baseline. No obstante, al igual que el efecto NBTI, la distribución HCI ayuda a Gated-CNN a obtener cajas más finas y a reducir en gran medida dV_{th} en los transistores correspondientes al peor caso. La reducción de dV_{th} inducida por HCI para los transistores T_{Ni} del peor caso oscila entre 28 (VGG) y 67% (SentimentalNet). El ahorro medio de dV_{th} para todos estos transistores es del 68%.

Por último, la Figura 9c se centra en el efecto HCI en los transistores T_{Wi} . Gated-CNN obtiene cajas más finas con respecto al baseline en la mayoría de los benchmarks. El peor caso de dV_{th} de los transistores se reduce sustancialmente. El ahorro de dV_{th} inducido por HCI para los transistores T_{Wi} correspondientes al peor caso varía entre el 29% (ZFNet) y el 64% (PilotNet); el ahorro medio de dV_{th} para todos estos transistores es del 85%.

6. Conclusiones

Esta tesis ha identificado los transistores de las celdas de memorias de activación SRAM on-chip de los aceleradores CNN que más envejecen desde la perspectiva de los efectos NBTI y HCI. Basándose en esta información, este trabajo ha presentado Gated-CNN, una novedosa técnica microarquitectónica que equilibra los ciclos de trabajo mediante el apagado de celdas específicas de memoria mediante power-gating, y distribuye uniformemente los patrones de acceso y flips entre todas las celdas con un mecanismo de rotación de direcciones.

Los resultados experimentales han demostrado que Gated-CNN prolonga la vida útil de los transistores de celda con un ahorro medio de degradación de V_{th} de al menos un 49% dependiendo del tipo de transistor.

References

- [1] A. Calimera et al. Analysis of NBTI-Induced SNM Degradation in Power-Gated SRAM Cells. In *Proc. IEEE ISCAS*, pages 785–788, 2010.
- [2] Muhammad Abdullah Hanif and Muhammad Shafique. DNN-Life: An Energy-Efficient Aging Mitigation Framework for Improving the Lifetime of On-Chip Weight Memories in Deep Neural Network Hardware Architectures. In *Proc. DATE*, pages 729–734, 2021.
- [3] Patrick Judd et al. Stripes: Bit-Serial Deep Neural Network Computing. In *Proc. MICRO*, pages 1–12, 2016.
- [4] M. T. Rahman et al. ARO-PUF: An Aging-Resistant Ring Oscillator PUF Design. In *Proc. DATE*, pages 1–6, 2014.
- [5] T. Siddiqua and S. Gurumurthi. Enhancing NBTI Recovery in SRAM Arrays Through Recovery Boosting. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, 20(4):616–629, 2012.
- [6] Abhishek Tiwari and Josep Torrellas. Facelift: Hiding and Slowing Down Aging in Multicores. In *Proc. MICRO*, pages 129–140, 2008.
- [7] R. Vattikonda et al. Modeling and Minimization of PMOS NBTI Effect for Robust Nanometer Design. In *Proc. DAC*, pages 1047–1052, 2006.



POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

On Microarchitectural Mechanisms to Combat Aging in On-Chip Mem- ories of Convolutional Neural Net- work Accelerators

TESI DI LAUREA MAGISTRALE IN
COMPUTER SCIENCE AND ENGINEERING - INGEGNERIA IN-
FORMATICA

Author: **Nicolás Landeros Muñoz**

Student ID: 10785675

Advisor: Prof. Davide Zoni

Co-advisors: Prof. Alejandro Valero Bresó

Academic Year: 2022-2023

Acknowledgements

Al Dr. Alejandro Valero y al Dr. Ruben Gran Tejero por su enorme apoyo y guía que me ha permitido crecer enormemente como profesional y a mi familia, en especial mi madre que gracias a su apoyo incondicional me ha permitido convertirme en la persona que soy hoy.

Abstract

Negative Bias Temperature Instability (NBTI) and Hot Carrier Injection (HCI) are two of the main reliability threats in current technology nodes. These aging phenomena degrade the transistor’s threshold voltage (V_{th}) over the lifetime of a digital circuit, resulting in slower transistors that eventually lead to a faulty operation when the critical paths become longer than the processor cycle time. Among all the transistors on a chip, the most vulnerable transistors to such wearout effects are those used to implement SRAM storage, since memory cells are continuously degrading. In particular, NBTI ages PMOS cell transistors when a given logic value is stored for a long period (i.e., a long duty cycle), whereas HCI does the same in NMOS cell transistors not only when the stored value flips but also when it is accessed. This work focuses on mitigating aging in the on-chip SRAM memories of Convolutional Neural Network (CNN) accelerators storing activations. This paper makes two main contributions. At the software level, we quantify the aging induced by current CNN benchmarks with a characterization study of duty cycle, flip, and access patterns in every activation memory cell. Based on the insights from this study, this work proposes a novel microarchitectural technique, Gated-CNN, that ensures a uniform aging degradation of every memory cell. To do so, Gated-CNN proposes power-gating and address rotation techniques tailored to the memory demands and temporal/spatial localities exhibited by CNN applications, as well as the memory organization and management of CNN accelerators. Experimental results show that, compared to a conventional design, the average V_{th} degradation savings are at least as much as 49% depending on the type of transistor.

Keywords— Access patterns, bit flip patterns, duty cycle, Hot Carrier Injection, Negative Bias Temperature Instability, threshold voltage degradation.

Abstract in lingua italiana

L'instabilità della temperatura di polarizzazione negativa (NBTI) e l'iniezione di portanti calde (HCI) sono due delle principali minacce all'affidabilità nei nodi tecnologici attuali. Questi fenomeni di invecchiamento degradano la tensione di soglia del transistor (V_{th}) nel corso della vita di un circuito digitale, determinando un rallentamento dei transistor che alla fine porta a un funzionamento difettoso quando i percorsi critici diventano più lunghi del tempo di ciclo del processore. Tra tutti i transistor di un chip, i più vulnerabili a questi effetti di usura sono quelli utilizzati per implementare la memoria SRAM, poiché le celle di memoria si degradano continuamente. In particolare, l'NBTI invecchia i transistor delle celle PMOS quando un determinato valore logico viene memorizzato per un lungo periodo (cioè un lungo duty cycle), mentre l'HCI fa lo stesso nei transistor delle celle NMOS non solo quando il valore memorizzato si capovolge ma anche quando vi si accede. Questo lavoro si concentra sull'attenuazione dell'invecchiamento delle memorie SRAM on-chip degli acceleratori di reti neurali convoluzionali (CNN) che memorizzano le attivazioni. Questo lavoro offre due contributi principali. A livello software, quantifichiamo l'invecchiamento indotto dagli attuali benchmark CNN con uno studio di caratterizzazione del duty cycle, del flip e dei modelli di accesso in ogni cella della memoria di attivazione. Sulla base dei risultati di questo studio, il lavoro propone una nuova tecnica microarchitetturale, Gated-CNN, che garantisce un degrado uniforme dell'invecchiamento di ogni cella di memoria. A tal fine, Gated-CNN propone tecniche di power-gating e di rotazione degli indirizzi adattate alle esigenze di memoria e alle località temporali/spaziali delle applicazioni CNN, nonché all'organizzazione e alla gestione della memoria degli acceleratori CNN. I risultati sperimentali mostrano che, rispetto a un progetto convenzionale, il risparmio medio di degrado di V_{th} è almeno del 49%, a seconda del tipo di transistor.

Parole chiave— Schemi di accesso, schemi di bit flip, duty cycle, Hot Carrier Injection, instabilità termica del bias negativo, degrado della tensione di soglia.

Contents

Acknowledgements	i
Abstract	iii
Abstract in lingua italiana	v
Contents	vii
Introduction	1
0.1 Context and Motivation	1
0.2 Objectives	3
0.3 Thesis Structure	3
1 Background	5
1.1 Artificial Intelligence	5
1.2 Neural Networks	7
1.2.1 Convolutional Neural Networks	8
1.3 CNN Accelerators	14
1.3.1 Temporal Architectures	14
1.3.2 Spatial Architectures	15
1.4 Reliability of CNN Accelerators	17
1.4.1 Categorization of System Faults	17
1.4.2 Fault Modeling in Neural Network Systems	18
1.4.3 Factors Influencing the Impact of a Fault in a Neural Network	18
1.4.4 Aging Effects and Power-Gating Opportunity	20
2 Framework	23
2.1 Datasets	23
2.2 Benchmarks	23
2.3 Modeled CNN Inference Accelerator	25
2.4 Software Tools and Hardware Environment	27

3	Characterization Study	29
3.1	Quantization Study	29
3.1.1	Implementation Details	29
3.1.2	Results and Discussion	31
3.2	Duty Cycles, Bit Flips and Memory Accesses	34
3.2.1	Implementation Details	34
3.2.2	Results and Discussion	41
4	Proposed Approach: Gated-CNN	45
4.1	Overview	45
4.2	Effective Bank Module	47
4.3	Inter-Bank Rotation Module	47
4.4	Power-Gating Bitmap Generation Module	48
4.5	Bitmap Update Module	49
4.6	Timing, Energy, Power, and Area Estimations	50
4.7	Implementation Details	50
5	Experimental Evaluation	51
5.1	Duty Cycle Analysis	51
5.2	Bit Flip and Access Analysis	53
5.3	Sensitivity Study to the I/O Buffer Size	54
5.4	Threshold Voltage Degradation	56
6	Related Work	59
6.1	Bit-Flipping Approaches	59
6.2	SRAM Cell Circuit	59
6.3	Rotation Schemes	60
6.4	Power-Gating	60
7	Conclusions and Future Work	63
7.1	Conclusions	63
7.2	Research Contributions	64
7.3	Future Directions	64
	Bibliography	67
A	Appendix A: Training and Testing of a Neural Network	77
B	Appendix B: Computational Cost of Convolutional Neural Networks during	

Inference	79
B.1 Depthwise Convolution Layer	80
B.2 Dense Layer	81
B.3 Pooling Layer	81
B.4 Activation function and normalization operation	81
C Appendix C: Systolic Arrays	83
C.1 Systolic Dataflows for Convolution	84
C.1.1 Weight Stationary	84
C.1.2 Output Stationary	85
C.1.3 Input Stationary	85
D Appendix D: Fault Injection	87
List of Figures	89
List of Tables	91

Introduction

This chapter introduces the reader to the context, motivation, and main goals of this work, as well as a description of the overall structure of the document.

0.1. Context and Motivation

The end of the Dennard scaling and the Moore's Law era is the major driver of the computer architecture community toward domain-specific accelerator chips. Unlike general-purpose processors, accelerators are optimized to handle a specific application domain while delivering a higher performance-to-power ratio under a limited chip budget. These domains include Artificial Intelligence (AI), such as computer vision, speech recognition, natural language processing, or autonomous driving, among others, whose applications are efficiently run with machine learning algorithms like Deep Neural Networks (DNNs).

DNN accelerators have consolidated as a commodity device that complements computing platforms from high-performance to embedded systems. Since the emergence of the DianNao accelerator [1], both academia and industry have proposed many architectural organizations to deal with the massive convolution computation in the inference process of Convolutional Neural Networks (CNNs) [2–12]. However, like general-purpose processors, CNN accelerators are implemented using CMOS transistors, and they usually demand high on-chip Static Random-Access Memory (SRAM) storage requirements to cache CNN parameters consisting of weights (synapses) and activations (neurons) of the different neural network layers.

Two of the main deleterious effects that speed up the CMOS transistor wearout are known as Negative Bias Temperature Instability (NBTI) and Hot Carrier Injection (HCI) [13]. These effects degrade (increase) the transistor's threshold voltage (V_{th}) over the lifetime of a digital circuit, and therefore, they increase the transistor's switching delay, resulting in permanent faults when the critical paths become longer than the processor cycle time.

By design, SRAM cells are particularly sensitive to both NBTI and HCI failure effects since they are continuously aging. NBTI degrades PMOS transistors when a given logic value is stored for a long period (i.e., a long duty cycle or on/off ratio), whereas HCI deteriorates not only NMOS loop inverter transistors when the stored logic value flips but also NMOS pass transistors when contents are accessed (i.e., on/off switching frequency). In addition, HCI may also degrade the

drain current of both NMOS and PMOS transistors [14]. Overall, these situations are strongly related to each other, meaning that combating solely NBTI might aggravate HCI as a side effect, and vice versa.

CNN parameters are usually represented with fixed-point data types, as opposite to floating-point, with the aim to reduce the computing and energy consumption requirements of CNN inference accelerators [1–4, 7, 9, 12]. Recent studies focusing on weights have shown that fixed-point data types are inherently more resilient to faults than floating-point counterparts [15]. Unlike weights, activations usually comprise a larger range of values, which can severely compromise the CNN accuracy even using fixed-point data types.

To check the previous claim, let’s refer to the following experiment in which permanent bit faults are randomly injected at different rates in either weights or activations represented with 16-bit fixed-point data (further details can be found in appendix D). Figure 1 shows the normalized accuracy of the inference process for three different CNNs under faults with respect to the fault-free original accuracy. Activations are more vulnerable to faults than weights, even for a percentage of faulty bits as low as 0.001%. In contrast, under faulty weights, accuracy does not degrade in ZFNet and SqueezeNet regardless of the studied fault rate.

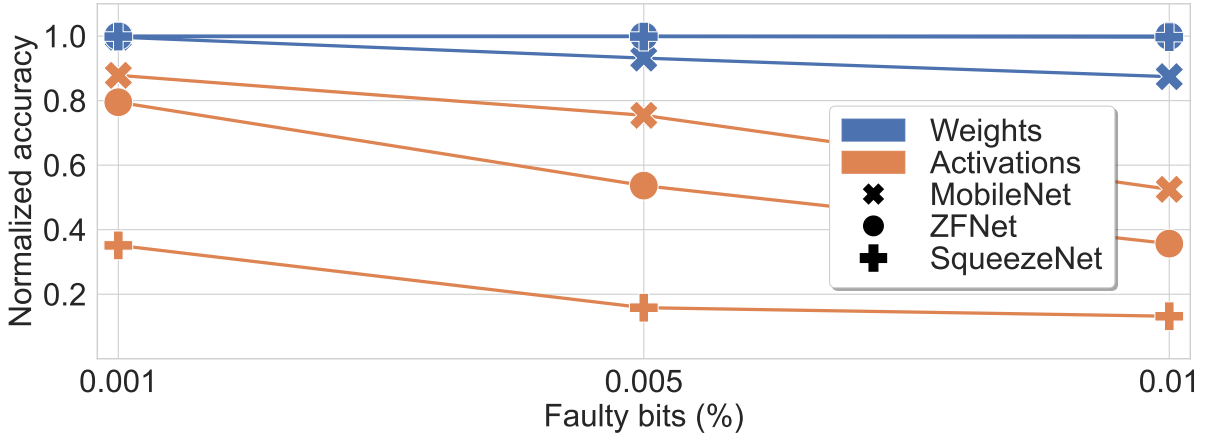


Figure 1: Normalized accuracy after injecting permanent faults in either weights or activations with respect to the original accuracy without faults. The number of faulty bits is calculated as a percentage of the largest CNN layer and it is the same for both weights and activations. The plotted results refer to the average accuracy after 100 trials.

The state-of-the-art approach for aging mitigation in on-chip SRAM storage of CNN accelerators solely focuses on NBTI in memories storing weights [16]. In contrast, this work will address not only NBTI but also HCI in on-chip memories of CNN accelerators storing activations.

0.2. Objectives

This work pursues two main goals:

- Present a comprehensive study characterizing the duty cycle, flips, and access patterns that current CNN applications induce in each activation memory cell.
- Based on the above study, propose a novel microarchitectural mechanism, namely Gated-CNN, tailored to the specific memory requirements and temporal/spatial localities of CNN applications, in addition to the memory organization and management of current CNN accelerators.

These two objectives can be divided into a number of milestones. These milestones allow to clarify the work carried out in this thesis and establish concrete tasks necessary to the development of this project. The specific milestones, in chronological order, are as follows.

- Study and selection of an experimental framework, including software to develop this work, configuration of simulation infrastructure, neural networks as benchmarks, datasets, and state-of-the-art CNN hardware accelerator structure.
- Characterization of the minimum requirements in terms of bit precision and memory of each benchmark.
- Modeling the dataflow of the chosen state-of-the-art accelerator in software.
- Simulation of inference process in the accelerator, leading to a characterization study of duty cycles, bit flips, and access patterns that benchmarks cause in the accelerator memories.
- Design and validation of microarchitectural mechanisms to reduce aging in activation memories of CNN accelerators.
- Analysis of experimental results, including the exploitation of a state-of-the-art analytical aging model to quantify the impact of duty cycles, flips, and access patterns on the activation memories of the CNN accelerator.

0.3. Thesis Structure

Excluding the introductory chapter, the remaining of this document is organized in seven chapters as follows.

- **Chapter 1** provides a background on artificial intelligence, convolutional neural networks, underlying hardware to support the execution of these networks, and aging-related problems in this hardware.

- **Chapter 2** describes the simulation framework, including a description on datasets, benchmarks, modeled state-of-the-art CNN accelerator, required software tools, and hardware environment used to develop the presented work.
- **Chapter 3** presents a characterization study of aging effects in CNN accelerators. The conclusions drawn in this chapter drive the proposed microarchitectural mechanisms of this work.
- **Chapter 4** proposes a couple of anti-aging microarchitectural solutions, consisting of power-gating and address rotation techniques.
- **Chapter 5** discusses experimental results, including detailed analysis of duty cycles, bit flips, and access patterns, as well as quantifying their impact on transistor aging and the benefits brought by the proposed design.
- **Chapter 6** comments on related work, classifying previous work into bit-flipping approaches, cell circuit modifications, rotation schemes, and power-gating solutions.
- **Chapter 7** summarizes the main conclusions of this work, future directions, and a list of publications of the candidate.

1 | Background

This chapter provides an introduction to artificial intelligence, neural networks with an emphasis on Convolutional Neural Networks , hardware accelerators to support the inference process of such networks, and reliability issues associated to those accelerators.

1.1. Artificial Intelligence

In recent years, the term Artificial Intelligence (AI) has achieved a growing popularity among the general public, although it is not a new concept as it was first used in 1956 by John McCarthy and deeply researched since then. Nowadays, AI has been widely adopted by the public thanks to organizations like OpenAI that have provided public access to AI-based technologies such as ChatGPT and DALL-E.

Machine Learning (ML) is one of the most developed branches of AI and is defined as the set of techniques and algorithms focused on finding patterns in data and using them to make predictions. These algorithms are divided into three main categories:

- **Supervised learning:** given a set of data and a set of labels, learn the mapping between data and label.
- **Unsupervised learning:** given a set of data, exploit its characteristics and learn a new suitable representation.
- **Reinforced learning:** given a set of rewards based on a set of actions, learn what actions to perform to maximize the rewards.

Figure 1.1 plots a classification of ML algorithms according to the above types of learning methods.

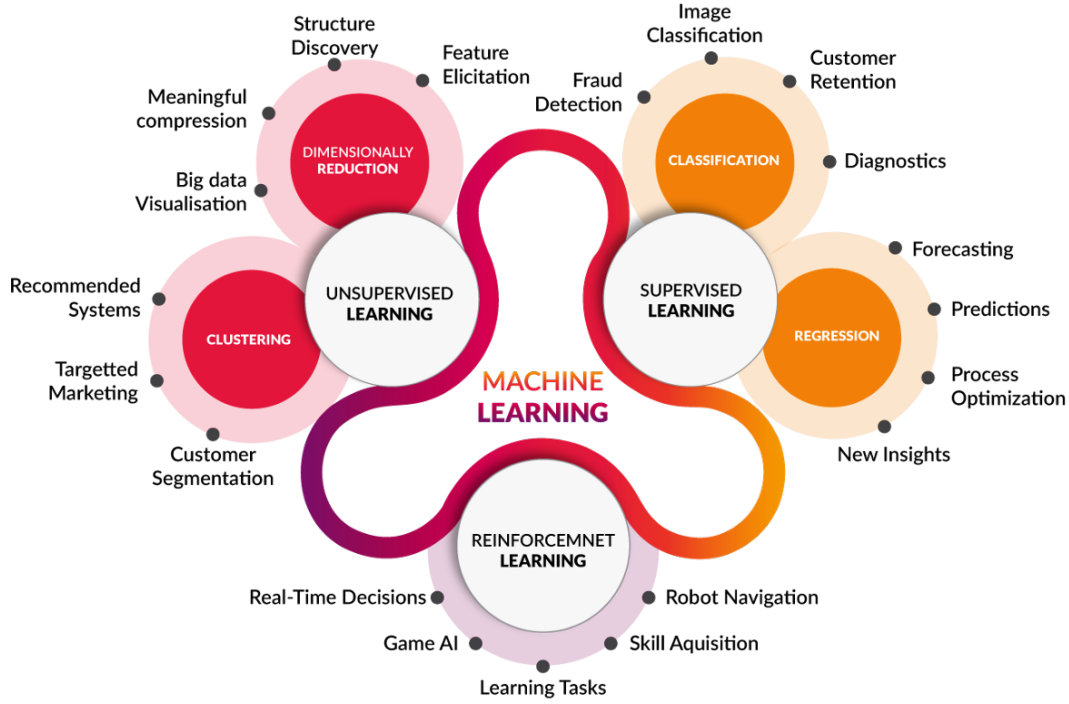


Figure 1.1: ML algorithms classified into supervised, unsupervised, and reinforced learning methods [18].

Tom Mitchell describes a ML program as “a program that learns from an experience $\mathbf{E}$, with respect to a task $\mathbf{T}$ and a performance measure $\mathbf{P}$. In turn, its performance on $\mathbf{T}$, as measured by $\mathbf{P}$, improves based on $\mathbf{E}$ ” [17].

- The task $\mathbf{T}$ depends on the type of learning; in the case of supervised learning, it can be the prediction of a numerical value (regression) or the class membership (classification).
- Experience $\mathbf{E}$ is usually referred to as data.
- The performance measure $\mathbf{P}$ is also known as the error or loss function.

We will focus on the supervised learning method, which consists of two different phases, namely feature extraction, the process of extracting the relevant information from the data for the task at hand, and classification, the process of finding a relationship between the extracted features and target labels.

Originally, the feature extraction phase was performed manually; however, since the ideal features are not always known, an approach arose in which the features were learned as well as the classification; this approach is called Deep Learning and works mainly on the basis of artificial neural networks.

1.2. Neural Networks

A neural network is a collection of layers of neurons connected to each other. Its name derives historically from the brain structure as it tries to mimic the way neurons communicate. Figure 1.2 shows typical schematics of neural networks, distinguishing between a generic neural network with input, hidden, and output layers, and a specific CNN with convolution, pooling, and fully-connected (classifier) layers.

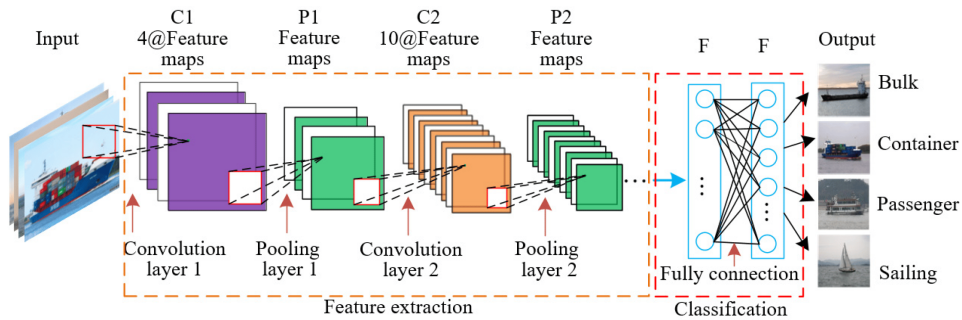
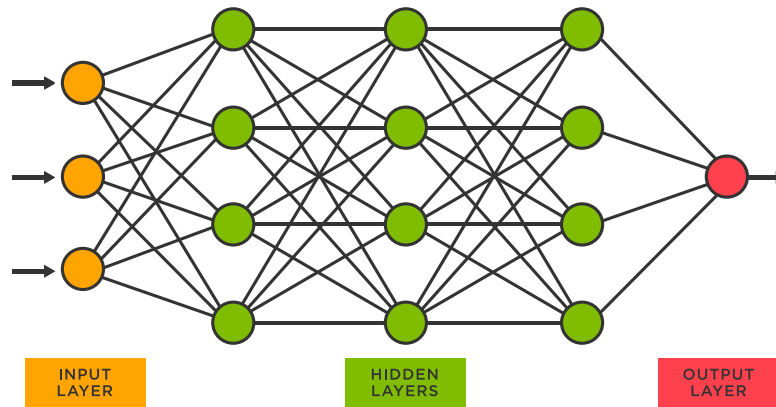


Figure 1.2: Examples of neural network schematics.

Figure 1.3 depicts the internal structure of a neuron and involved computations. The outputs of each layer are obtained as a linear combination of weighted outputs of the previous layer. To provide non-linearity and distinguish a neural network from a simple linear regression each outputs pass through a non-linear function known as the activation function.

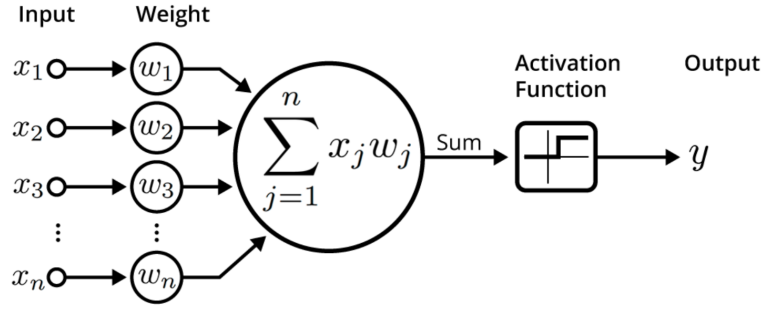


Figure 1.3: Neuron structure with the main involved calculations [21].

Over the years, multiple types of layers have been developed to extract different features of input data and provide accurate predictions, according to the connection between neurons they can be classified into three types:

- **Convolutional:** each neuron is connected to a neighborhood of the outputs of the previous layer. They are typically used for the process of extracting features.
- **Dense or fully-connected:** each neuron is connected to all the outputs of the previous layer. They are typically used for the classification process.
- **Recurrent:** each neuron is also connected to its past values. They are a dynamic extension of the neuron concept and are typically used when the nature of data is of sequential type (e.g., video or text).
- **Skip/shortcut connections:** each neuron is also connected to neurons in layers preceding the one immediately prior to it.

Figure 1.2b displays a typical architecture of a CNN trained for classification, in which relevant features of the input image are extracted by means of convolutional layers, and then these features are used by dense layers to classify the image according to one of the existing categories.

In order for a neural network to act as a classifier and provide accurate predictions, which is referred to as the inference process of a neural network, it must be trained first. See Appendix A for details about the training process.

1.2.1. Convolutional Neural Networks

This section describes the main operations carried out specifically in CNN networks. To do so, the section is organized in different subsections according to the types of layers described above for CNNs.

Convolution Layer

The inputs and outputs of a convolution layer can be represented as a three-dimensional matrix (four in the case of batch processing). In particular, for an image, the input dimensions are height $\times$ width $\times$ number of channels.

A convolution operation on an image is a local transformation in which an output image is created by means of a kernel K that moves through the input image. Figure 1.4 illustrates the operation of convolution in a 3×3 image with a single channel and a 2×2 kernel. The multi-channel extension implies that the kernel K has as many channels as the input. Highlighted in blue can be seen the involved values required to compute the output value located in the top-left corner of the output matrix. With regard of neural networks, the images resulting from a convolution are called activation maps or feature maps because they represent features found in the original image.

Input		Kernel		Output																	
<table border="1" style="border-collapse: collapse;"> <tr><td>0</td><td>1</td><td>2</td></tr> <tr><td>3</td><td>4</td><td>5</td></tr> <tr><td>6</td><td>7</td><td>8</td></tr> </table>	0	1	2	3	4	5	6	7	8	*	<table border="1" style="border-collapse: collapse;"> <tr><td>0</td><td>1</td></tr> <tr><td>2</td><td>3</td></tr> </table>	0	1	2	3	=	<table border="1" style="border-collapse: collapse;"> <tr><td>19</td><td>25</td></tr> <tr><td>37</td><td>43</td></tr> </table>	19	25	37	43
0	1	2																			
3	4	5																			
6	7	8																			
0	1																				
2	3																				
19	25																				
37	43																				

Figure 1.4: Convolution operation involving a 3×3 input image and a 2×2 kernel. The result is a 2×2 output image [22].

The generalized mathematical expression for the convolution of an image is shown in Equation 1.1, where O and I are the output and input images respectively, $0 \leq w \leq W - 1$ the width index, $0 \leq h \leq H - 1$ height, K is the kernel, k_w and k_h are the kernel dimensions, C_i corresponds to the number of channels of the input image, and finally, b refers to the bias term. K and b are learned during the training process of the network (see Appendix A for further details).

$$O[w][h] = b + \sum_{x=0}^{k_w-1} \sum_{y=0}^{k_h-1} \sum_{z=0}^{C_i-1} I[i+x][j+y][z] \cdot K[x][y][z] \quad (1.1)$$

A convolutional layer performs multiple convolutions to the input activation map through multiple kernels, resulting in an output activation map with a number of channels equal to the number of kernels used. Table 1.1 presents the parameters and hyper-parameters used during this thesis in the context of a convolutional layer.

Table 1.1: Parameters used in the context of a convolutional layer

Parameter	Description	Note
I	Input feature map	
N	Number of samples	
C_i	Number of input channels	
H_i	Input height	
W_i	Input width	
K	Kernel	Also known as filters
F	Number of kernels	
k_h	Kernel height	
k_w	Kernel width	
s_h	vertical stride	The step with which kernels are vertically shifted across the input feature map.
s_w	horizontal stride	The step with which kernels are horizontally shifted across the input feature map.
p_h	vertical padding	Number of rows with zero value added to the outline of the input feature map.
p_w	horizontal padding	Number of columns with zero value added to the outline of the input feature map.
O	Output feature map	
C_o	Number of output channels	Obtained as $C_o = F$
H_o	Output height	Obtained as $H_o = \lfloor \frac{H_i - k_h + 2p_h}{s_h} \rfloor + 1$
W_o	Output width	Obtained as $W_o = \lfloor \frac{W_i - k_w + 2p_w}{s_w} \rfloor + 1$

Therefore, the generalized expression for a convolutional layer is given in equation 1.2.

$$O[n][w][h][c_o] = B[c_o] + \sum_{x=0}^{k_w-1} \sum_{y=0}^{k_h-1} \sum_{z=0}^{C_i-1} I[n][i + s_w x][j + s_h y][z] \cdot K[n][x][y][z] \quad (1.2)$$

Where $0 \leq n \leq N - 1$ is the sample index and $0 \leq c_o \leq F - 1$ the output channel index.

The above definitions are not only valid for two-dimensional convolutions, but also one-dimensional convolutions (kernels moving in one dimension, usually used in text) and it can be extended to three-dimensional convolutions (kernels moving in three dimensions, mainly used in 3D image processing).

Alternatively, a convolution can be performed as a matrix multiplication, to which a relaxed form of the Toeplitz matrix is used; Figure 1.5 shows how the convolution of an image is transformed into a multiplication between two matrices (filter and input feature map). The main disadvantage of this approach is that it resorts to redundant data when constructing the

matrix, which produces inefficiency in storage and possibly a complicated access pattern.

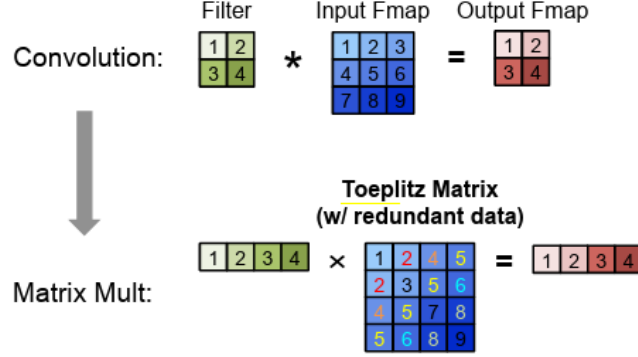


Figure 1.5: Convolution as matrix multiplication [23].

Depth-wise Convolution Layer

This is a variation of the original convolution layer with the aim of reducing the number of computational operations. For this purpose, each filter operates in only one channel. The output consists of the concatenation of the feature map obtained from each convolution. As a consequence, the number of filters is no longer a parameter that can be chosen by the network designer, since it always corresponds to the number of input channels and the number of channels in the output is always the same that the number of channels in the input.

Equation 1.3 details the computation of the output activations before passing through the normalization function, notice that c indexes both the input and output channels. A graphical interpretation of a Depth-wise convolution is shown in Figure 1.6.

$$O[n][i][j][c] = b[c] + \sum_{x=0}^{k_w} \sum_{y=0}^{k_h} I[n][i + Sx][j + Sy][c] \cdot K[n][x][y][c] \quad (1.3)$$

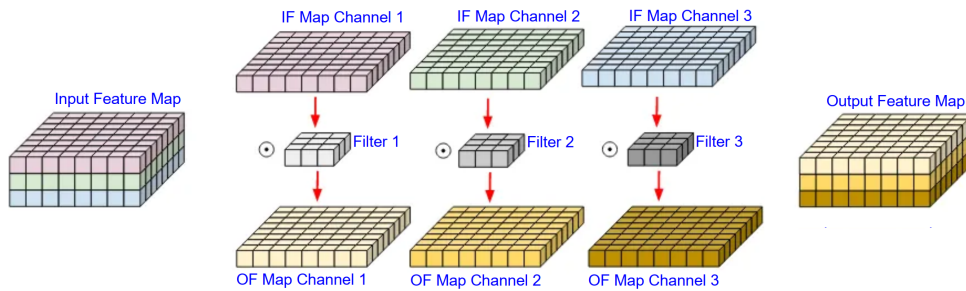


Figure 1.6: Depth-wise convolution over a three channel input. [24].

Lastly, the depth-wise convolution can be viewed and handled as C concatenated convolutions with $F = 1$ and the same input feature map.

Pooling Layer

Contiguous pixels within images often have redundant information that unnecessarily increases the computational cost and storage demand. To avoid this, networks often make use of subsampling, either by convolutional layers with stride > 1 or through the use of a pooling layer.

Pooling consists of taking a neighborhood of inputs (patch) and condensing all its information into a single output. There are two typical ways of performing pooling:

- **MaxPooling:** the output of a patch corresponds to the maximum value of the activations within it.
- **AveragePooling:** the output of a patch corresponds to the average value of the activations within it.

Figure 1.7 illustrates how MaxPooling works. Note that unlike a convolution layers, there is no overlap between patches. The pooling layer has no weights to be trained and is computationally inexpensive compared to a convolution, which is why it is often preferred when decreasing the dimensionality of an activation map.

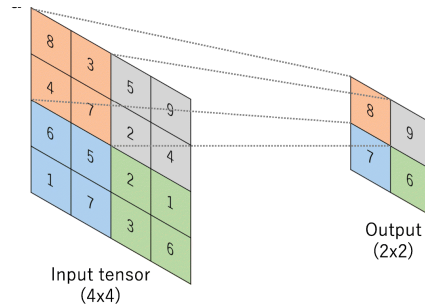


Figure 1.7: MaxPooling operation transforming a 4×4 activation map into a 2×2 map [25].

Dense Layer

Also known as fully-connected layer, these are layers in which each output depends on all the activations of the previous layer. A dense layer is defined by the number of activations at its input N and the number of activations at its output M , the number of weights therefore corresponds to an $M \times N$ matrix. Equation 1.4 provides the formula for the resulting activations of a dense layer prior to the application of the activation function. O and I correspond to the output and input activations, respectively, W to the weights and b to the bias.

$$O[i] = \sum_{j=0}^N I[j] \cdot W[i][j] + b[i] \quad (1.4)$$

Analyzing the above equation, a dense layer is simply a multiplication between the input vector and a matrix of weights. Figure 1.8 shows this operation graphically.

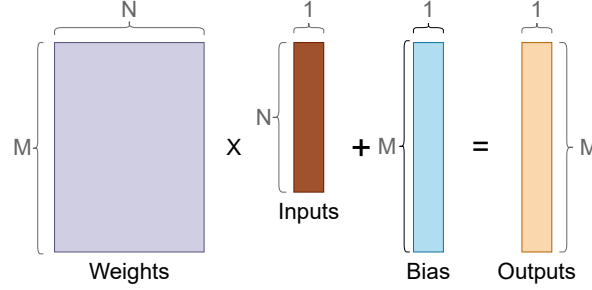


Figure 1.8: Dense layer as a matrix multiplication.

Lastly, a dense layer can be seen as a convolution layer where the filters are of the same size as the input width and height using a symmetric stride equal to 1, this implies that $(k_w, k_h) = (W_i, H_i) \wedge s_h = s_w = 1$ which produces a feature output map of dimensions $W_o = 1, H_o = 1, C_o = M$

Activation Function and Normalization

The activation function is a function applied to the output of a layer to provide non-linearity. Historically, several types have been used, such as the sigmoid or tangent hyperbolic function, but due to its computational simplicity, the predominant function is the Rectified Linear Unit (ReLU) function, whose formula is shown as Equation 1.5.

$$f(x) = \max(x, 0) \quad (1.5)$$

Batch normalization is a technique used to accelerate the learning process. It was originally presented in [26] and consists of, given a four-dimensional input, like (B, H, W, C) , the data are normalized with respect to an axis, for instance, channels (C) . During learning, the data are normalized, then shifted and scaled by two learned parameters. During inference, normalization is performed with mean and variance values obtained during training (Equation 1.6a), then displacement and scaled with the learned parameters (Equation 1.6b).

$$I[b][h][w][c] = \frac{I[b][h][w][c] - \mu_{mov}[c]}{\sigma_{mov}[c]}, \quad (1.6a)$$

$$O[b][h][w][c] = \alpha I[b][h][w][c] + \beta. \quad (1.6b)$$

Interested readers are referred to Appendix B, which covers the computational cost of each type of operation required for the use of neural networks.

1.3. CNN Accelerators

Due to the popularity of CNN-based applications, in recent years, multiple hardware architectures have been adopted for neural network computation, distinguishing between two different types: temporal and spatial architectures.

A temporal architecture is composed of multiple computational units under a centralized control and direct communication with the memory hierarchy. On the other hand, in spatial architectures, the control is distributed and the computational units incorporate a small memory unit (e.g., a register file) to store data and communicate with each other.

CPUs and GPUs are well known examples of temporal architectures where the computational units are known as ALUs. Spatial architectures are based on FPGA or ASIC designs, and their computational units are usually termed as Processing Elements (PEs); some recent examples can be found in [1, 3–10, 12].

1.3.1. Temporal Architectures

Matrix multiplication has historically been one of the most studied computational problems, which is why there exist multiple optimized libraries that implement this operation, both CPU (e.g., OPEN-BLAS or Intel MLK) and GPU (e.g., cuBLAS or cuDNN); therefore, in temporal architectures, the convolution and dense layers are usually processed by their matrix representation.

Both CPU and GPU systems are general-purpose chips that must support a wide range of applications. Therefore, it is unlikely the optimization of these systems at hardware level for efficient CNN computation. However, in recent years, some attempts have been made. For instance, Intel introduced DL Boost as a special instruction set to accelerate neural network computation. On the other hand, Nvidia introduced tensor cores as a new type of on-chip cores that accelerate long matrix operations and perform mixed precision MACs (MMACs) in a single operation.

The main disadvantage of a CPU lies in its low throughput due to the limited number of ALUs operating in parallel, which makes this platform an undesirable choice for both CNN training and inference processes. On the other hand, GPUs deliver a much higher throughput, but their high power consumption and cost make them a poor choice, especially for inference in power-constrained embedded environments.

1.3.2. Spatial Architectures

FPGA and ASIC systems belong to spatial architectures. The former benefit from their reprogrammability, simple design flow, and relatively low cost. However, their high flexibility makes them less energy-efficient than ASICs. This is the reason why they are usually used for prototyping. In contrast, ASICs have a high complexity and development cost, but they usually deliver higher performance and energy efficiency.

As can be seen in Figure 1.9a, the typical design of a spatial architecture consists of a global on-chip SRAM buffer, an array of PEs, and external DRAM memory. The global buffer and the PE array usually communicate through input/output FIFOs (iFIFO/oFIFO). The PEs are interconnected through a Network-on-Chip (NOC). Each PE is composed of an ALU capable of executing a MAC operation, a tiny register file used as a local scratchpad, and a PE FIFO (pFIFO) used to control the incoming and outgoing data traffic of the PE. The above description implies that there are four levels of memory, as depicted in Figure 1.9b, varying from register file accesses with the lowest latency to off-chip DRAM accesses with the highest latency, typically two or three orders of magnitude higher.

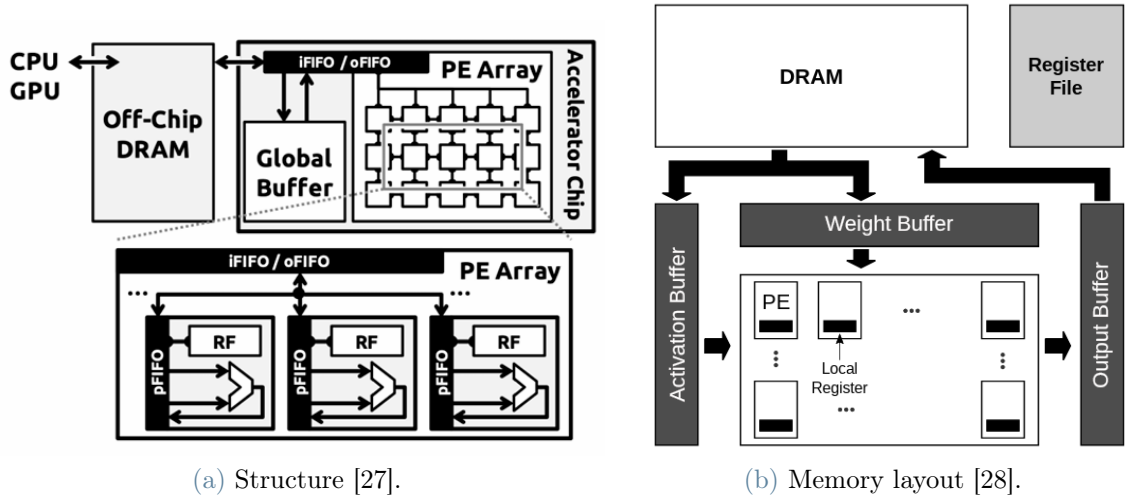


Figure 1.9: Typical structure and memory layout of spatial CNN accelerators.

Considering that each MAC input consists of a weight and an activation, together with the fact that these data may have different numerical precision requirements, many state-of-the-art CNN accelerators implement physically different on-chip buffers for each type of data. In addition, activations in a CNN network usually have a tetra-dimensional shape during feature extraction, consisting of the number of samples N , number of channels C , height H and width W . To store them in memory, they have to be vectorized. There are three common ways to do this, namely NCHW, NHWC, and CHWN, and described graphically in Figure 1.10.

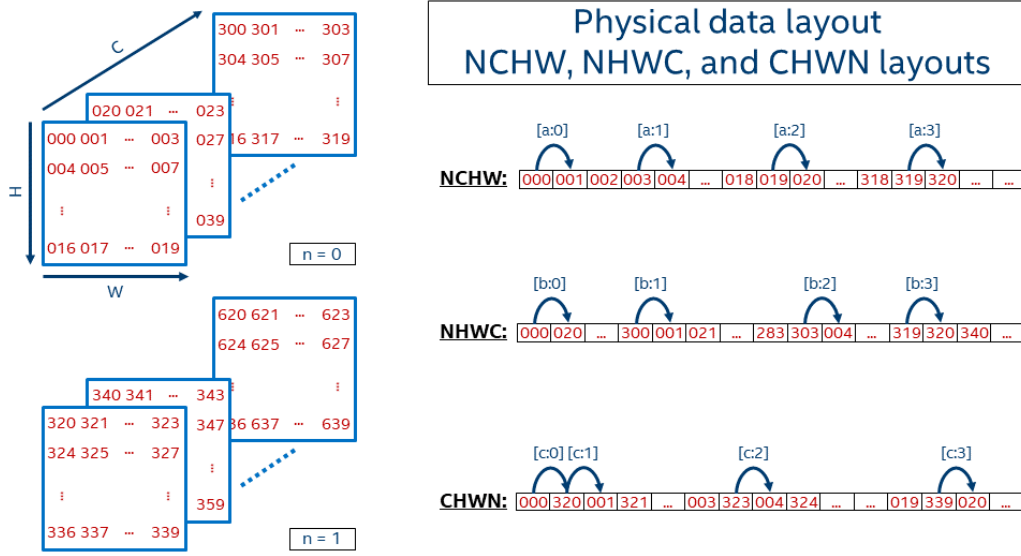


Figure 1.10: Three different ways to vectorize a 4-D feature map into a 1-D vector [29].

Figure 1.11 shows an academic state-of-the-art accelerator for CNN inference, namely Eyeriss, which includes an array of 168 PEs and a global buffer of 108 KB [6]. The figure also highlights the use of RLC encoding to reduce the number of DRAM accesses.

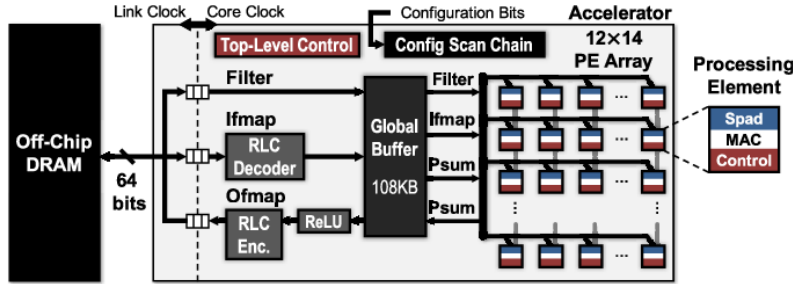


Figure 1.11: Structure of Eyeriss accelerator [6].

The Tensor Processing Unit (TPU) is an example of a commercially available CNN accelerator used in Google's data centers for training and inference processes [5]. Most of the calculations in TPU are carried out in the Matrix Multiply Unit (or PEs) colored yellow in Figure 1.12. The input data to this unit are provided by the blue SRAM buffers labeled as Weight FIFO and Unified Buffer, which store, respectively, weights and input activations of the i -th layer of the executed network. Output activations, which correspond to the input activations of the $i + 1$ -th layer, are stored in the accumulators. These output activations undergo certain transformations according to the functions implemented by the activation, normalization, and pooling units before returning to the Unified Buffer.

Both the PE array in Eyeriss and the Matrix Multiply Unit in the TPU implement a systolic array. The reader is referred to Appendix C for further details about the organization

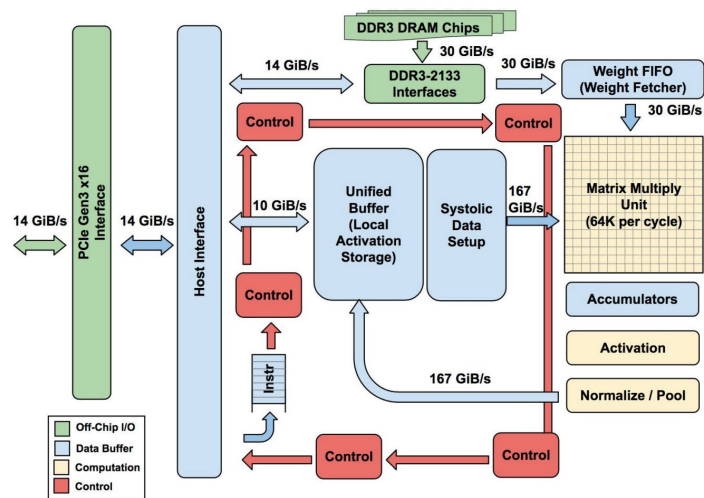


Figure 1.12: Structure of TPU [5].

and dataflow of systolic processor arrays.

1.4. Reliability of CNN Accelerators

In the context of digital circuits, reliability refers to the quality of such systems to tolerate faults during their operation. Reliability has been extensively studied for all kinds of computing systems, and those running neural networks are not an exception, especially when nowadays neural networks are being deployed in safety-critical environments like autonomous driving.

The remainder of this section categorizes different types of faults, comments on how faults are usually modeled in the context of neural networks, what are the main characteristics of neural networks that influence on the impact of faults, and finally, describes the two main transistor degradation effects that cause faults.

1.4.1. Categorization of System Faults

According to their behavior, faults can be classified as:

- **Permanent faults:** those that are always present from the moment they originate, generally due to process variations or hardware aging.
- **Transient faults:** these faults only affect the operation of the system for a limited period of time, usually due to an external agent such as an ionizing particle or a high operating temperature.
- **Intermittent faults:** those that occur periodically, usually due to an underlying cause, such as low operating voltage.

On the other hand, according to their manifestation, faults are divided into four types:

- **Benign faults:** those that although present, they do not have an effect on the operation or performance of the application (e.g., accuracy of a CNN). These faults are said to be masked.
- **Corrected faults:** detected and corrected by the system, typically by Error Correction Codes (ECCs).
- **Detected Unrecoverable Errors (DUEs):** those that are detected and therefore the system is aware of their existence but cannot do anything to correct them.
- **Silent Data Corruptions (SDCs):** undetected faults that affect system performance; thus, it is the most dangerous type of fault.

This work focus is mitigating SDC resulting from permanent faults due to system aging.

1.4.2. Fault Modeling in Neural Network Systems

Since there are multiple platforms capable of running neural networks, each with its own fault model, an abstract model for expressing errors is imperative. The typical approach used in most research studies is to model faults directly in the neural network by injecting them in the activation values or weights, which emulates faulty effects on memory or PE/ALU of the underlying hardware. There are two ways to model those faults when they are present either in memory or computational units:

- **Bit flip:** changes the logic value in one (or several) bit, which typically represents transient faults.
- **Stuck-at:** models permanent faults consisting of one (or several) bit always having the same logic value.

1.4.3. Factors Influencing the Impact of a Fault in a Neural Network

Two of the main characteristics of neural networks that influence the impact of faults on the accuracy are the data representation of the network parameters and type and depth of the affected neural network layer.

The impact of a faulty bit on the accuracy of a CNN heavily depends on how much the affected value (either an activation or a weight) has been modified by it; of course, a bit fault that modifies the fractional part is much less significant than one that affects the integer part. In the

context of neural networks, depending on the underlying hardware, one of the following data types is typically employed:

- **Fixed-point:** generally used in ASIC accelerators, they consist of a pre-set number of bits dedicated to the integer and fractional parts, including an additional bit for the sign. In particular, the number of bits dedicated to the integer part will determine how large a numerical deviation can be, while the proportion between fractional and integer bits will determine how likely it is that a faulty bit affects accuracy.
- **Floating-point:** typically used in general-purpose processors (e.g., CPUs or GPUs), it represents a dynamic range of numbers using the standard IEEE-754 format with sign, exponent, and mantissa. A fault in the sign, and especially in the exponent, most likely will affect performance, whereas the impact of a fault in the mantissa mainly depends on the value of the exponent.

Prior works such as [30, 31] have shown that faults in early layers of a neural network are less likely to cause a noticeable effect on accuracy than later layers; this is mainly because the different operations performed on the data from an early layer to a later layer can have the effect of masking a failure, for example:

- **Activations:** ReLU has the effect of masking any fault that affects a negative value. In addition, due to the bounded output of other functions such as sigmoid, they can mask faults that greatly modify the magnitude of an activation. However, such functions are rarely used in modern architectures.
- **Pooling:** MaxPooling has the ability to under-sample the input so it can mask faults present in the pooling window if they result in a lower value than the maximum value of the pooling window. Other types of pooling functions, like AveragePooling, mitigate the impact of the fault because the faulty data are averaged together with other potentially golden data.

Multiple works have dealt with detection of faults in neural networks. For instance, Goldstein et al. adapt AN code to detect errors in MAC units [32] and Hari et al. adapts Algorithm-Based Fault Tolerance (ABFT) to the convolution operation [33]. Other works use a threshold to mark, at the output of a layer, those activations with values too high as possible faults [31, 34].

In terms of mitigation, some works propose to mask faults with zero values, which significantly reduces the impact of faults [35, 36]. Other works limit the range of the activations either based on the value of their neighbors, the maximum value seen during training, or a value pre-established by the network designer [37–39]. Zahid et al. propose Fault-Aware Training (FAT), a technique in which faults are injected into the network during training, making the trained network more robust [40].

1.4.4. Aging Effects and Power-Gating Opportunity

Negative Bias Temperature Instability (NBTI) and Hot Carrier Injection (HCI) are two of the main detrimental effects that gradually increase the transistor's threshold voltage (V_{th}) over the lifetime of a circuit. It is important to emphasize that aging caused by these effects increases or exacerbates during the lifetime of a computing system and it is irreversible, that is, it is not possible to *reset* an aged transistor back to its initial performance or state.

Figure 1.13 depicts the implementation of a typical SRAM cell consisting of 6 transistors (6T cell). The four transistors labeled as either T_{Pi} or T_{Ni} form an inverter loop storing the logic value, whereas the remaining two T_{Wi} transistors act as pass transistors, controlled by the wordline (WL) signal, to access the cell contents through the bitline (BL) and its complementary ($\overline{BL}$).

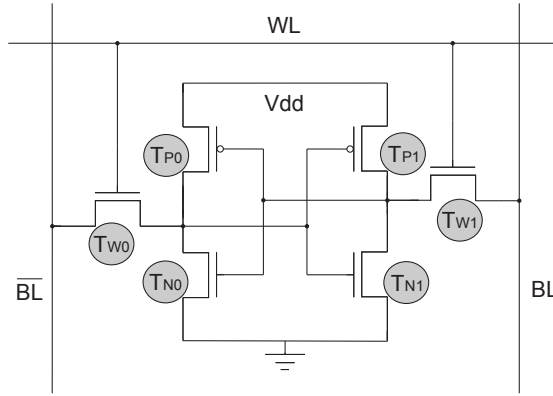


Figure 1.13: Implementation of a 6T SRAM cell, distinguishing between inverter loop PMOS (T_{Pi}) and NMOS (T_{Ni}) transistors, as well as NMOS (T_{Wi}) pass transistors.

The NBTI phenomenon mainly affects PMOS transistors when a logic ‘0’ is applied to their gates. In a 6T cell, this takes place in two ways. When the cell is under a ‘0’ duty cycle, that is, when the cell is stable and stores a logic ‘0’, the transistor T_{P0} is under stress and is affected by NBTI. On the contrary, under a ‘1’ duty cycle, the counterpart transistor T_{P1} suffers from NBTI. The degradation caused by each type of duty cycle is complementary, meaning that, for a given duty cycle, the PMOS transistor not under stress is partially under recovery from the NBTI effect. In other words, under a ‘0’ (‘1’) duty cycle, T_{P1} (T_{P0}) is under a partial recovery phase. Thus, if every bit cell of a memory array experiences a balanced duty cycle ratio 50% for each logic value), the wearout effect is evenly balanced between the two PMOS transistors and minimized compared to other cells with a higher duty cycle distribution. However, a balanced duty cycle ratio is unusual in a conventional design.

$$dV_{th_{NBTI}} = A_{NBTI} \times t_{ox} \times \sqrt{C_{ox} \times (V_{dd} - V_{t0})} \times \left(1 - \frac{V_{ds}}{\alpha_{NBTI} \times (V_{dd} - V_{t0})}\right) \times e^{\frac{V_{dd}}{t_{ox} \times E_{NBTI}} - \frac{E_a}{k \times T}} \times t_{stress}^{0.25} \times \left(1 - \sqrt{\frac{etha \times t_{rec}}{t_{stress} + t_{rec}}}\right) \quad (1.7)$$

Equation 1.7 shows the standard formula to compute the V_{th} degradation (dV_{th}) of a PMOS transistor due to the NBTI phenomenon [41]. Parameters t_{stress} and t_{rec} denote the amount of time (in seconds) that the transistor is under stress and recovery modes, respectively. In a 6T cell, T_{P0} and T_{P1} accumulate t_{stress} time when the cell stores a logic ‘0’ and ‘1’, respectively.

HCI mainly affects NMOS transistors when there is a logic value transition at their gates. In a 6T cell, T_{N0} and T_{N1} transistors are affected if the stored logic value flips as a consequence of a write operation. On the other hand, every cell access (read/write operation) induces HCI wearout to the T_{W0} and T_{W1} transistors. Note that the impact of HCI is proportional to the on/off switching frequency, and, contrary to NBTI, there is no recovery phase. Therefore, HCI is minimized when the number of accesses to a cell is reduced and writes do not change the stored value.

$$dV_{th_{HCI}} = A_{HCI} \times \alpha_{HCI} \times f \times e^{\frac{V_{dd}-V_{t0}}{t_{ox} \times E_{HCI}}} \times \sqrt{t} \quad (1.8)$$

The contribution of the HCI effect to the dV_{th} of an NMOS transistor is computed using the standard Equation 1.8 [42]. Parameter t refers to the amount of time (in seconds) that the gate transitions from ‘0’ to ‘1’ and vice versa. In a 6T cell, transistors T_{N0} and T_{N1} accumulate t time when the stored value flips, whereas transistors T_{W0} and T_{W1} do the same when the wordline is driven from high to low and vice versa.

The above equations for NBTI and HCI effects conform the aging model of this work. For those parameters not previously discussed, Table 1.2 summarizes them with a description and value for a 32 nm technology node.

Table 1.2: Main parameters of the NBTI and HCI aging model for a 32 nm technology node. Values of all these parameters can be found in [41–43].

Parameter	Description	Value
t_{ox}	Oxide thickness	1.65 nm
C_{ox}	Gate capacitance per unit area	4.6×10^{-20} F/nm ²
V_{dd}	Supply voltage	0.9 V
V_{t0}	Initial threshold voltage	0.2 V
V_{ds}	Drain-source voltage	0.7 V
E_{NBTI}	Technological constant	0.2 V/nm
Ea	Activation energy	0.13 eV
k	Boltzmann constant	8.6174×10^{-5} eV/K
T	Temperature	353.15 K
α_{HCI}	Activity factor	1
f	Clock frequency	1 GHz
E_{HCI}	Technological constant	0.8 V/nm

Power-gating is a well known technique to drastically reduce the power consumption of memory

structures [44]. This technique offers an opportunity to mitigate the mentioned aging effects.

An aging-aware power-gating configuration consists of an NMOS high- V_{th} sleep transistor connecting the 6T cell to ground. In this way, the cell ground terminal is connected to a virtual ground. When the sleep transistor drives current (active state), the cell operates as usual, yet with a ground voltage equal to the virtual ground. On the contrary, when the sleep transistor is off (*switch-off* state), the cell is disconnected from the ground and both T_{P0} and T_{P1} transistors remain partially under recovery from NBTI at the same time, since both cell nodes hold a logic ‘1’ [45]. Of course, as long as the cell remains off, not only high duty cycle distributions are reduced to combat NBTI, but also cell contents are neither accessed nor flipped, preventing HCI wearout. Notice too that, contrary to NMOS cell transistors, the NMOS sleep transistor is resilient to HCI since it is implemented using a high- V_{th} device [46, 47].

2 | Framework

This chapter presents the framework for the proposed aging-aware mechanism. The framework consists of a description of the used input datasets, the studied CNN benchmarks, an overview of the developed CNN inference accelerator model used as baseline, and finally, the main software tools and hardware environment employed in this project.

2.1. Datasets

Four different datasets were used in this thesis.

- **Imagenette [48]:** subset of ImageNet (one of the largest public image datasets available for research) with 10 easily classifiable classes.
- **Colorectal histology [49]:** a collection of textures in histological images of human colorectal cancer with a categorization of eight different types of tissue.
- **IMDB large movie reviews [50]:** set of reviews for binary sentiment classification.
- **Udacity’s self-driving car simulation dataset [51]:** autogenerated image data set using the autonomous driving simulation software provided by Udacity

Imagenette, Colorectal histology, and IMDB belong to classification tasks, whereas Udacity’s self-driving dataset refers to a regression task.

2.2. Benchmarks

This work defines a set of benchmarks consisting of a variety of widely used CNNs focusing on classification and regression tasks.

- **AlexNet (2012) [52]:** the first CNN to participate and win the ImageNet image recognition contest. AlexNet was one of the first CNNs to incorporate inter-layer normalization operations to accelerate the learning process.
- **ZFNet (2013) [53]:** modified version of AlexNet, winner of 2013 ImageNet competition.

- **VGG-16 (2014) [54]:** runner-up of 2014 ImageNet competition, it popularized the use of small 3×3 kernels that are still in use today.
- **PilotNet (2016) [55]:** DNN developed by Nvidia to test the control of the steer of an autonomous car on a roadway.
- **SqueezeNet (2016) [56]:** DNN designed for mobile and embedded applications, introduced the fire module, which consists of a combination of a 1×1 convolution to reduce the number of channels, called squeeze, and a set of convolutions of different dimensions (typically 1×1 and 3×3) concatenated to obtain multilevel characteristics called expand.
- **DenseNet (2017) [57]:** one of the first CNNs to introduce skip connections, which consist of connections in the activations of a layer from activations not necessarily of the immediately preceding layer.
- **MobileNet (2017) [58]:** DNN designed for mobile and embedded applications, introduced depth-wise convolution, a computationally lightweight way to perform convolutions.
- **SentimentalNet [59]:** generic 3-layer DNN for sentimental classification of text.

Table 2.1 shows specific details for each network, including the depth and type of layers, as well as the smallest, largest, and average layer size in KB assuming an activation size of 16 bits.

Table 2.1: Overview of the studied CNN benchmarks. Labels *Conv*, *FC*, and *DWConv* stand for convolution, fully-connected, and depth-wise convolution layers, respectively.

Benchmark	Task	Depth	Smallest layer size	Largest layer size	Average layer size
AlexNet	Image classification	$5 \times \text{Conv}$, $3 \times \text{FC}$, $3 \times \text{MaxPooling}$	18 KB (<i>MaxPooling3</i>)	0.58 MB (<i>Conv1</i>)	153 KB
ZFNet	Image classification	$5 \times \text{Conv}$, $3 \times \text{FC}$, $3 \times \text{MaxPooling}$	18 KB (<i>MaxPooling3</i>)	2.28 MB (<i>Conv1</i>)	324 KB
VGG16	Image classification	$13 \times \text{Conv}$, $3 \times \text{FC}$, $5 \times \text{MaxPooling}$	49 KB (<i>MaxPooling5</i>)	6.4 MB (<i>Conv1</i>)	1.32 MB
PilotNet	Turning angle regression	$5 \times \text{Conv}$, $5 \times \text{FC}$	2 KB (<i>Conv5</i>)	0.13 MB (<i>Conv1</i>)	26 KB
SqueezeNet	Image classification	$26 \times \text{Conv}$, $3 \times \text{MaxPooling}$, $1 \times \text{GblAvgPooling}$	3 KB (<i>Conv26</i>)	2.4 MB (<i>Conv1</i>)	431 KB
DenseNet	Image classification	$120 \times \text{Conv}$, $1 \times \text{MaxPooling}$, $3 \times \text{AvgPooling}$, $1 \times \text{FC}$, $1 \times \text{GblAvgPooling}$	12 KB (<i>Conv120</i>)	1.6 MB (<i>Conv1</i>)	254 KB
MobileNet	Image classification	$15 \times \text{Conv}$, $13 \times \text{DWConv}$, $1 \times \text{GblAvgPooling}$	49 KB (<i>DWConv12</i>)	1.55 MB (<i>DWConv1</i>)	340 KB
SentimentalNet	Sentimental classification	$1 \times \text{Conv}$, $1 \times \text{MaxPooling}$, $2 \times \text{FC}$	15 KB (<i>MaxPooling1</i>)	30 KB (<i>Conv1</i>)	15 KB

2.3. Modeled CNN Inference Accelerator

To study the effect of aging in current CNN accelerators as well as microarchitectural countermeasures to reduce its impact, this work models an abstract view of a generic accelerator in software. The proposed model is based on state-of-the-art models from both the academia and the industry to speed up the inference process in CNNs, such as DaDianNao [2], Google’s TPU [5], Eyeriss [6], or Bit-Tactical [9].

Figure 2.1a depicts the hardware organization of the baseline accelerator consisting of an 8×8 Processing Element (PE) array, a couple of 2 MB Input/Output (I/O) buffers for activation storage, a 2 MB weight buffer, dispatchers for every buffer, and a control unit. The computational and storage resources have been sized according to the domain of embedded systems [60]. Both activations and weights use a 16-bit fixed-point format.

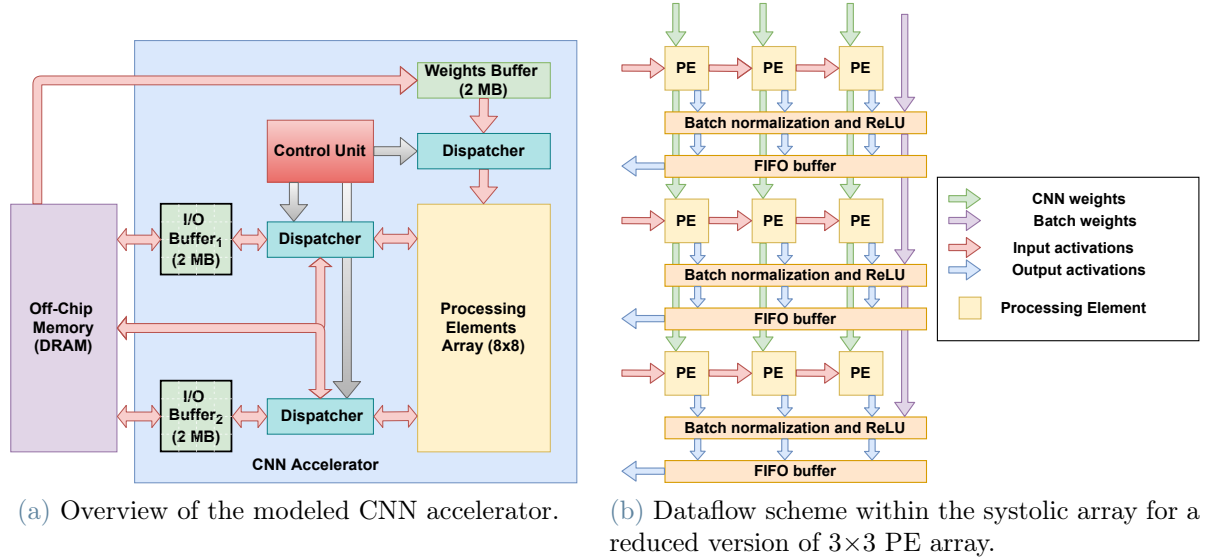


Figure 2.1: Overview and dataflow of the baseline CNN accelerator.

The PE array is a systolic array processor conformed by 64 PEs interconnected through a two-dimensional mesh. Each PE independently computes 16-bit fixed-point dot-products through partial sums with an input activation from one I/O buffer, acting as input buffer, and a weight from the weight buffer.

For illustration purposes, Figure 2.1b shows the dataflow in a 3×3 PE array, corresponding to the output stationary approach described in Appendix C. Input activations (red arrows) and weights (green arrows) are fed from left and top edges of the array, respectively. Then, they propagate to PEs in the same row and column through unidirectional links. To do so, PEs incorporate an internal register to store the incoming data in a given cycle, and forward the same data in the next cycle. Such a dataflow allows each PE to compute an output activation.

Moreover, each column of PEs processes the output activations from a specific weight filter, that is, those PEs generate adjacent output activations.

Notice too that each row includes a batch normalization and a Rectified Linear Unit (ReLU). These components are fed with the output activations from the corresponding row of PEs (blue arrows) and batch weights from the weights buffer (purple arrows). Then, the transformed output activations are temporarily stored in a 8×16 -bit FIFO buffer before writing them to the contrary I/O buffer acting as output buffer.

We assume that internal PE registers and FIFO buffers are implemented with resilient technologies like 8T cells to address aging at the cost of a slight area increase [61, 62]

The memory buffers provide intermediate storage for both activations and weights to reduce costly off-chip memory accesses. Since faulty weights have a much less impact on the network accuracy than faulty activations (see Figure 1), this thesis focuses on aging mitigation in the I/O buffers.

In the same way as EIE [4] and Alcolea et al. [11] inference accelerators, the I/O buffers swap their roles on every inference step. An inference step is defined as the processing of the output activations of a neural network layer given a set of input activations from the previous layer and weights. In this way, a given I/O buffer stores even layers and the counterpart buffer stores odd layers. On the contrary, the weight buffer caches weights to be issued in the proper order by the dispatcher to the PE array. Subsequent inference steps replace old weights with those required by the current step.

The relatively small I/O buffer size implies to spill the activations to off-chip memory when a layer does not fit in 2 MB. According to Table 2.1, this issue only affects ZFNet, VGG16, and SqueezeNet. More precisely, a single layer of ZFNet and SqueezeNet exceed the I/O buffer size, whereas 4 out of 21 layers from VGG16 exceed this size.

The I/O buffers are split into banks and sub-banks, similarly to as done in [3, 12]. In particular, each buffer consists of eight 256 KB banks. In turn, each bank is composed of eight 32 KB sub-banks. Figure 2.2 plots the bank organization of an I/O buffer. For illustration purposes, the buffer size has been limited to 512 bytes. The 16-bit activations are sub-bank (*sb*) interleaved and sequentially arranged bank after bank. This implies that the first 256 KB activations (64-byte activations in the simplified example) of a layer are always stored in *bank*₀. The figure also shows how a buffer address is partitioned in different fields according to the buffer organization. The three most significant bits identify the bank (*bank*₀ in the example), whereas the three least significant bits (apart from bit 0 referring to the activation byte) identify the sub-bank within the bank (*sb*₆). The remaining bits specify the row of the requested activation (*act*₁₄).



Figure 2.2: Bank organization of a 512-byte I/O buffer and an example of addressing. Label *sb* refers to sub-bank.

Finally, all the dispatchers are capable to transmit up to eight 16-bit activations or weights per cycle. Dispatchers are driven by the control unit, which exploits control information of the currently computed layer.

2.4. Software Tools and Hardware Environment

This work required to instrument and extend different software tools. First, Python has been chosen as the high-level programming language for this work, since it is highly compatible with machine learning libraries like *Pandas* and *Numpy*. In particular, the baseline CNN accelerator dataflow and on-chip memories, presented above, as well as the proposed microarchitectural solutions (see Chapter 3 and Chapter 4), have been modeled with TensorFlow simulation framework [63].

Tensorflow is an open-source framework developed by Google for modeling large data flow graphs and building ML programs which exploit the cuDNN neural network acceleration library developed by Nvidia. The use of Tensorflow for neural network development is quite straightforward. During the training phase, the programmer uses built-in operators to build a non-modifiable data flow graph based on the ML algorithm to be implemented. During inference, data is fed to the graph, which performs all pre-specified operations on it and delivers the model output.

To estimate timing, energy, power, and area numbers, the proposed design was synthesized with Synopsys Design Compiler, a software package that compiles synthesizable hardware description languages (HDL) into a netlist to target an ASIC standard cell library. Simulations were done using a technology library corresponding to a low-power 32 nm technology available to European universities via Mentor Graphics Modelsim. In addition, I/O buffers of the accelerator were modeled with the CACTI-P simulation framework [64]. CACTI establishes an analytic model

for SRAM-based on-chip memories and DRAM-based off-chip memories, providing, among other statistics, access time, energy consumption per cycle and memory event, and area of the modeled memory structures.

Finally, all the simulations were run on a Linux server using an NVIDIA RTX 3090 Ti GPU and a I7-6700K CPU. This environment was facilitated by the *Universidad de Zaragoza*, Spain, institution in which the MS candidate established a collaboration to perform this research. Table 2.2 details the specific versions of some of the software tools.

Software/Library	Version
Python	3.8.12
Pandas	1.3.2
Numpy	1.20.2
Tensorflow	2.5.0
cuDNN	8.2.1.32

Table 2.2: Main software environments and programming libraries used in this work.

3 | Characterization Study

This chapter describes a characterization study concerning a quantization method to efficiently represent network parameters in the CNN accelerator, as well as aging-related statistics consisting of duty cycles and bit flips in the activation memories. The results and conclusions extracted in this chapter drive the proposal of microarchitectural mechanisms introduced in the next chapter. For every study, implementation details in Tensorflow are detailed first. Then, results are presented and discussed.

All the experiments performed from this point on have been developed using a test dataset of 750,500,15000 and 3400 samples for the Colorectal, Imagenette, IMDB and Udacity self driving datasets respectively, with the exception of the duty cycle and bit flips simulation, which were obtained using 150 images, independent of the dataset, which is sufficient to stabilize all the aging patterns studied. Source code is available at <https://github.com/NicolasLanderos/Gated-CNN>.

3.1. Quantization Study

3.1.1. Implementation Details

Basically, a quantization approach consists of finding the minimum number of bits to represent a set of values, while maintaining the performance or accuracy of computations to be performed on a system. Quantization may also imply to change the data type of the values (e.g., from floating-point to fixed-point).

Tensorflow provides a default quantization method through Tensorflow lite. However, its options are limited and does not tolerate an arbitrary number of bits to represent data. Therefore, this thesis proposes to implement a manual quantization assuming a 16-bit fixed-point data type, not only for weights and network inputs, but also the output of every layer performing operations that alter the data. This is because Tensorflow continues to operate internally in standard IEEE-754 32-bit floating-point, and some operations may result in data with values out of range.

Since weights are static throughout the inference process, it is sufficient to quantize the pre-trained floating-point values beforehand. On the contrary, activations change for each input sample. To deal with this, a custom layer called *Quantization* is incorporated to the data flow

graph.

Quantization pseudo-code is displayed in Algorithm 3.1, it takes as input the activations A , the number of bits destined to encode the fractional part f_b , the number of bits for each activation a_b and an *Active* flag that determines if the quantization is performed or not, which is useful for measuring the performance of the model without quantization. This custom layer calculates the maximum M and minimum m value possible for the given bit configuration, then it converts the values of the input activations to their nearest downward fixed-point representation, finally the edge cases (all those values greater than M and less than m are saturated in M and m respectively using the max and min function).

Algorithm 3.1 Quantization custom layer

Require: A : Activations, *Active*: Active flag, f : number of fraction bits, a : number of activations bits

Ensure: A_Q : quantized activations

```

1: if Active == True then
2:   Calculate maximum quantized value  $M = \frac{((1 < (a-1)) - 1)}{2^f}$ 
3:   Calculate minimum quantized value  $m = -M$ 
4:    $A_Q \leftarrow \lfloor A * 2^f \rfloor / 2^f$ 
5:    $A_Q \leftarrow \min(A_Q, M)$ 
6:    $A_Q \leftarrow \max(A_Q, m)$ 
7:   return  $A_Q$ 
8: else
9:   return  $A$ 
10: end if

```

Figure 3.1 shows the data flow graph of an example of neural network before and after quantization. Note that layers such as ReLU and MaxPooling do not need the addition of a quantization layer because quantized data at their input are still quantized at their output.

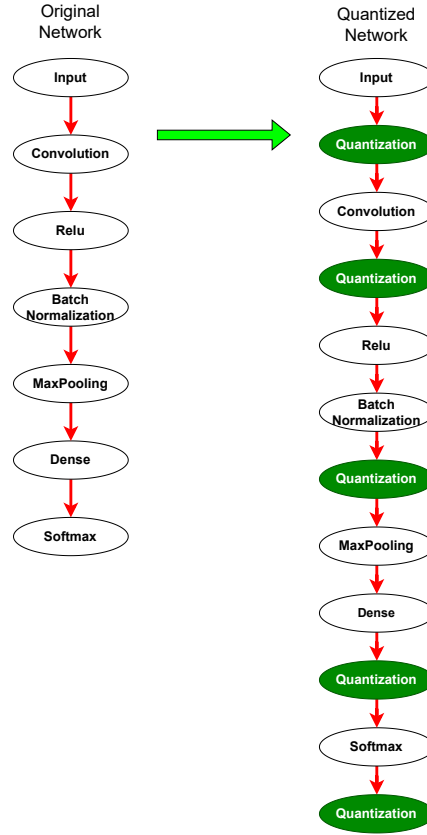


Figure 3.1: Quantization of a network with the introduction of quantization layers.

3.1.2. Results and Discussion

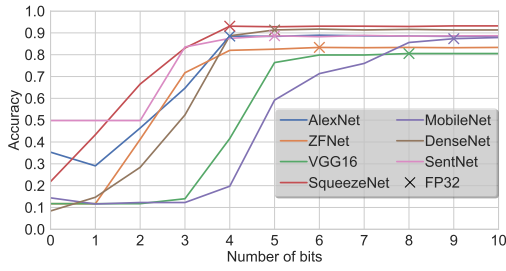
Once the quantization method has been defined, assuming a fixed-point representation, the performance of every CNN is analyzed according to the number of bits dedicated to the fractional part of the activations, designating for the other part a number of bits sufficiently large so that it does not impact on the performance (i.e., top-1 accuracy) of the network. Analogously, the integer part of the activations is also varied in number of bits. This analysis is performed for both activations and weights as shown in Algorithm 3.2.

Algorithm 3.2 Quantization test

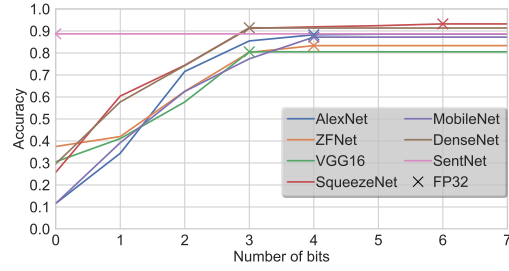
Require: N : Network list, D : Dataset

- 1: **for** $n \in N$ **do**
 - 2: **for** $b \leftarrow 0 \dots 15$ **do**
 - 3: Get accuracy of n in D with b fractional activation bits
 - 4: Get accuracy of n in D with b integer activation bits
 - 5: Get accuracy of n in D with b fractional weight bits
 - 6: Get accuracy of n in D with b fractional weight bits
 - 7: **end for**
 - 8: **end for**
-

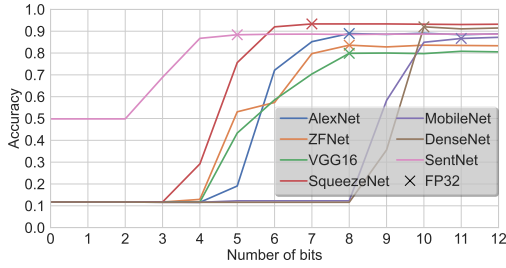
Figure 3.2 illustrates the results obtained for classification benchmarks using accuracy as performance metric in colorectal dataset for image classification and IMBD movies review for sentiment classification. Imaginette dataset was also analyzed with identical results. In terms of regression, Figure 3.3 shows the performance of PilotNet using loss function. Note that, in this case, only results for fractional bits are shown since zero integer bits are required to represent activations and weights without any performance loss.



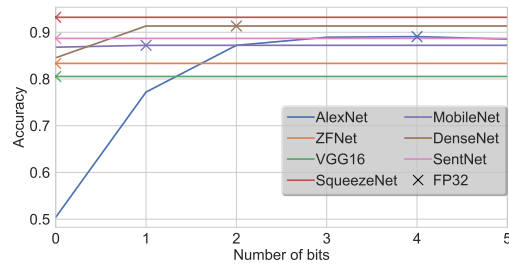
(a) Network accuracy vs number of bits for fractional part of activations



(b) Network accuracy vs number of bits for integer part of activations

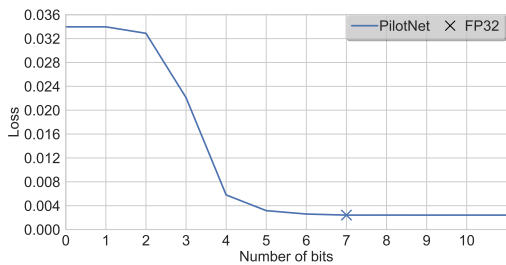


(c) Network accuracy vs number of bits for fractional part of weights

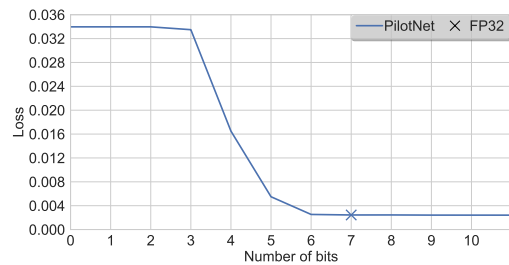


(d) Network accuracy vs number of bits for integer part of weights

Figure 3.2: Top-1 accuracy varying the number of bits for activations and weights. x marker corresponds to accuracy after training the network with 32-bit floating-point



(a) Network accuracy vs number of bits for fractional part of activations



(b) Network accuracy vs number of bits for fractional part of weights

Figure 3.3: Loss function varying the number of bits of activations in PilotNet. x marker corresponds loss after training the network with 32-bit floating-point

Sensitivity to truncation expresses differently in each benchmark. For instance, concerning activations, just 5 bits (0 integer + 5 fraction) are enough for SentimentalNet to reach the original accuracy, whereas up to 13 bits (4 integer + 9 fraction) are required by MobileNet. Based on these results, Table 3.1 summarizes the bit distributions for integer and fractional parts for each benchmark that ensure no performance degradation compared to 32-bit floating-point data type. With the aim to standardize the quantization, 16 bits are assumed for both activation and weight values, which is a common choice for most inference accelerators [1, 2, 4–6, 9].

Table 3.1: Quantization details for each benchmark.

Benchmark	Activation Representation	Weight Representation	Activation value max	Activation value min	Saturation ratio
AlexNet	4 int. bits & 4 frac. bits	4 int. bits & 8 frac. bits	44	-3	7×10^{-6}
ZFNet	4 int. bits & 6 frac. bits	0 int. bits & 8 frac. bits	14	-2	0
VGG16	3 int. bits & 8 frac. bits	0 int. bits & 8 frac. bits	4	0	0
PilotNet	0 int. bits & 7 frac. bits	0 int. bits & 7 frac. bits	1	-1	1×10^{-3}
SqueezeNet	6 int. bits & 4 frac. bits	0 int. bits & 7 frac. bits	109	-133	2×10^{-6}
DenseNet	3 int. bits & 5 frac. bits	2 int. bits & 10 frac. bits	13	-10	2×10^{-6}
MobileNet	4 int. bits & 9 frac. bits	1 int. bits & 11 frac. bits	6	0	0
SentimentalNet	0 int. bits & 5 frac. bits	0 int. bits & 5 frac. bits	2	0	2×10^{-9}

Note that the bit over-provisioning assuming 16-bit values is extended to the fractional part¹. The choice of adding fractional bits instead of integer bits is based on reducing the potential impact of faulty bits detailed in Section 1.4.3. More precisely, devoting more than necessary bits for the integer part would translate into integer most significant bits with a 100% ‘0’ duty cycle, thereby exacerbating NBTI in those bit cells. Notice too that, similarly to previous works [3, 8, 10], this implies the modeled baseline accelerator to dynamically adjust the number of integer and fraction bits required by each benchmark. Notice too that weights tend to require fewer bits for the integer part than activations, making the SDC effect less frequent and significant in comparison.

The table also incorporates the maximum and minimum values of activations in its non-quantized version. This explains why networks like SqueezeNet require a high number of bits for the integer

¹Leveraging bit over-provisioning as a backup for faulty bits deserves further exploration and is out of the scope of this work.

part and why SentimentalNet and PilotNet do not see their performance altered by not having integer part bits. The saturation ratio column shows the ratio between saturated activations (value out of the quantizable range) and total activations.

3.2. Duty Cycles, Bit Flips and Memory Accesses

3.2.1. Implementation Details

Since the study of this thesis focuses on aging in the activation memory buffers, during the inference process, only writes to the activation memories are modeled, performing an abstraction from the systolic array model. However, even when its not modeled, the systolic array info (number of columns/rows) is taken into account to determine the number of cycles between writes, order of writes, and number of parallel writes.

The write simulation is used to account for the logic values stored in the cells over time in order to derive flip and duty cycle values. To account for the number of reads and writes per address an analytical analysis is used, thanks to the predictable access patterns present in neural networks.

Algorithm 3.3 Inference test

Require: N_Q : Quantized network, B : An I/O buffer D : inference dataset

Ensure: B_f : Buffer flips, B_d : Buffer duty, c : cycles

- 1: Identify layers L from N_Q that write in buffer B
 - 2: **for** $d \in D$ **do**
 - 3: **for** $l \in L$ **do**
 - 4: Simulate the processing of l and update B_d, B_f, c
 - 5: **end for**
 - 6: **end for**
-

Algorithm 3.3 shows the logic used for the inference simulation. The following assumptions apply:

- The accelerator can deliberately choose in which activation memory buffer to store the first layer of the network. This simulation defines stablish that 50% of the samples start in each buffer. In this way, the bias in footprint memory caused by the fact that one buffer could only process even layers and the other odd ones is eliminated.
- Layers with size exceeding the buffer size are written to external off-chip memory, causing no changes to the stored data in the buffer.
- Inference will be performed with $batch_size = 1$, which is not uncommon for real-time applications. This implies that the accelerator cannot implement filter reuse.

- There is no memory structure between the buffers and the input to the systolic array to reduce the buffer accesses.
- Feature maps are stored in memory in NCHW format described in chapter 1. In this way, activations on the same channel and row are contiguous to each other, facilitating read operations.
- Pooling is performed off-chip.
- Each layer is always written starting from the first address of the buffer unless otherwise stated.

The rest of the implementation details refer to how read and write operations in the input/output activation memory buffers are modeled for each type of layer. Lastly, details about how aging-related stats are collected are also discussed.

Input Layers

This type of layer simply describes the transfer of data between external memory to the buffers. To model it in terms of memory, it is sufficient to write the activations in the buffer at a rate of B activations/cycle where B is the bandwidth between the accelerator and external memory in terms of activations.

Convolution Layers

This type of layer reads activations constantly from the input buffer and periodically writes activations to the output buffer. According to the output-stationary dataflow, the columns of the systolic array receive adjacent convolutional windows and the rows filters of adjacent output channels in order to process contiguous patches of the output periodically, as shown in Figure 3.4 and Figure 3.5 using a 3×3 systolic array for illustration purposes.

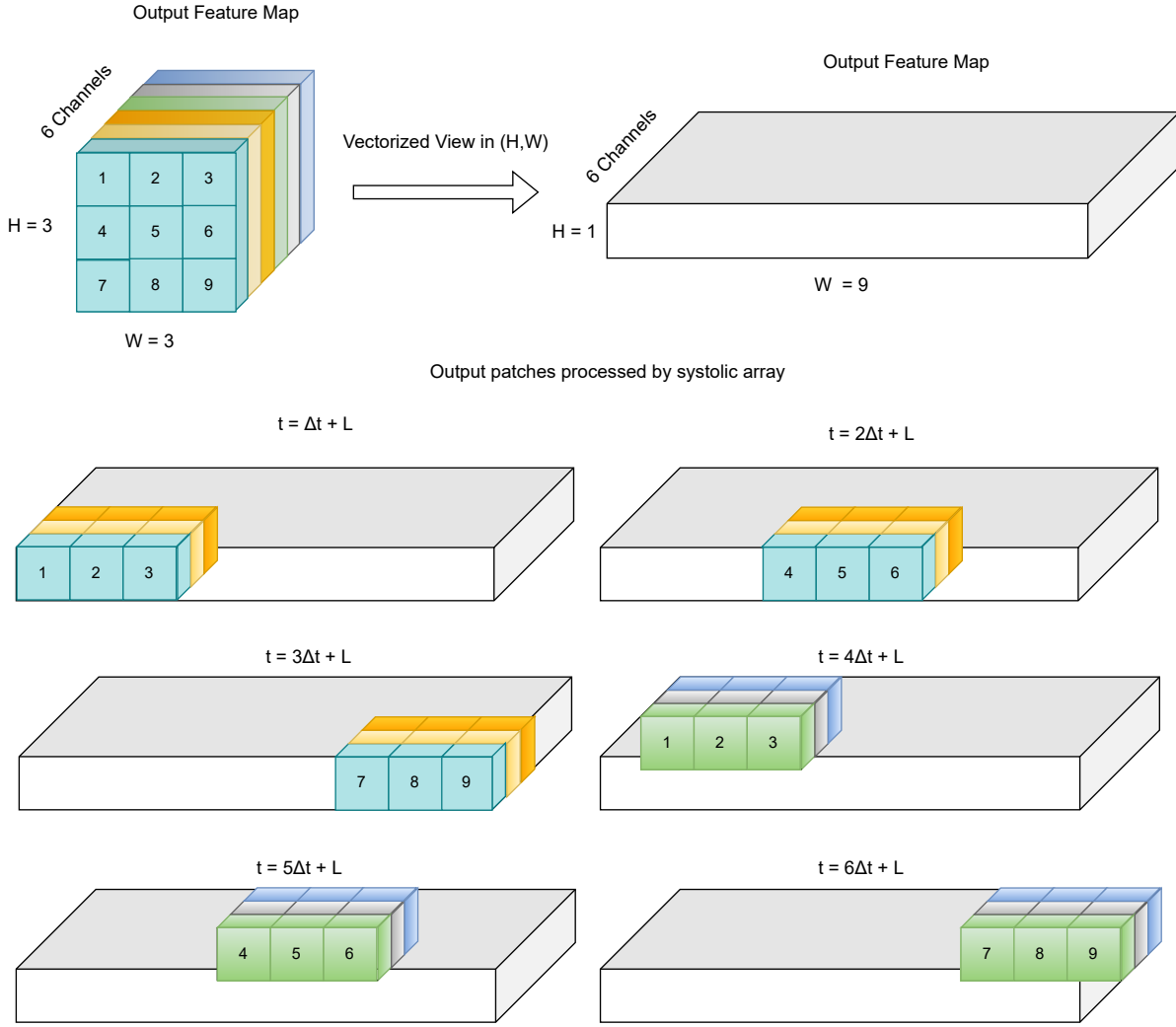


Figure 3.4: Order of activation processing in a convolutional layer for a 3×3 systolic array. Each patch is obtained after Δt cycles, L is the initial data propagation delay in the array plus the delay introduced by normalization and activation processing.

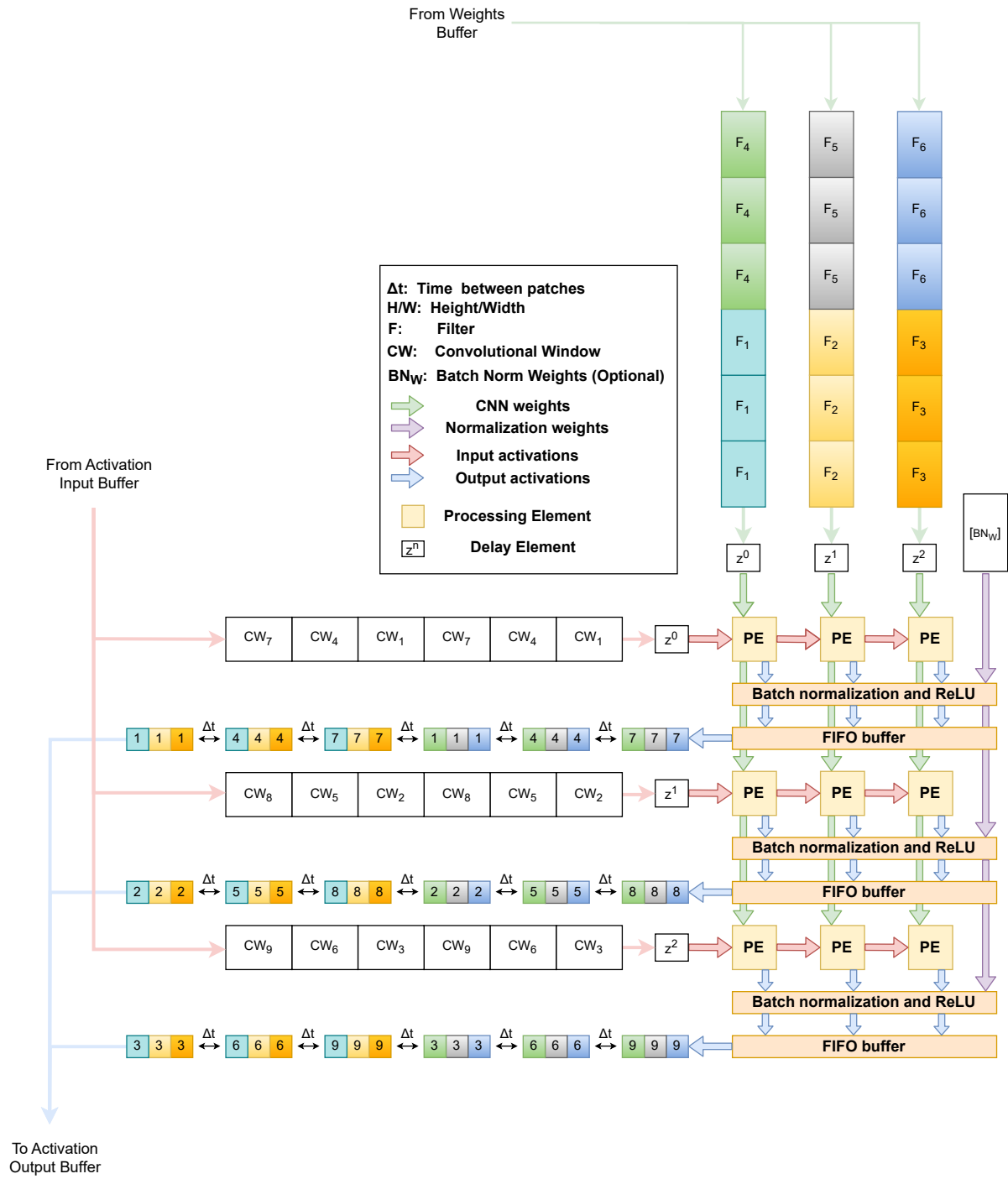


Figure 3.5: Inputs and outputs of the systolic array for the processing of the convolutional layer presented in Figure 3.4.

Using the parameters defined in Table 1.1 the following metrics are defined:

- **Propagation delay:** $L_p = R + C - 1$ are the cycles needed to propagate all the data in the array at the start of operation.
- **Data operation delay:** L_o are the cycles needed to transform the output in its normalization version (when batch normalization is used) and pass it through the activation function ReLU.
- **Cycles between patches:** $\Delta t = k_w k_h C_i$ correspond to the number of MACs required to process each activation output.
- **PE columns used:** $C_U = \min(C, F)$. Logically, if there are fewer filters than columns, not all columns can be used.
- **PE rows used:** $R_U = \min(R, HW)$. Logically, if there are fewer activations per feature map than rows, not all rows can be used.
- **Cycles required to process a layer:** $l_{cyc} = \Delta t \cdot \lceil \frac{HW}{R_U} \rceil \cdot \lceil \frac{F}{C_U} \rceil + L_p + L_o$

Dense Layers

Remember that a dense layer can be seen as a convolutional layer with $(H_o, W_o) = 1$, which implies that only one row of the systolic array can be used, since there will be no reuse of weights, there are two alternatives to increase performance: i) process multiple samples at once ($batch_size > 1$), which is out of the real-time inference context assumed above (besides, it greatly increases the memory requirements of the buffers), or ii) incorporate to the accelerator connections between the weight buffer and each PE as done in [65], at the cost of a larger area and bandwidth. Since the dimensions of the proposed systolic array are not too large, the latter alternative has been chosen. In this way, the convolutional window is broadcasted in the columns and each PE receives the weights for processing a different output activation.

The metrics for dense layers are the same than for convolutions with the exception of the following ones:

- PE columns used: $C_U = \min(C, M)$
- PE rows used: $R_U = \min(R, \lceil \frac{M}{C_U} \rceil)$
- Cycles required to process a layer: $C_l = \Delta t \cdot \lceil \frac{M}{R_U C_U} \rceil + L_p + L_o$

MaxPooling Layer

Since pooling is performed off-chip by assumption, this layer also consists of a data transfer between external memory and I/O buffer. Pooling is not computationally expensive, so the number of activations written to the buffer per cycle will only be limited by the bandwidth.

Normalization and Activation Operations

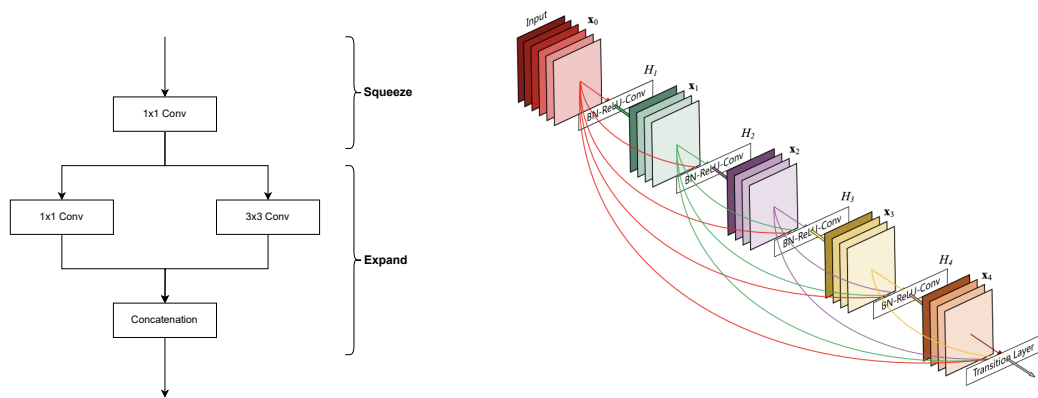
Normalization and activation operations are processed in conjunction with the previous convolutional/dense layer at the cost of a couple of extra cycles of latency. Batch normalization has only 4 weights per channel, so the effect on the bandwidth required between the I/O buffer and the systolic array is minimal.

Depth-wise Convolution

Contrary to dense layers, depth-wise (DW) layers have reuse of weights but not of activations. The same strategies discussed in dense layers can be adapted to the connection between I/O buffer and PE instead of weight buffer and PE. However, since DW layers require much less MACs than a convolutional layer, and including a connection between I/O buffer and PE is more expensive (since there are two I/O buffers compared to only one weight buffer), the accelerator is not modified, which implies that only one column is used for the processing of these layers. Regarding metrics, the only change with respect to a convolutional layer is $\Delta t = K_w K_h$.

Non-sequential Layer Modeling

As mentioned above, the accelerator is designed to process layers in *ping-pong* form between two buffers, which is well suited for sequential networks (those composed of an input, an output and a single data flow between them). However, many current networks have non-sequential elements. For instance, this is the case of SqueezeNet in the assumed benchmark suite for this work. SqueezeNet includes expand layers, which consist of the concatenation of two convolutional layers with the same input data and different filter size. On the other hand, in DenseNet, there are dense blocks in which the output of a convolutional layer is concatenated with the input of the convolution before it. Figure 3.6 shows graphically both expand and dense layers.



(a) Fire module of SqueezeNet con- (b) Example of dense block with 5 convolutions used
formed by a squeeze and expand part. in DenseNet [57].

Figure 3.6: Non-sequential layers used in SqueezeNet and DenseNet benchmarks.

In order to adapt these layers to the modeled accelerator, the following is done:

- **Expand:** both convolutions are processed as if they were a single layer: first, the 1×1 convolution and then the 3×3 convolution storing their activations in a contiguous memory area, which performs the concatenation implicitly, as depicted in Figure 3.7c.
- **Dense block:** the first convolution is performed in a normal way, but the inputs are not discarded after being used because when performing the second convolution, the outputs of these are written in a memory area contiguous to these inputs, which performs the concatenation in an implicit way, as seen in Figure 3.7b.

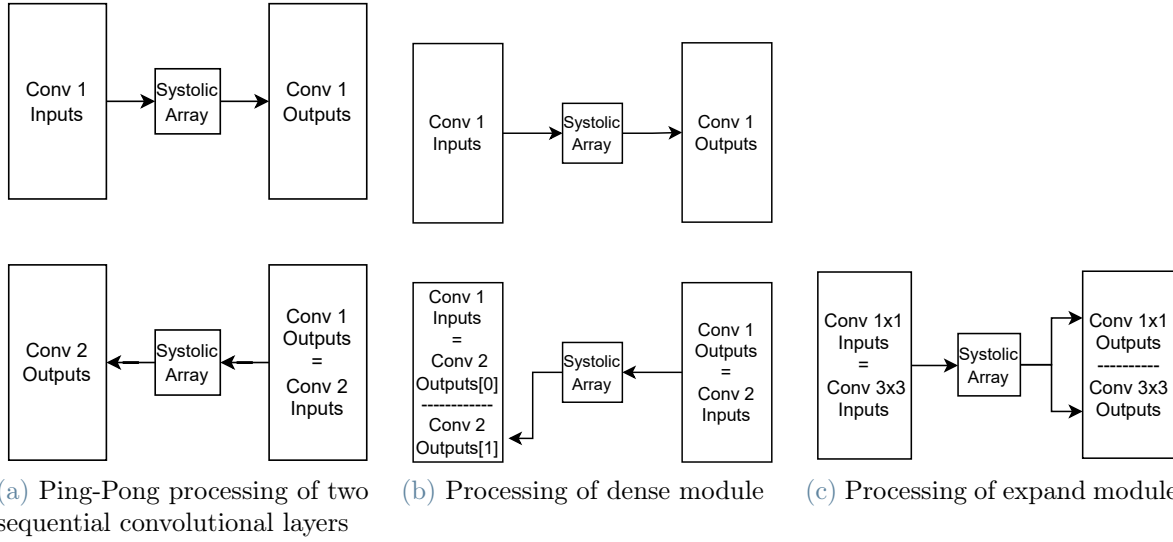


Figure 3.7: Processing of sequential and non-sequential convolutions by the modeled accelerator. Ping-pong sequential processing is shown for comparison purposes.

In summary, the simulation of writing to the buffer by a layer is summarized in three steps: obtaining the write metrics (activations per write, cycles between writes) and updating the buffer statistics (value, duty, flips) after each of these writes, as shown in the algorithm 3.4

Algorithm 3.4 Layer simulation

Require: l : Layer, O : Outputs activations B_f : Buffer flips B_d : Buffer duty, c : cycles B : buffer, A_i : accelerator info

- 1: Compute Δt : cycles between patches and p : patch size from A_i, l
 - 2: **for** each patch $o \in O$ of size p **do**
 - 3: Write o in B and Update B_d, B_f
 - 4: $c = c + \Delta t$
 - 5: **end for**
-

Aging-related Statistics

With the aim to collect statistics about duty cycles and bit flips, the activation memory buffers are modeled as arrays, where each element represents a memory cell with a logic value ‘0’ or ‘1’. In addition three more arrays of the same size are defined. One of them records the number of cycles with logic value ‘1’ per cell, another stores the number of cycles with logic value ‘0’ per cell, and the last one holds the number of flips in each cell. These arrays are updated each time a new value is written in the memory buffers.

Finally, recall that memory accesses also cause HCI aging in the cell pass transistors. Unlike duty cycles and bit flips, memory accesses can be calculated analytically since they do not depend on the specific read/written values. Writes are easy to account for, a layer with N output activations will always write to N contiguous addresses of the I/O buffer (one activation per address). Reads are taken into account according to the type of layer as follows:

- **Pooling:** since pooling is processed externally, it is assumed that the external processor has enough memory to read all data once and does not require repeated reads.
- **DW-Conv:** each convolution window is read one time.
- **Conv:** each convolution window is read $\lceil \frac{F}{C_U} \rceil$ times.
- **Dense:** each activation is read $\lceil \frac{N}{C_U R_U} \rceil$ times.

3.2.2. Results and Discussion

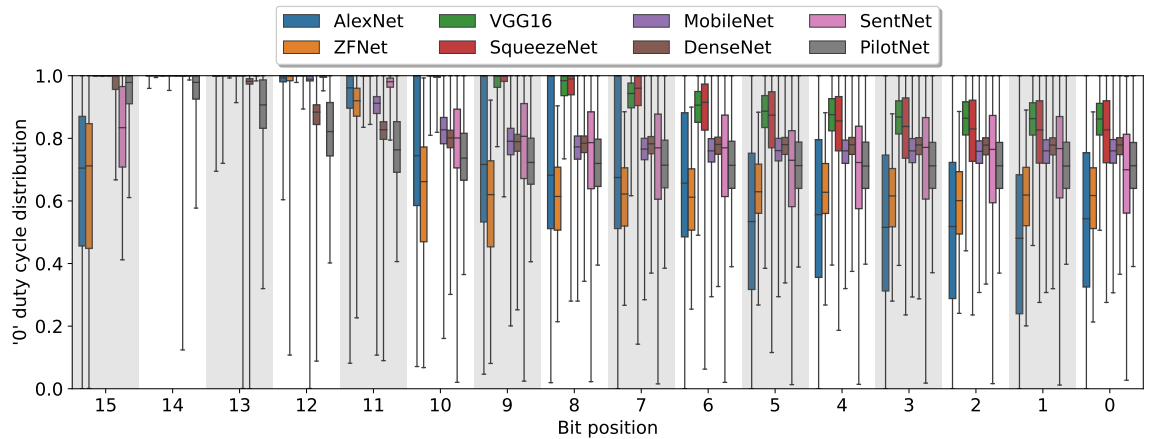


Figure 3.8: ‘0’ duty cycle distributions on each bit position of the 16-bit active activations.

Figure 3.8 depicts the ‘0’ duty cycle distributions (i.e., percentage of time storing a logic ‘0’ over the total execution time) experienced in every 16-bit activation word, represented in little-endian. The complementary distributions refer to the ‘1’ duty cycle. The distributions include the ‘0’ duty cycles for every active cell storing useful bits. That is, the figure does not include the

effect of duty cycles (either 100% ‘0’ or ‘1’ duty cycles) causing the maximum NBTI degradation in idle memory cells².

Apart from AlexNet and ZFNet, a biased ‘0’ duty cycle can be seen in the sign bit (15th bit) for every benchmark. In fact, for VGG16, SqueezeNet, and MobileNet, no negative activations are written in the I/O buffers, exacerbating NBTI in the sign bit. Recall that the integer part varies in number of bits depending on the benchmark, occupying until the 9th bit at most in SqueezeNet (see the rightmost column of Table 3.1). The ‘0’ duty cycle is also biased in most of these bits, and particularly the most significant ones, highlighting that the majority of activations are close to zero. On the other hand, a high ‘1’ duty cycle in a few bits can be seen in benchmarks like MobileNet and DenseNet.

In contrast to sign and integer bits, fractional bits show a broadened duty cycle distribution, pushing the median toward the ideal 50% duty cycle. However, in benchmarks like VGG16 and SqueezeNet, the median ‘0’ duty cycle exceeds 90% for all the bit positions. Moreover, long ‘0’ and ‘1’ duty cycles can be seen in many bit positions for applications such as AlexNet, SqueezeNet, and SentimentalNet.

Overall, NBTI wearout, from either ‘0’ or ‘1’ duty cycles, is appreciated not only in the sign and integer bit cells, but also in many cells storing fractional bits. That is, prior aging-aware techniques for CPU architectures working at a word granularity like [66–68] would be ineffective in CNN accelerators.

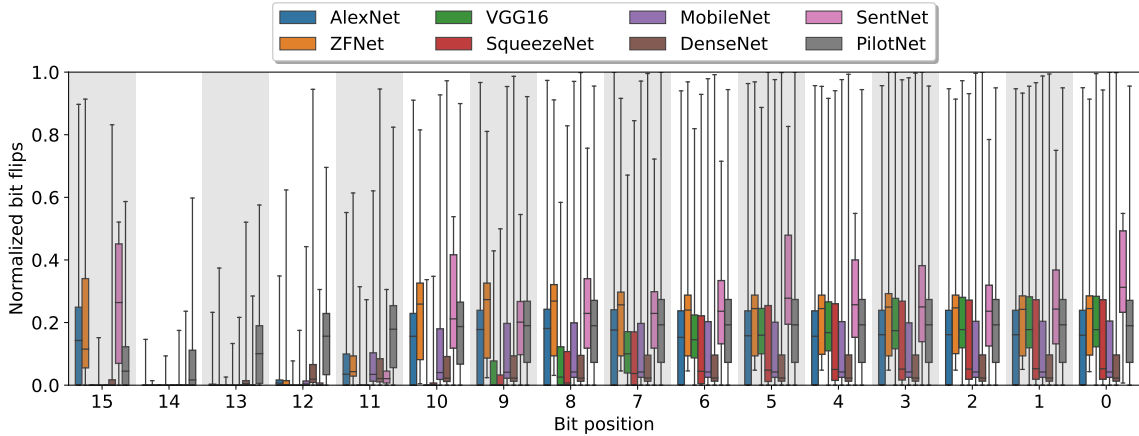


Figure 3.9: Normalized bit flips on each bit position of the 16-bit active activations with respect to the worst-case bit cell with the highest flip peak.

Figure 3.9 illustrates the normalized number of flips on every active bit cell with respect to the worst-case cell accumulating the highest flip peak for each benchmark. As expected from the

²For every box-and-whisker distribution plotted in this work, top and bottom box edges specify the 75th and 25th percentiles, lines within the boxes represent the median, and whiskers denote the maximum and minimum values.

previous analysis, the sign and integer bits, where the duty cycle is generally highly biased, show a lower number of flips compared to the fractional bits, where the duty cycle is generally less biased and a higher amount of flips are observed.

The flip pattern shows that few cells suffer from a high number of flips close to the highest flip peak, since all the boxes stay below 50% of the flip distribution. However, those cells are spread across all the fractional bits for all the studied benchmarks.

Note that unpredictable CNN workload input values may cause different duty cycle and flip patterns. However, since the proposed approach combats wearout in all the cell transistors, unpredictable workloads would benefit from the proposed technique in the same way as predictable workloads.

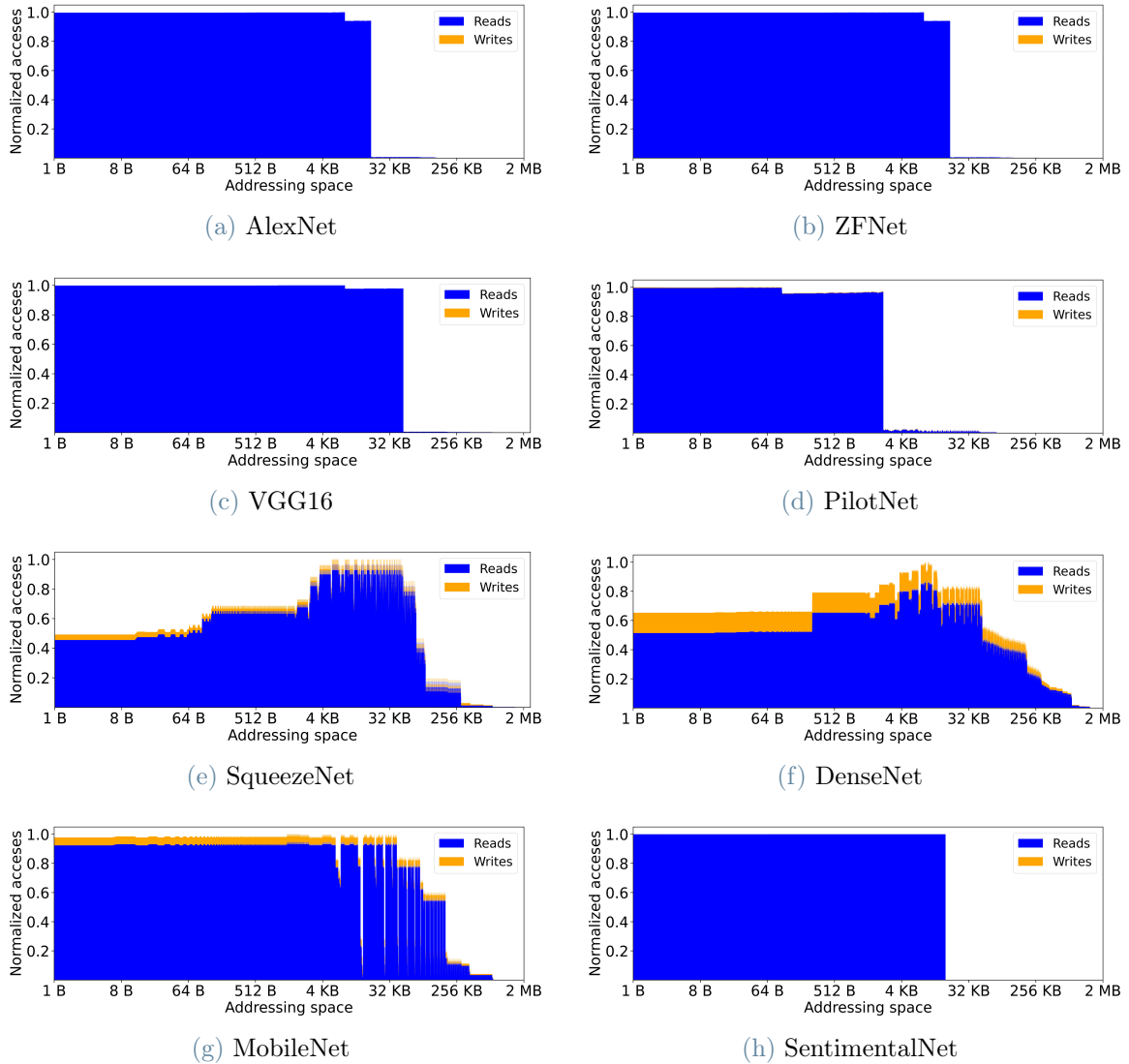


Figure 3.10: Normalized number of accesses (reads and writes) of the networks across the addressing space with respect to the memory address with the highest number of accesses.

Figure 3.10 plots, for each benchmark, the normalized number of read/write accesses to every byte of an I/O buffer with respect to the byte with the highest access peak. X-axis is shown in logarithmic scale for illustrative purposes.

As expected, most accesses are skewed toward low-order words, since every layer of a network is stored from address 0x0 onward. according to the defined implementation assumptions. In addition, considering all the studied benchmarks, the average number of read operations is $201\times$ higher than writes due to the activation reuse exhibited in convolutions. Therefore, T_{Wi} pass transistors are far more exposed to HCI degradation than T_{Ni} inverter loop transistors.

4 | Proposed Approach: Gated-CNN

This chapter introduces the proposed Gated-CNN design. First, it is shown a general overview of the approach, which consists of four main modules. Then, these modules are described in detail. After that, timing, power, energy, and area overheads are also discussed. Finally, software simulation details required to implement Gated-CNN in TensorFlow are briefly described.

4.1. Overview

Based on the previous characterization study, there are two different application patterns that speed up aging:

1. Memory cells dedicated to sign and significant integer bits suffer from a biased duty cycle compared to cells dedicated to fractional bits, exacerbating the NBTI degradation effect.
2. Cells located at the initial addresses of the I/O buffer are more frequently accessed compared to those located at higher addresses, causing a higher number of bit flips and reads/writes, exacerbating the HCI aging effect.

The proposed Gated-CNN design is aimed at minimizing both NBTI and HCI aging effects in every cell transistor of the I/O buffers of CNN accelerators. To do so, the key idea of Gated-CNN is to combine a bank address rotation scheme with a bank power-gating mechanism to spread out not only flip and access patterns (second pattern) but also switch-off cycles to balance duty cycle distributions across all the banks (first pattern). Gated-CNN leverages the following properties of CNN applications and specific-domain accelerators:

- The memory size of activation layers largely differ not only among different CNN applications but also among layers of a given CNN (see Table 2.1), leading to a dynamic under-utilization of the I/O buffers.
- CNN applications expose both temporal and spatial localities in a predictable manner. Once the activations of a layer have been read in order to compute the activations of the next layer, the former activations are not reused anymore.

- CNN accelerators include two I/O buffers that alternatively exchange input and output roles forcing a given buffer to store even or odd layers (see Section 2.3).

Consequently, the dynamic under-utilization is exploited to propose a cyclic storage of activation layers in successive banks, ensuring an homogeneous bank usage. Moreover, after using an activation layer to compute the next one, those banks storing the former stale layer are powered off. Notice too that exchanging buffer roles maximizes the period of time in which these banks are powered off.

The Gated-CNN component is coupled to one I/O buffer. That is, two Gated-CNN components are required for the baseline CNN accelerator. In addition, the proposed approach is designed for an I/O buffer with 8 banks. However, it could be redesigned to support a different number of banks with minor changes.

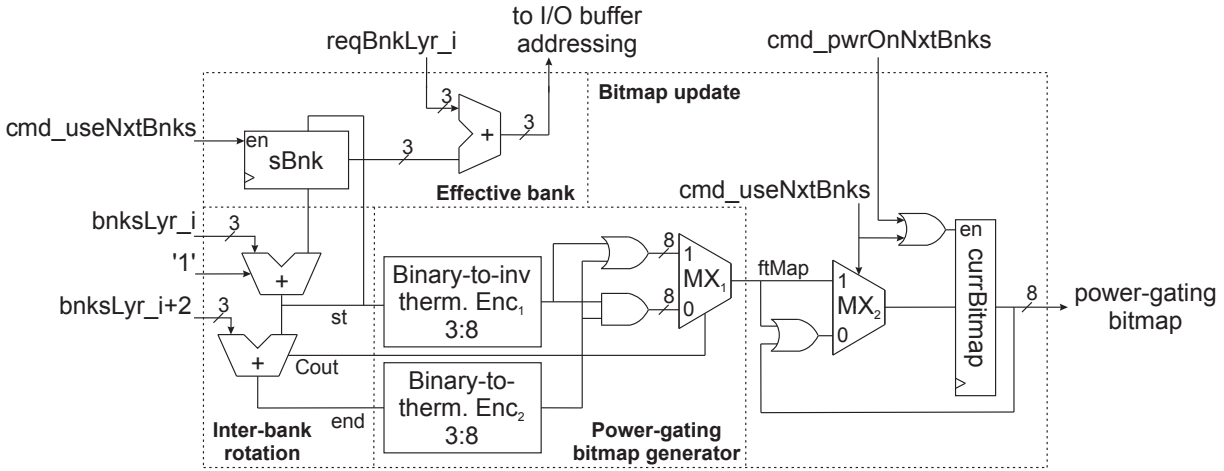


Figure 4.1: Proposed Gated-CNN design consisting of four main modules: effective bank, inter-bank rotation, power-gating bitmap generation, and bitmap update.

Figure 4.1 depicts the architecture of the four main modules of Gated-CNN. The *effective bank* module calculates the effective bank id where a requested activation is to be found in the I/O buffer. In addition, it forwards the starting bank id of the currently stored layer to the *inter-bank rotation* module. This module identifies the starting and ending banks required by the next inference step (next layer to be stored in the buffer), and forwards this information to the *power-gating bitmap generator* module. According to such banks, the power-gating bitmap generator computes a bitmap array that states which banks should be powered on/off in the next inference step. This bitmap is forwarded to the *bitmap update* module. This module updates the on/off state of the bank power-gating sleep transistors at runtime, masking the bank wake-up latency penalty.

4.2. Effective Bank Module

To request a given activation to the I/O buffer, the control logic of the accelerator refers to a logical address computed as an offset (in bytes) from the first activation of the currently stored layer in the buffer (i -th layer). Such a first activation is stored at address 0x0 (physical base $bank_0$) by default. The input $reqBnkLyr_i$ refers to the three most significant bits of the logical address, identifying the logical bank storing the requested activation (see Section 2.3).

The effective bank module translates from a logical to an effective physical bank. To do so, the 3-bit register labeled as $sBnk$ contains the physical base bank id (from $bank_0$ to $bank_7$) where the first activation of the i -th layer can be found in the buffer. This register is implemented with resilient 8T cells to address aging at the cost of a slight area increase [61, 62]. The adder outputs the current physical bank of the requested activation with the addition of the $sBnk$ and $reqBnkLyr_i$ bits.

Note that a layer might wrap around the I/O buffer (adder overflow or carry out bit set to ‘1’). That is, a layer occupying high-order and low-order banks, leaving free banks in the middle. Notice too that the adder output is concatenated with the 18 least significant bits of the logical address to properly index the 2 MB buffer with a 21-bit address (not shown in the figure).

Finally, the write enable (en) of the $sBnk$ register is driven by the $cmd_useNxtBnks$ signal. This signal is set when the physical base bank changes as a consequence of the next inference step storing activations of the new incoming layer. The subsequent section describes how the $sBnk$ contents are updated.

4.3. Inter-Bank Rotation Module

The inter-bank rotation module cyclically changes the value in $sBnk$ in a round-robin fashion throughout the banks. The next incoming layer, that is, the $(i+2)$ -th layer¹, starts occupying the successive available bank with respect to the last bank used by the previous i -th layer. To do so, the upper adder outputs that successive bank id (st bits) with the addition of $sBnk$, the $bnksLyr_i$ input, and the carry in bit set to logic ‘1’. The $bnksLyr_i$ input stands for the number of banks used by the i -th layer, which is computed as $i\text{-th layer size} \% \text{bank size}$. The $i\text{-th layer size}$ parameter is derived from the software profiling of the CNN. Note that $bnksLyr_i$ ranges from 0 (a single bank) to 7 (all the banks). The st bits will be stored in the $sBnk$ register when $cmd_useNxtBnks = '1'$ to properly compute an effective bank of the $(i+2)$ -th layer.

The bottom adder outputs the last bank that will occupy the $(i+2)$ -th layer (end bits). In contrast to the upper adder, this adder includes a carry out ($Cout$) bit, which flags a bank-wrapping situation for the $(i+2)$ -th layer. The st , end , and $Cout$ bits act as inputs for the

¹Remember that the I/O buffers exchange roles in every inference step, meaning that one buffer stores even layers and the other odd layers (see Section 2.3).

power-gating bitmap generation module described in the next section.

4.4. Power-Gating Bitmap Generation Module

The power-gating bitmap generator module dynamically distinguishes between active and idle banks of the incoming $(i+2)$ -th layer on every inference step. To do so, two binary-to-thermometer 3:8 encoders are required to generate a power-gating 8-bit map.

Cycle	t0	t1	t2 = t1 + 10	t3 = t2 + 1
Description	Access i-th layer	Access i-th layer and wake up banks for i+2-th layer	Access i-th layer and set bitmap for i+2-th layer	Access i+2-th layer
Inputs, registers, and intermediate signals	$sBnk = 0$ $bnksLyr_i = 2$ $st = 3$ $outEnc_1 = '11111000'$ $bnksLyr_i+2 = 1$ $end = 4$ $outEnc_2 = '00011111'$ $Cout = '0'$ $ftMap = '00011000'$ $(outEnc_1 \& outEnc_2)$ $currBitmap = '00000111'$	$currBitmap = '00011111'$ $(ftMap \mid currBitmap)$	$sBnk = 3$ $currBitmap = '00011000'$ $(ftMap)$	$sBnk = 3$ $bnksLyr_i = 1$ $st = 5$ $outEnc_1 = '11100000'$ $bnksLyr_i+2 = 3$ $end = 0$ $outEnc_2 = '00000001'$ $Cout = '1'$ $ftMap = '11100001'$ $(outEnc_1 \mid outEnc_2)$ $currBitmap = '00011000'$
Commands	$cmd_pwrOnNxtBnks = '0'$ $cmd_useNxtBnks = '0'$	$cmd_pwrOnNxtBnks = '1'$ $cmd_useNxtBnks = '0'$	$cmd_pwrOnNxtBnks = '0'$ $cmd_useNxtBnks = '1'$	$cmd_pwrOnNxtBnks = '0'$ $cmd_useNxtBnks = '0'$

Figure 4.2: Driving example of the CNN-Gated component, including the state of the inputs, registers, and intermediate signals at specific cycles of the inference process. Signals not appearing in a given cycle means that the state is preserved with respect to the previous cycle on the left.

To help understand how this module works, Figure 4.2 shows a driving example including the state of the inputs, registers, and intermediate signals at specific cycles. At cycle $t0$, the i -th layer occupies $bank_0$, $bank_1$, and $bank_2$ ($sBnk = 0$ and $bnksLyr_i = 2$), whereas the $(i+2)$ -th layer will occupy $bank_3$ and $bank_4$ ($bnksLyr_i+2 = 1$). The upper inverse encoder, referred to as Enc_1 , is fed with the st bits and outputs an 8-bit map in which bits are set to ‘0’ except those from the st -th bit up to the most significant bit. In the example, $st = 3$ generates the output $outEnc_1 = '11111000'$.

On the other hand, the bottom encoder, namely Enc_2 , is fed with the end bits. In contrast to Enc_1 , all the output bits of this encoder are set to ‘0’ except those from the end -th bit down to the least significant bit. In the example, $end = 4$ generates the output $outEnc_2 = '00011111'$.

In order to generate a future power-gating bitmap for the $(i+2)$ -th layer ($ftMap$), both encoding outputs are merged as follows. When $Cout = '0'$, i.e., when $st \leq end$, the encoder outputs are bit-wise ANDed. In the example, $outMX_1 = '00011000'$, meaning that $bank_3$ and $bank_4$ will remain powered on and the rest powered off in the next inference step at cycle $t3$. Otherwise,

Table 4.1: Timing, energy, power, and area values for the proposed Gated-CNN approach and an I/O buffer using a 32 nm technology node.

	Gated-CNN	I/O buffer	Overhead (%)
Access time (ns)	0.19	1.88	10.11
Dyn. energy (pJ)	0.29	162.19	0.18
Leak. power (mW)	1.39	178.14	0.78
Area (mm^2)	0.003	3.378	0.09

$st > end$, the required banks wrap around the I/O buffer. In such a case, the encoder outputs are bit-wise ORed. This is the case of the example at cycle $t3$, where $Cout = '1'$ ($st = 5 > end = 0$). Finally, note that $Cout$ determines the appropriate bit-wise operation by driving the MX_1 multiplexer.

4.5. Bitmap Update Module

The purpose of the bitmap update module is to establish the appropriate bitmap to the power-gating mechanism cycle by cycle. This module takes as inputs $ftMap$ from the previous module, and two command signals, $cmd_pwrOnNxtBnks$ and $cmd_useNxtBnks$, specifying, respectively, when the banks required by the $(i+2)$ -th layer should be powered on and when the next inference step starts.

The 8-bit *currBitmap* register, implemented with 8T cells, stores the current power-gating bitmap applied to the banks, that is, it refers to the i -th layer. Using the example in Figure 4.2, this register stores the bitmap ‘00000111’ at cycle $t0$. At cycle $t1$, those banks required by the $(i+2)$ -th layer start to be woken up in advance. To do so, $cmd_pwrOnNxtBnks$ and $cmd_useNxtBnks$ are set to ‘1’ and ‘0’, respectively. This allows $ftMap$ and *currBitmap* to be bitwise ORed, resulting in a new *currBitmap* = ‘00011111’ (MX_1 selection bit = ‘0’ and $en = '1'$), specifying not only the required banks of the i -th layer but also those of the $(i+2)$ -th layer set to ON. Like previous works [69, 70], it is assumed the wake up latency of a bank to be 10 cycles. That is, the requested banks are powered on 10 cycles ahead of starting the next inference step, which ensures the wake-up latency to be out of the critical path.

At cycle $t2$, $cmd_pwrOnNxtBnks$ and $cmd_useNxtBnks$ are set to ‘0’ and ‘1’, respectively, establishing the proper bitmap for the next inference step (*currBitmap* = $ftMap$ = ‘00011000’). That is, at this cycle, those banks no longer required are powered off. Notice too that, at the same time, the *sBnk* contents are updated accordingly. Finally, at cycle $t3$, a new $ftMap$ is obtained according to the size of the subsequent $(i+2)$ -th layer.

4.6. Timing, Energy, Power, and Area Estimations

To measure timing, energy, power, and area numbers, the proposed Gated-CNN design has been synthesized with Synopsys Design Compiler and simulated with Mentor Graphics Modelsim. The technology library corresponds to a low-power 32 nm technology available to European universities. Table 4.1 summarizes the results. For comparison purposes, results include the estimations for an I/O buffer and the overheads of Gated-CNN with respect to the buffer. The I/O buffer has been modeled with the CACTI-P simulation framework [64].

The access time of Gated-CNN refers to its longest path delay from *sBnk* to *currBitmap* registers, which is out of the critical path. The dynamic energy of Gated-CNN refers to obtaining a new power-gating bitmap given a set of inputs, whereas these expenses for the I/O buffer are related to accessing the memory. Compared to the numbers of the I/O buffer, the overheads of Gated-CNN are minimal. In addition, both energy and power overheads are largely compensated with the benefits brought by power gating idle buffer banks.

Finally, CACTI-P has been also used to model the bank power-gating in the I/O buffers, including all the sleep transistors and interconnects. The area overhead of the power-gating technique is by 1.39% with respect to the I/O buffer, which is a similar overhead as those reported in other power-gating designs from the industry [71, 72].

4.7. Implementation Details

Once the baseline CNN accelerator and cycle-accurate dataflow have been modeled in TensorFlow (see Chapter 3, extending the architecture to support the proposed Gated-CNN design is relatively straightforward. Basically, it is sufficient to add the state OFF to the possible value of every cell. At the beginning of the inference of each layer, it is determined which cells are turned ON/OFF. For the address rotation scheme, a variable tracks the write address for the next layer based on the current base write address and the layer size.

5 | Experimental Evaluation

This chapter quantifies the reduction of the transistor aging effects brought by the proposed Gated-CNN technique. First, an analysis of duty cycles, bit flips, and access patterns in those memory cells with the highest aging degradation is presented. Then, a sensitivity study of the proposed approach to the I/O buffer size is also discussed. Finally, the aging reduction is quantified with the aging model described in Chapter 1.

5.1. Duty Cycle Analysis

Figure 5.1a plots, for all the studied benchmarks, the ‘0’ duty cycle distribution for the worst-case cell, that is, the longest ‘0’ duty cycle. Gated-CNN incorporates a *switch-off* state to reflect the amount of time that the worst-case cell is powered off. The cross symbol in every bar refers to the average ‘0’ duty cycle for all the cells. BL and GC stand for a baseline approach without any aging-mitigation scheme and Gated-CNN, respectively. Like the previous characterization study in Section 3.2, we conservatively assume that the baseline results only refer to active cells storing useful contents. On the contrary, Gated-CNN uses all the cells, and consequently all of them are included in the results.

As expected, the maximum ‘0’ duty cycle distribution for the baseline approach is 100% in every benchmark. In addition, the average ‘0’ duty cycle is also highly biased, as mentioned above. On the other hand, by powering off idle banks and uniformly distributing switch-off cycles across them, Gated-CNN largely reduces the longest ‘0’ duty cycle beyond 50% in all the benchmarks apart from AlexNet.

The differences among benchmarks largely depend on the power-off opportunities provided by the CNN layer dimensions. In this sense, the highest ‘0’ duty cycle reductions are seen in Senti-mentalNet and PilotNet due to these benchmarks have the lowest average layer size requirements (15 KB and 26 KB, respectively, see Table 2.1). Another factor that contributes to obtain large reductions is the fact that some layers do not fit in the I/O buffer and they are spilled to off-chip memory. Meanwhile, all the I/O buffer banks are powered off except for the banks containing the activations sent off-chip and those waiting for the results to be written. Such a number of switch-off cycles mainly depends on the required processing time of the layer. The high ‘0’ duty cycle reductions observed in ZFNet and SqueezeNet are mainly due to both a relatively small

average layer size and a larger than 2 MB layer.

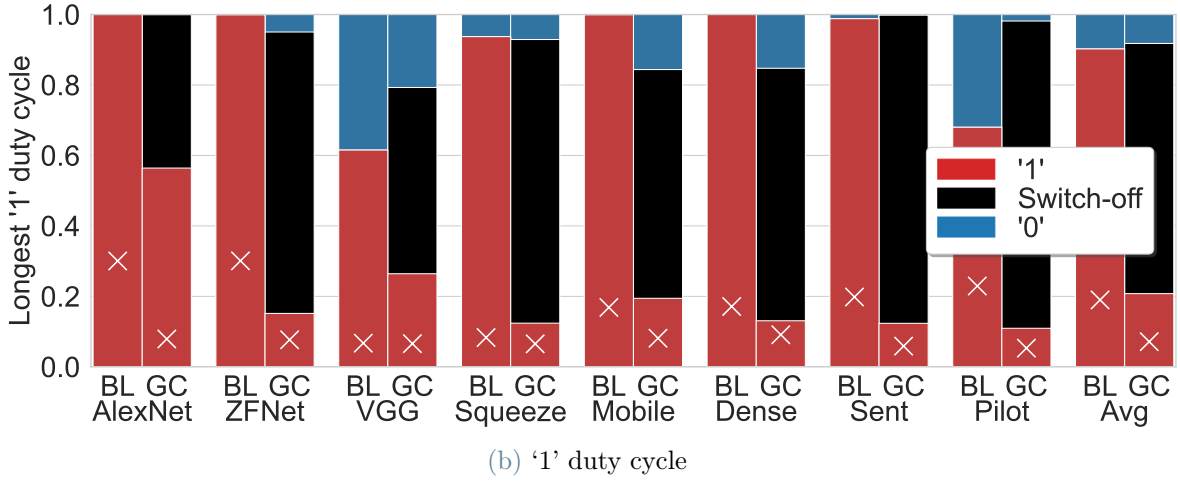
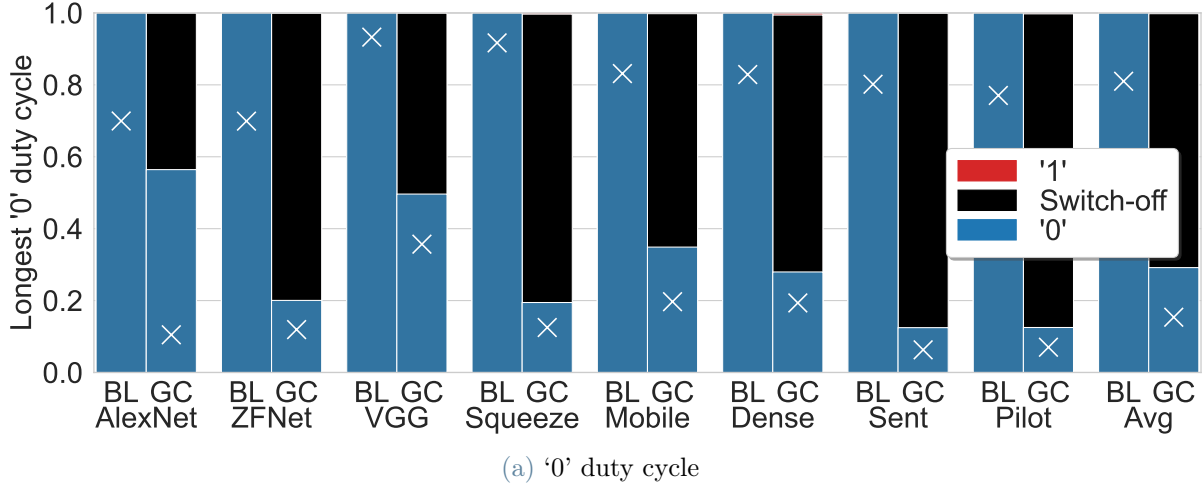


Figure 5.1: Longest duty cycle distributions across all the studied benchmarks. The cross symbol represents the average duty cycles for all the cells. BL and GC refer to the baseline and Gated-CNN designs.

On the other hand, AlexNet does not reduce the '0' duty cycle as much as expected from the workload characteristics. This is due to the combination of the sizes of both banks and layers leading to some banks being more exercised than others (less switch-off cycles than expected), since these banks recurrently store compute-intensive layers. Nevertheless, the obtained reduction for the worst-case cell is as high as 44% in this CNN, whereas the average duty cycle reduction reaches a 90%.

Overall, Gated-CNN reduces the average highest '0' duty cycle by 71%. Taking into account all the cells, the average '0' duty cycle reduction is by 85%.

The counterpart, longest '1' duty cycle distribution, should be also considered for a complete analysis of the NBTI effect. Figure 5.1b shows the results. Unlike the worst-case cells of the

baseline with a 100% ‘0’ duty cycle (Figure 5.1a), the longest ‘1’ duty cycle distribution for the baseline does not reach 100% in all the benchmarks. In addition, the average ‘1’ duty cycle greatly reduces in the baseline scheme. These observations confirm the presence of more zeros than ones in the activations. For the Gated-CNN design, the percentage of the switch-off state in the distribution is the same as in the previous analysis. Like the baseline, the maximum ‘1’ duty cycle is further reduced with the presence of zeros. This can be clearly seen in VGG16, MobileNet, and DenseNet.

In summary, Gated-CNN reduces the average longest ‘1’ duty cycle by 79%. Taking into account all the cells, the average ‘1’ duty cycle reduction is by 93%. Comparing the results of both duty cycle distributions, transistors T_{P0} are more exposed to the NBTI effect than transistors T_{P1} (see Section 1.4.4).

5.2. Bit Flip and Access Analysis

Figure 5.2 shows the normalized highest number of bit flips and accesses of the worst-case cells in the Gated-CNN design with respect to those cells of the baseline. These results only refer to active cells storing useful bits in both approaches. Otherwise, the normalized average numbers (cross symbols) would be skewed toward zero.

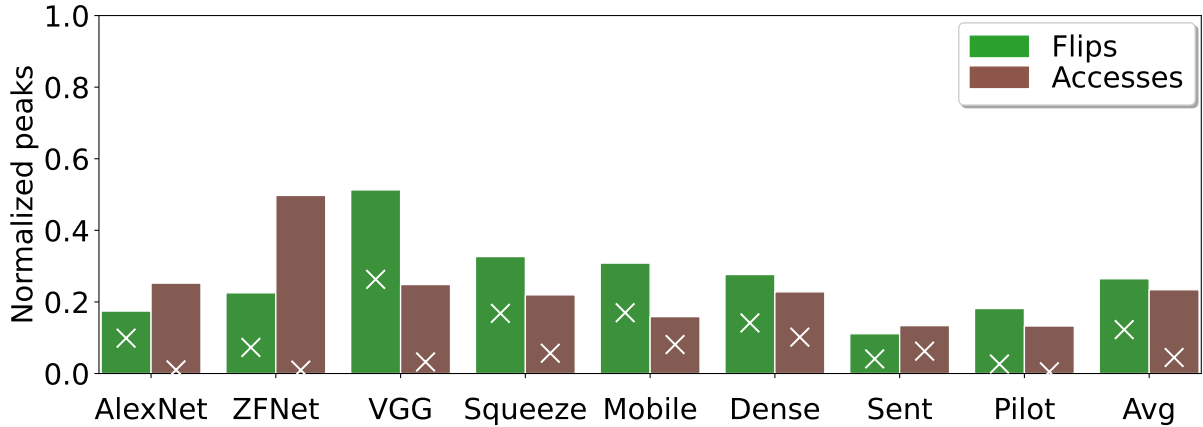


Figure 5.2: Normalized highest bit flip and access peaks of Gated-CNN with respect to the baseline design. The cross symbol refers to the normalized average bit flips and accesses for all the cells.

By rotating data across banks, Gated-CNN substantially reduces both the maximum flip and access peaks compared to the baseline. The largest peak reduction is seen in SentimentalNet. This is mostly due to every layer fits in a single bank, preventing flips in the remaining banks. The other benchmarks obtain higher peaks mostly due to larger layers occupying multiple banks. However, both the highest flip and access peaks are reduced at least as much as 49% (VGG16 and ZFNet). In addition, the average peaks do not exceed a 27% (VGG16) in any benchmark.

Overall, the highest flip and access peaks are reduced on average by 74%. Taking into account all the cells, the average reductions are as much as 88 and 96% for flips and accesses, respectively.

5.3. Sensitivity Study to the I/O Buffer Size

The power-off and rotation capabilities of Gated-CNN not only depend on the workload characteristics but also on the size of the I/O buffer. In this work, we have assumed a generic accelerator with a total on-chip activation storage of 4 MB ($2 \text{ MB} \times 2 \text{ I/O buffers}$) capable to speed up the inference of diverse CNNs with disparate activation storage requirements. Such an overall activation storage coincides with the assumed activation storage for the DNN-Life’s baseline accelerator [16], and it is much less than the storage capacity of other inference accelerators like the Google’s TPU with 24 MB for activations [5].

This section evaluates the sensitivity of Gated-CNN to the I/O buffer size in terms of both the power-off and rotation capabilities. More precisely, the size of the I/O buffer is adjusted to the largest layer size of each CNN application to quantify the Gated-CNN benefits only from the workload characteristics (see Table 2.1). Like the proposed original approach, a number of eight same-sized banks is assumed regardless of the buffer size.

Plots in Figure 5.3 illustrate the previous duty cycle distributions (Fix) shown in Figure 5.1 and the new distributions adjusting the I/O buffers to the largest layer size (Adj). Results only refer to the Gated-CNN design. For an adjusted size of the I/O buffer, the opportunity to switch-off banks significantly decreases in benchmarks with less than a 2 MB buffer size like SentimentalNet and PilotNet, but the longest duty cycle reductions are still at least as high as 44%. On the contrary, other benchmarks also with a smaller buffer size, like AlexNet and MobileNet, further reduce the longest duty cycle with respect to the previous results. This is mainly due to layers occupying a greater number of banks, which in turn changes the round-robin bank assignment to every layer, preventing compute-intensive layers to always remain in the same banks.

On the other hand, CNNs with more than a 2 MB buffer size increase the longest duty cycle due to huge layers are not spilled to off-chip memory anymore, reducing the number of switch-off cycles. This is the case of ZFNet and SqueezeNet. For VGG16, the reason for a duty cycle reduction is the same as explained above for AlexNet and MobileNet.

Notice too that, similarly to the longest duty cycle, the limitations of Adj in number of switch-off cycles due to either a buffer size reduction or spilling prevention translate into a higher average duty cycle over Fix in most benchmarks.

To sum up, despite adjusting the I/O buffer size to the largest layer storage requirements, the average longest ‘0’ and ‘1’ duty cycle reductions are as much as 63 and 76%, respectively.

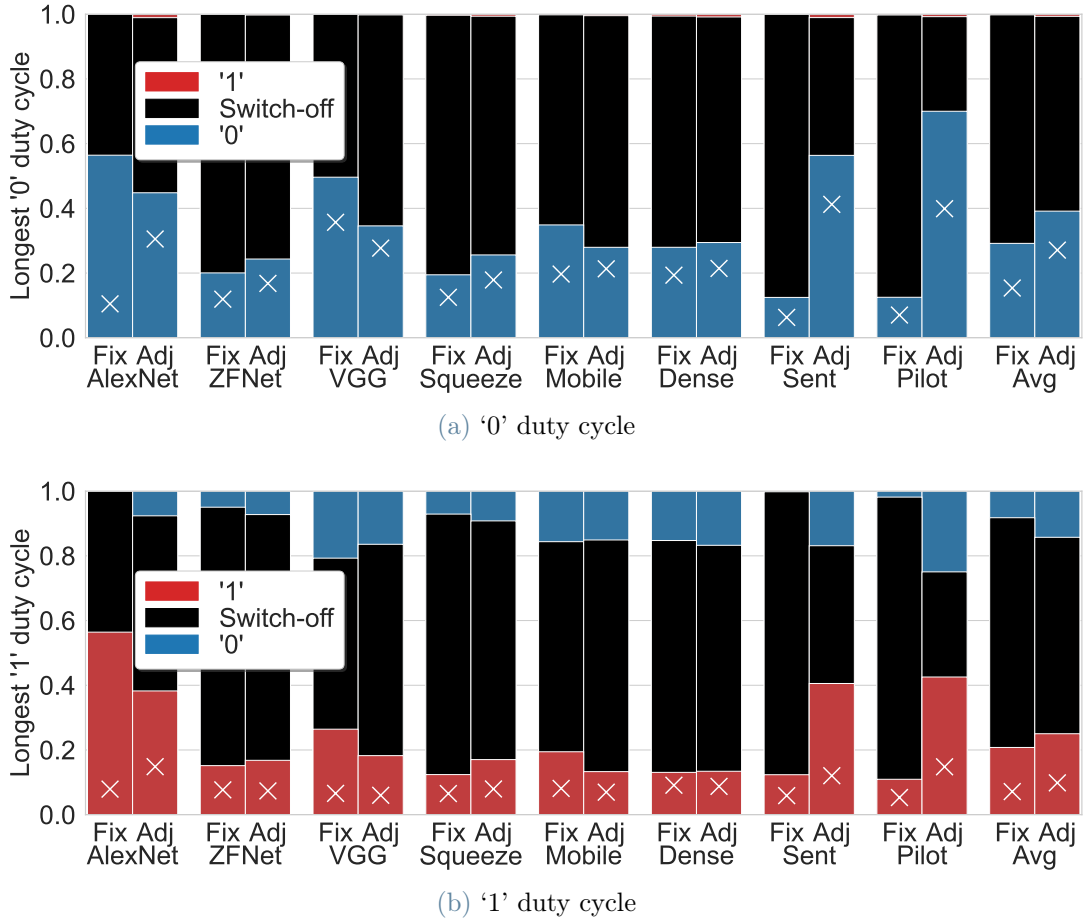


Figure 5.3: Longest duty cycle distributions for Gated-CNN. Results differentiate between a fixed 2 MB I/O buffer size (Fix) and a buffer size adjusted to the largest layer (Adj).

Figure 5.4 depicts the normalized highest flip and access peaks for fixed (Figure 5.2) and adjusted buffer sizes. Small CNNs like SentimentalNet and PilotNet substantially increase the normalized peaks, especially flip peaks. This is mainly due to, with an adjusted buffer, layers of small CNNs occupy a larger number of banks, increasing the likelihood of a higher number of flips/accesses in a given cell. In spite of this, peak reductions are at least by 50% with respect to the baseline scheme.

The remaining CNNs show minor peak increases due to the same reason as stated above. On the contrary, AlexNet, VGG16, and SqueezeNet show peak reductions. This is due to the adjusted buffer size changes the bank rotation assignment in such a way that reduces the flip and access peaks with respect to the fixed 2MB buffer size.

Overall, the average maximum flip and access peak reductions are by 62 and 79%, respectively, compared to the baseline design.

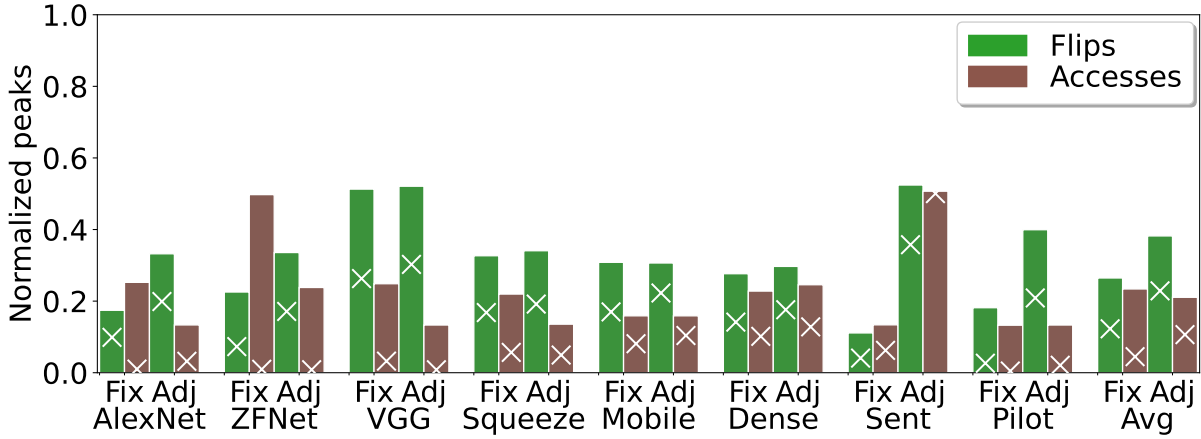


Figure 5.4: Normalized highest bit flip and access peaks of the Gated-CNN approach with respect to the baseline design. Results differentiate between a fixed I/O buffer size (Fix) and a buffer size adjusted to the largest layer (Adj).

5.4. Threshold Voltage Degradation

This section analyzes the threshold voltage degradation (dV_{th}) caused by the studied aging phenomena in all the transistors. The voltage degradation has been obtained for a 3-year lifetime [73]. The inference of the chosen subset of images is repeated over and over until the established lifetime is reached [42]. Results differentiate between the two types of aging effects: NBTI-induced wearout in PMOS transistors (T_{Pi}) is derived from Equation 1.7, whereas HCI-induced degradation in NMOS transistors is computed from Equation 1.8. In turn, NMOS transistors are distinguished between inverter loop (T_{Ni}) and pass transistors (T_{Wi}). Results have been obtained assuming a fixed 2 MB size for the I/O buffer.

Figure 5.5a shows the normalized dV_{th} distribution of the baseline and Gated-CNN with respect to the maximum dV_{th} after 3 years. The plotted results refer to the NBTI degradation affecting the T_{Pi} transistors. As observed, the proposed approach reduces the NBTI stress in all the transistors, since boxes and whiskers are located below those of the baseline in all the benchmarks. In addition, the heights of the boxes referring to Gated-CNN are shorter than those of the baseline in most benchmarks. That is, a more homogeneous NBTI degradation across all the transistors is achieved by the proposed Gated-CNN approach in at least a 50% of the cells.

The NBTI-induced dV_{th} savings for the worst-case T_{Pi} transistors range from 20 (AlexNet) to 51% (SentimentalNet). The average dV_{th} reduction for all these transistors is by 49%.

Figure 5.5b plots the normalized dV_{th} distribution caused by the HCI effect in the T_{Ni} transistors. Unlike the NBTI effect, some boxes of Gated-CNN stay above those of the baseline approach as a consequence of spreading out flips across NMOS transistors. In other words, HCI is more uniformly distributed across transistors, which might imply distribution percentiles above those

of the baseline. Nevertheless, like the NBTI effect, the HCI distribution helps Gated-CNN to obtain thinner boxes and largely reduce the dV_{th} in the worst-case transistors.

The HCI-induced dV_{th} reduction for the worst-case T_{Ni} transistors ranges from 28 (VGG) to 67% (SentimentalNet). The average dV_{th} savings for all these transistors is by 68%

Finally, Figure 5.5c focuses on the HCI effect in the T_{Wi} transistors. Similarly to the previous results, Gated-CNN obtains thinner boxes with respect to the baseline in most benchmarks, whereas the worst-case dV_{th} of the transistors is substantially reduced.

The HCI-induced dV_{th} savings for the worst-case T_{Wi} transistors vary between 29 (ZFNet) and 64% (PilotNet), whereas the average dV_{th} savings for all these transistors is by 85%.

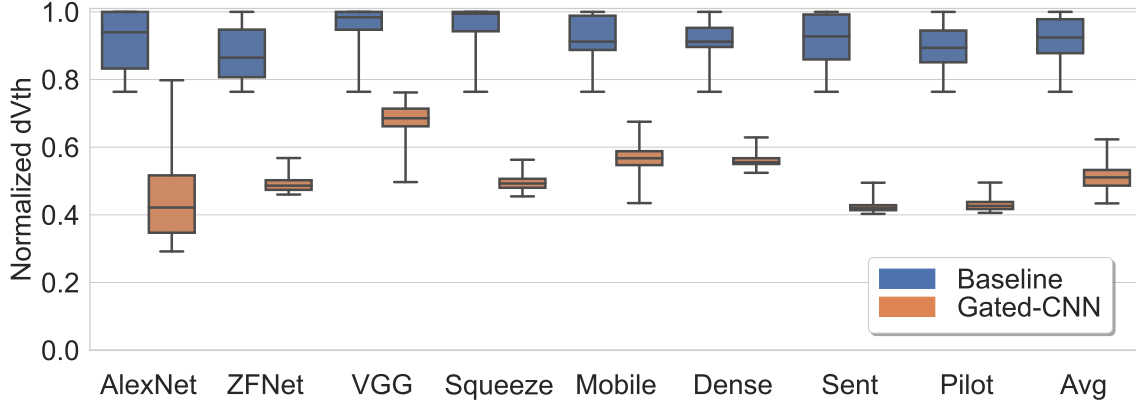
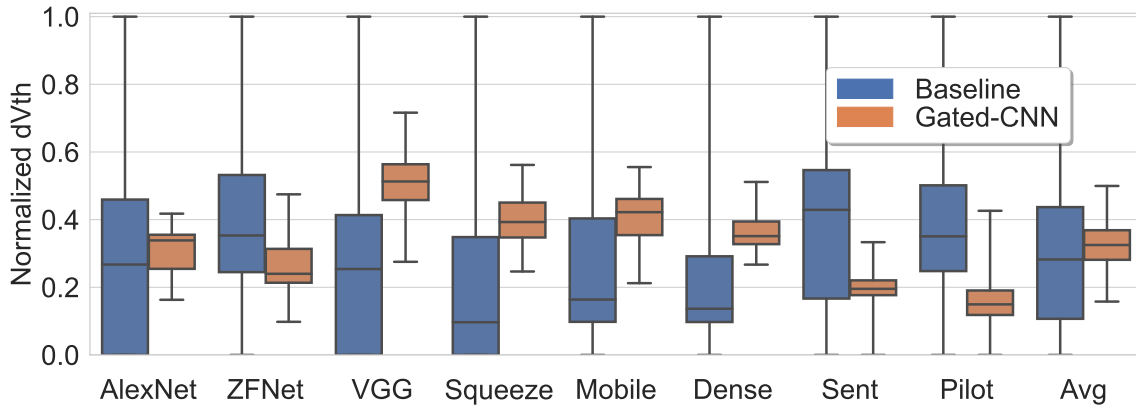
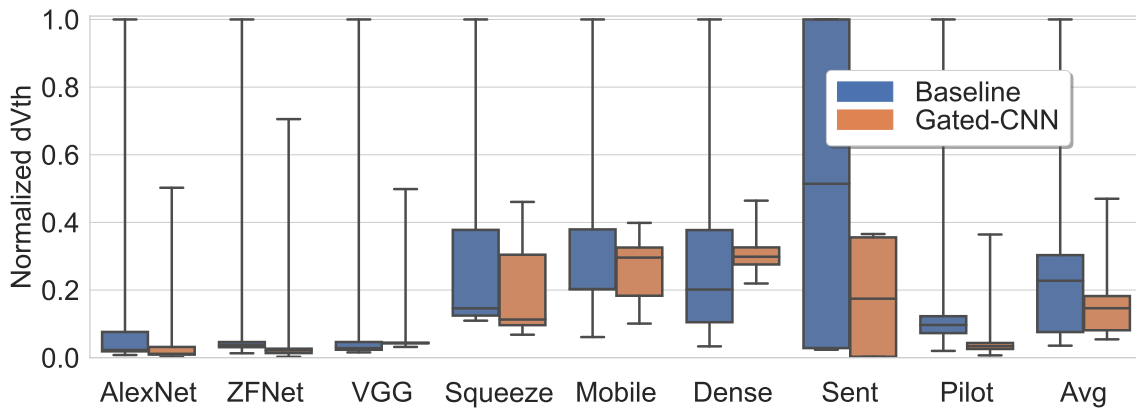
(a) T_{Pi} transistors (NBTI-induced aging)(b) T_{Ni} transistors (flip HCI-induced aging)(c) T_{Wi} transistors (access HCI-induced aging)

Figure 5.5: Normalized dV_{th} for every cell transistor with respect to the maximum dV_{th} after a 3-year lifetime, distinguishing among different types of transistors (aging affects).

6 | Related Work

Prior related work focusing on aging-aware mechanisms for on-chip memories can be classified into bit-flipping techniques, modifying the design of 6T SRAM cells, data rotation schemes, and power-gating approaches.

6.1. Bit-Flipping Approaches

A significant amount of work has addressed the NBTI wearout by periodically inverting the memory contents. The Penelope processor complements the contents of idle CPU cache blocks and registers [74]. Gebregiorgis *et al.* attack the same memory structures by identifying bit positions with an optimal NBTI signal probability and inverting the remaining bits according to such signals [75]. The iRMW mechanism flips the CPU cache contents depending on the type of write operation [76]. Similarly, recent FPGA designs have also relied on bit-flipping techniques to address aging effects in both the combinational and memory CMOS-based circuits [77]. In CNN accelerators, the state-of-the-art DNN-Life technique has faced the NBTI effect in the on-chip SRAM buffer of CNN accelerators storing weights of neural networks [16]. DNN-Life proposes to periodically invert the weights buffer contents to balance the duty cycle distribution. In particular, the technique encodes weights in such a way that bits to be written in this buffer are randomized. After a read operation, weights are decoded back to the original value before feeding the processing elements.

Overall, the above bit-flipping techniques effectively balance the cell duty cycles. However, these mechanisms come at the cost of aggravating the HCI effect.

6.2. SRAM Cell Circuit

Recovery Boosting focuses on NBTI mitigation in CPU memories by modifying the memory cells in such a way that both the ground voltage and the bitlines are raised to V_{dd} when cell contents are invalid [78]. Kothawade *et al.* implement CPU register files combining normally-sized transistors with NBTI-resilient up-sized transistors in a manner that the latter store the output of aging-inducing instructions [79]. Ricketts *et al.* investigate the usage of robust 8T SRAM cells together with power saving techniques to minimize the impact of NBTI [61]. Gong *et al.* employ 8T cells to implement the most significant bits from CPU integer registers, since

they present highly biased ‘0’ duty cycles, whereas 6T cells are used for the remaining bits [62]. Dounavi *et al.* take advantage of a cell aging prediction mechanism along with a repairing technique using spare cells [80].

Unfortunately, either spare cells, up-sizing, or adding more transistors to the 6T SRAM cell design imply high area and power overheads, preventing the adoption of these techniques in area and power-constrained designs like CNN accelerators.

6.3. Rotation Schemes

Dynamic Indexing identifies idle cache blocks where V_{dd} can be reduced to alleviate NBTI degradation, and uniformly spread out such idle blocks across the cache with different index update functions [81]. Colt attacks both the HCI and NBTI effects in CPU caches [82]. The former effect is minimized by applying a cache set rotation using an LFSR, whereas the latter effect is combated by periodically complementing the memory contents. Proactive Recovery establishes a suspended NBTI wearout mode on a rotating basis thanks to including spare cache memory arrays in the CPU design [83]. Other works focus on NBTI mitigation in CPU registers, where both inter and intra-register bit rotation mechanisms are performed, either splitting the register in two halves [66], introducing a barrel-shifter between the address decoder and the memory cells [67], or proposing alternative physical-to-logical register mappings [84].

These techniques cannot be directly applied to domain-specific CNN accelerators since: i) the memory organization and management in general-purpose processors is different, and ii) the memory demands and the temporal/spatial localities of general-purpose applications largely differ with respect to those of CNN applications.

6.4. Power-Gating

Power-gating has been previously exploited in CPU and GPU architectures to alleviate transistor aging. Valero *et al.* focus on CPU caches by power gating zero data bytes and rearranging the bytes of the cache blocks to uniformly distribute power-off cycles among all the bytes [68]. Such a rotation scheme is ineffective in CNN accelerators according to the characterization study shown in Chapter 3. In addition, the percentage of null bytes in CNN activations can be rather low (e.g., 13% in AlexNet). Finally, fine-grain power-gating schemes impose a large area overhead, making such approaches impractical for area-constrained embedded systems.

The ARGO approach exploits the observation that some GPU registers are never used, and consequently power gates such registers [69]. The RC+RAR technique also exploits this observation, and increases the power-gating opportunities by compressing entire GPU registers using a variation of the base-delta-immediate compression algorithm [85]. In addition, both ARGO and RC+RAR approaches modify the wavefront register allocation to distribute switch-off cycles

among all the registers. The under-utilization of GPU register files is caused by badly programmed applications where the underlying GPU hardware is not taken into account. Contrary to such general-purpose applications, the memory under-utilization of CNN applications dynamically changes at runtime and is caused by the unique memory demands of these applications, requiring specific solutions for CNN accelerators.

7 | Conclusions and Future Work

This chapter exposes the main conclusions of this work, the research contributions of the candidate, and a list of future directions.

7.1. Conclusions

Negative Bias Temperature Instability (NBTI) and Hot Carrier Injection (HCI) are two of the main aging phenomena that compromise the system's lifetime reliability. Both effects degrade the transistor's threshold voltage (V_{th}) over time, making transistors slower and eventually resulting in timing violations. These effects are especially critical in those transistors used to implement SRAM memories because they are permanently aging. In this sense, NBTI affects transistors when a cell stores a given logic value for a long period (i.e., a long duty cycle), whereas HCI manifests when the cell content is accessed and flipped.

On the other hand, the slowdown of the transistor scaling has become one of the major drivers toward domain-specific accelerators, which deliver a higher performance-to-power ratio with respect to general-purpose processors. Convolutional Neural Network (CNN) accelerators have consolidated as commodity devices that speed up the inference process required on classification and regression tasks. These accelerators usually implement large on-chip SRAM memories to cache the neural network parameters consisting of weights and activations, thereby minimizing costly off-chip memory accesses.

This thesis has identified those cell transistors of on-chip SRAM activation memories of CNN accelerators that age the most from the perspective of both NBTI and HCI effects. Based on this information, this work has presented Gated-CNN, a novel microarchitectural technique that reduces the largest duty cycles by power gating specific memory cells, and evenly distributes the access and flip patterns across all the cells with an address rotation mechanism.

Experimental results have shown that Gated-CNN extends the lifetime of the cell transistors with average V_{th} degradation savings at least as much as 49% depending on the type of transistor.

7.2. Research Contributions

As a result of the work carried out during this thesis, the following works have been published:

1. Nicolás Landeros Muñoz, Alejandro Valero, Rubén Gran Tejero, and Davide Zoni. *Gated-CNN: Combating NBTI and HCI aging effects in on-chip activation memories of Convolutional Neural Network accelerators*. Elsevier Journal of Systems Architecture, pp. 1-15, 2022. ISSN: 1383-7621. Ranked Q1 in 2022 JCR category COMPUTER SCIENCE, HARDWARE & ARCHITECTURE.
2. Nicolás Landeros Muñoz, Alejandro Valero, Rubén Gran Tejero, and Davide Zoni. *Tratamiento Antiedad en Aceleradores de Redes Neuronales Convolucionales*. In *Actas de las XXXII Jornadas de Paralelismo*, pp. 329-338, 2022. ISBN: 978-84-1302-185-0.

The candidate has also collaborated in the development of a research line focusing on the design of fault-tolerant microarchitectural mechanisms in on-chip memories of CNN accelerators for energy-saving purposes. In particular, the following works mitigate the impact of permanent faults on the CNN accuracy when lowering the power supply voltage of on-chip memories below the minimum safety voltage:

1. Yamilka Toca-Díaz, Nicolás Landeros Muñoz, Rubén Gran Tejero, and Alejandro Valero. *On Fault-Tolerant Microarchitectural Techniques for Voltage Underscaling in On-Chip Memories of CNN Accelerators*. In Proceedings of the 26th Euromicro Conference on Digital System Design, pp. 138-145, 2023. ISBN: 979-8-3503-4419-6.
2. Yamilka Toca-Díaz, Nicolás Landeros Muñoz, Rubén Gran Tejero, and Alejandro Valero. *Diseño de Memorias On-Chip para Aceleradores CNN Alimentados a Baja Tensión*. To appear in *Actas de las XXXIII Jornadas de Paralelismo*, 2023.
3. Yamilka Toca-Díaz, Nicolás Landeros Muñoz, Alejandro Valero, and Rubén Gran Tejero. *Impact on the Accuracy of Aggressive Voltage Underscaling in CNN Accelerators*. In Proceedings of the 18th International Summer School on Advanced Computer Architecture and Compilation for High-Performance and Embedded Systems, pp. 47-50, 2022. ISBN: 978-88-947027-0-5.

7.3. Future Directions

As for future work, this thesis can be extended in three main directions:

- Characterizing other types of aging effects in CNN accelerators such as time dependent dielectric breakdown and electromigration effects, and consequently exploring new microarchitectural techniques contributing to the device lifetime extension.

- Extending the current anti-aging technique to other types of neural networks such as recurrent neural networks, which could eventually lead to the study of another type of accelerator.
- Exploring and evaluating the potential use of non-volatile memories such as Domain Memory Wall in a neural network accelerators to address energy consumption and reliability.

Bibliography

- [1] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 269–284, 2014.
- [2] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam. DaDianNao: A Machine-Learning Supercomputer. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 609–622, 2014.
- [3] Patrick Judd, Jorge Albericio, Tayler Hetherington, Tor M. Aamodt, and Andreas Moshovos. Stripes: Bit-Serial Deep Neural Network Computing. In *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 1–12, 2016.
- [4] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. EIE: Efficient Inference Engine on Compressed Deep Neural Network. In *Proceedings of the 43rd International Symposium on Computer Architecture*, pages 243–254, 2016.
- [5] Norman P. Jouppi, Cliff Young, Nishant Patil, David A. Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. In-Datacenter Performance Analysis of a Tensor Processing Unit. In

- Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 1–12, 2017.
- [6] Yu-Hsin Chen, Tushar Krishna, Joel S. Emer, and Vivienne Sze. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE Journal of Solid-State Circuits*, 52(1):127–138, 2017.
 - [7] Dongjoo Shin, Jinmook Lee, Jinsu Lee, and Hoi-Jun Yoo. 14.2 DNPU: An 8.1TOPS/W reconfigurable CNN-RNN processor for general-purpose deep neural networks. In *Proceedings of the IEEE International Solid-State Circuits Conference*, pages 240–241, 2017.
 - [8] Hardik Sharma, Jongse Park, Naveen Suda, Liangzhen Lai, Benson Chau, Vikas Chandra, and Hadi Esmaeilzadeh. Bit Fusion: Bit-Level Dynamically Composable Architecture for Accelerating Deep Neural Network. In *Proceedings of the ACM/IEEE 45th Annual International Symposium on Computer Architecture*, pages 764–775, 2018.
 - [9] Alberto Delmas Lascorz, Patrick Judd, Dylan Malone Stuart, Zissis Poulos, Mostafa Mahmoud, Sayeh Sharify, Milos Nikolic, Kevin Siu, and Andreas Moshovos. Bit-Tactical: A Software/Hardware Approach to Exploiting Value and Bit Sparsity in Neural Networks. In *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 749–763, 2019.
 - [10] Jinmook Lee, Changhyeon Kim, Sanghoon Kang, Dongjoo Shin, Sangyeob Kim, and Hoi-Jun Yoo. UNPU: An Energy-Efficient Deep Neural Network Accelerator With Fully Variable Weight Bit Precision. *IEEE Journal of Solid-State Circuits*, 54:173–185, 2019.
 - [11] Adrián Alcolea Moreno, Javier Olivito, Javier Resano, and Hortensia Mecha. Analysis of a Pipelined Architecture for Sparse DNNs on Embedded Systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 28(9):1993–2003, 2020.
 - [12] Jaehyeong Sim, Somin Lee, and Lee-Sup Kim. An Energy-Efficient Deep Convolutional Neural Network Inference Processor With Enhanced Output Stationary Dataflow in 65-nm CMOS. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 28(1):87–100, 2020.
 - [13] Turki Alnuayri, Saqib Khursheed, Antonio Leonel Hernández Martínez, and Daniele Rossi. Differential Aging Sensor Using Subthreshold Leakage Current to Detect Recycled ICs. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 29(12):2064–2075, 2021.
 - [14] T. Nigam, B. Parameshwaran, and G. Krause. Accurate product lifetime predictions based on device-level measurements. In *Proceedings of the IEEE International Reliability Physics Symposium*, pages 634–639, 2009.
 - [15] Annachiara Ruospo, Ernesto Sanchez, Marcello Traiola, Ian O’Connor, and Alberto Bo-

- sio. Investigating Data Representation for Efficient and Reliable Convolutional Neural Networks. *Elsevier Microprocessors and Microsystems*, 86:1–14, 2021.
- [16] Muhammad Abdullah Hanif and Muhammad Shafique. DNN-Life: An Energy-Efficient Aging Mitigation Framework for Improving the Lifetime of On-Chip Weight Memories in Deep Neural Network Hardware Architectures. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition*, pages 729–734, 2021.
- [17] Tom Mitchell. *Machine Learning*. McGraw Hill Education, 1997.
- [18] Deepika Yadav. Different machine learning models. <https://medium.com/@yadavdeepika729/different-machine-learning-models-2387575e64cd>, 2022. [Online; accessed 10-August-2023].
- [19] TIBCO. What is a neural network? <https://www.tibco.com/reference-center/what-is-a-neural-network>, 2021. [Online; accessed 10-August-2023].
- [20] Yongmei Ren, Jie Yang, Qingnian Zhang, and Zhiqiang Guo. Multi-feature fusion with convolutional neural network for ship classification in optical images. *Applied Sciences*, 9(20), 2019.
- [21] Nick McCullum. Deep Learning Neural Networks Explained in Plain English. <https://www.freecodecamp.org/news/deep-learning-neural-networks-explained-in-plain-english/>, 2020. [Online; accessed 10-August-2023].
- [22] Dive into Deep Learning. Convolutions for Images. http://d2l.ai/chapter_convolutional-neural-networks/conv-layer.html. [Online; accessed 10-August-2023].
- [23] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. Efficient processing of deep neural networks: A tutorial and survey. *CoRR*, abs/1703.09039, 2017.
- [24] Atul Pandey. Depth-wise Convolution and Depth-wise Separable Convolution. <https://medium.com/@zurister/depth-wise-convolution-and-depth-wise-separable-convolution-37346565d4ec>, 2018.
- [25] Anil Bhatt. Max-Pooling, Combining Channels using 1×1 convolutions, Receptive Field calculation. <https://www.linkedin.com/pulse/max-pooling-combining-channels-using-11-convolutions-field-n-bhatt>, 2020. [Online; accessed 10-August-2023].
- [26] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.
- [27] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. Eyeriss: A spatial architecture for energy-

- efficient dataflow for convolutional neural networks. 2016. In 43rd ACM/IEEE International Symposium on Computer Architecture (ISCA).
- [28] Maurizio Capra, Beatrice Bussolino, Alberto Marchisio, Muhammad Shafique, Guido Masera, and Maurizio Martina. An updated survey of efficient hardware architectures for accelerating deep convolutional neural networks. *Future Internet*, 12(7), 2020.
 - [29] oneDNN Documentation. Understanding Memory Formats. https://oneapi-src.github.io/oneDNN/dev_guide_understanding_memory_formats.html, 2022. [Online; accessed 08-September-2023].
 - [30] Sangheon Kwon, Kyungmin Lee, Yoonsoo Kim, Kyungah Kim, Changmin Lee, and Won Woo Ro. Measuring error-tolerance in sram architecture on hardware accelerated neural network. In *2016 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*, pages 1–4, 2016.
 - [31] Guanpeng Li, Siva Kumar Sastry Hari, Michael Sullivan, Timothy Tsai, Karthik Pattabiraman, Joel Emer, and Stephen W. Keckler. Understanding error propagation in deep learning neural network (dnn) accelerators and applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, New York, NY, USA, 2017. Association for Computing Machinery.
 - [32] Brunno F. Goldstein, Victor C. Ferreira, Sudarshan Srinivasan, Dipankar Das, Alexandre S. Nery, Sandip Kundu, and Felipe M. G. França. A lightweight error-resiliency mechanism for deep neural networks. In *2021 22nd International Symposium on Quality Electronic Design (ISQED)*, pages 311–316, 2021.
 - [33] Siva Kumar Sastry Hari, Michael B. Sullivan, Timothy Tsai, and Stephen W. Keckler. Making convolutions resilient via algorithm-based error detection techniques. *IEEE Transactions on Dependable and Secure Computing*, 19(4):2546–2558, 2022.
 - [34] Fernando Fernandes dos Santos, Pedro Foletto Pimenta, Caio Lunardi, Lucas Draghetti, Luigi Carro, David Kaeli, and Paolo Rech. Analyzing and increasing the reliability of convolutional neural networks on gpu. *IEEE Transactions on Reliability*, 68(2):663–677, 2019.
 - [35] Brandon Reagen, Paul Whatmough, Robert Adolf, Saketh Rama, Hyunkwang Lee, Sae Kyu Lee, José Miguel Hernández-Lobato, Gu-Yeon Wei, and David Brooks. Minerva: Enabling low-power, highly-accurate deep neural network accelerators. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 267–278, 2016.
 - [36] Le Ha Hoang, Muhammad Abdullah Hanif, and Muhammad Shafique. Ft-clipact: Resilience analysis of deep neural networks and improving their fault tolerance using clipped activation. *CoRR*, abs/1912.00941, 2019.

- [37] Elbruz Ozen and Alex Orailoglu. Boosting bit-error resilience of dnn accelerators through median feature selection. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(11):3250–3262, 2020.
- [38] Behnam Ghavami, Mani Sadati, Zhenman Fang, and Lesley Shannon. Fitact: Error resilient deep neural networks via fine-grained post-trainable activation functions. *CoRR*, abs/2112.13544, 2021.
- [39] Zitao Chen, Guanpeng Li, and Karthik Pattabiraman. A low-cost fault corrector for deep neural networks through range restriction. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 1–13, 2021.
- [40] Ussama Zahid, Giulio Gambardella, Nicholas J. Fraser, Michaela Blott, and Kees A. Visser. FAT: training neural networks for reliable inference under hardware faults. *CoRR*, abs/2011.05873, 2020.
- [41] R. Vattikonda, Wenping Wang, and Yu Cao. Modeling and Minimization of PMOS NBTI Effect for Robust Nanometer Design. In *Proceedings of the 43rd Design Automation Conference*, pages 1047–1052, 2006.
- [42] Abhishek Tiwari and Josep Torrellas. Facelift: Hiding and Slowing Down Aging in Multicores. In *Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture*, pages 129–140, 2008.
- [43] F. Oboril and M. B. Tahoori. ExtraTime: Modeling and Analysis of Wearout due to Transistor Aging at Microarchitecture-Level. In *Proceedings of the 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 1–12, 2012.
- [44] Stefanos Kaxiras, Z. Hu, and M. Martonosi. Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power. In *Proceedings of the 28th Annual International Symposium on Computer Architecture*, pages 240–251, 2001.
- [45] A. Calimera, E. Macii, and M. Poncino. Analysis of NBTI-Induced SNM Degradation in Power-Gated SRAM Cells. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 785–788, 2010.
- [46] T. Siddiqua and S. Gurumurthi. Enhancing NBTI Recovery in SRAM Arrays Through Recovery Boosting. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 20(4):616–629, 2012.
- [47] M. T. Rahman, D. Forte, J. Fahrny, and M. Tehranipoor. ARO-PUF: An Aging-Resistant Ring Oscillator PUF Design. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition*, pages 1–6, 2014.
- [48] Jeremy Howard. imagenette.

- [49] Jakob Kather, Cleo-Aron Weis, Francesco Bianconi, Susanne Melchers, Lothar Schad, Timo Gaiser, Alexander Marx, and Frank Zöllner. Multi-class texture analysis in colorectal cancer histology. *Scientific Reports*, 6:27988, 06 2016.
- [50] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.
- [51] Udacity. self-driving-car. <https://github.com/udacity/self-driving-car>, 2020. [Online; accessed 10-August-2023].
- [52] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems*, 25:1097–1105, 2012.
- [53] Matthew D Zeiler and Rob Fergus. Visualizing and Understanding Convolutional Networks. In *European Conference on Computer Vision*, pages 818–833. Springer, 2014.
- [54] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv preprint arXiv:1409.1556*, 2014.
- [55] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Praseen Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to End Learning for Self-Driving Cars, 2016.
- [56] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. *arXiv preprint arXiv:1602.07360*, 2016.
- [57] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016.
- [58] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *CoRR*, abs/1704.04861, 2017.
- [59] Jason Brownlee. How to Predict Sentiment From Movie Reviews Using Deep Learning (Text Classification). <https://machinelearningmastery.com/predict-sentiment-movie-reviews-using-deep-learning/>, 2016.
- [60] Amir Yazdanbakhsh, Kiran Seshadri, Berkin Akin, James Laudon, and Ravi Narayanaswami. An Evaluation of Edge TPU Accelerators for Convolutional Neural Networks, 2021.

- [61] A. Ricketts, J. Singh, K. Ramakrishnan, N. Vijaykrishnan, and D. K. Pradhan. Investigating the impact of nbtI on different power saving cache strategies. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition*, pages 592–597, 2010.
- [62] N. Gong, Jiang S., J. Wang, Aravamudhan B., K. Sekar, and R. Sridhar. Hybrid-Cell Register Files Design for Improving NBTI Reliability. *Elsevier Microelectronics Reliability*, 52(9–10):1865–1869, 2012.
- [63] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems, 2016.
- [64] Sheng Li, Ke Chen, Jung Ho Ahn, Jay B. Brockman, and Norman P. Jouppi. CACTI-P: Architecture-Level Modeling for SRAM-based Structures with Advanced Leakage Reduction Techniques. *HP Laboratories, Palo Alto, CA, USA. Technical Report HPL-2012-187*, 2012.
- [65] Muhammad Abdullah Hanif, Rachmad Vidya Wicaksana Putra, Muhammad Tanvir, Rehan Hafiz, Semeen Rehman, and Muhammad Shafique. MPNA: A massively-parallel neural array accelerator with dataflow optimization for convolutional neural networks. *CoRR*, abs/1810.12910, 2018.
- [66] H. Amrouch, T. Ebi, and J. Henkel. Stress Balancing to Mitigate NBTI Effects in Register Files. In *Proceedings of the 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 1–10, 2013.
- [67] Saurabh Kothawade, Koushik Chakraborty, and Sanghamitra Roy. Analysis and Mitigation of NBTI Aging in Register File: An End-To-End Approach. In *Proceedings of the 12th International Symposium on Quality Electronic Design*, pages 1–7, 2011.
- [68] Alejandro Valero, Negar Miralaei, Salvador Petit, Julio Sahuquillo, and Timothy M. Jones. On Microarchitectural Mechanisms for Cache Wearout Reduction. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(3):857–871, 2017.
- [69] Majid Namaki-Shoushtari, Abbas Rahimi, Nikil Dutt, Puneet Gupta, and Rajesh K Gupta. ARGO: Aging-aware GPGPU Register File Allocation. In *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*, pages 1–9, 2013.

- [70] Sangpil Lee, Keunsoo Kim, Gunjae Koo, Hyeran Jeon, Murali Annavaram, and Won Woo Ro. Improving Energy Efficiency of GPUs Through Data Compression and Compressed Execution. *IEEE Transactions on Computers*, 66(5):834–847, 2017.
- [71] Harold Pilo, Vinod Ramadurai, Geordie Bracer, John Gabric, Steve Lamphier, and Yue Tan. A 450ps Access-Time SRAM Macro in 45nm SOI Featuring a Two-Stage Sensing-Scheme and Dynamic Power Management. In *Proceedings of the IEEE International Solid-State Circuits Conference - Digest of Technical Papers*, pages 378–621, 2008.
- [72] Harold Pilo, Chad A. Adams, Igor Arsovski, Robert M. Houle, Steven M. Lamphier, Michael M. Lee, Frank M. Pavlik, Sushma N. Sambatur, Adnan Seferagic, Richard Wu, and Mohammad I. Younus. A 64Mb SRAM in 22nm SOI technology featuring fine-granularity power gating and low-energy power-supply-partition techniques for 37% leakage reduction. In *Proceedings of the IEEE International Solid-State Circuits Conference - Digest of Technical Papers*, pages 322–323, 2013.
- [73] Evelyn Mintarno, Vikas Chandra, David Pietromonaco, Robert Aitken, and Robert W Dutton. Workload-Dependent NBTI and PBTI Analysis for a sub-45nm Commercial Microprocessor. In *IRPS*, pages 1–6, 2013.
- [74] J. Abella, X. Vera, and A. González. Penelope: The NBTI-Aware Processor. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 85–96, 2007.
- [75] A. Gebregiorgis, M. Ebrahimi, S. Kiamehr, F. Oboril, S. Hamdioui, and M. B. Tahoori. Aging Mitigation in Memory Arrays Using Self-controlled Bit-flipping Technique. In *Proceedings of the 20th Asia South Pacific Design Automation Conference*, pages 231–236, 2015.
- [76] S. Ganapathy, R. Canal, A. González, and A. Rubio. iRMW: A Low-Cost Technique to Reduce NBTI-Dependent Parametric Failures in L1 Data Caches. In *Proceedings of the 32nd IEEE International Conference on Computer Design*, pages 68–74, 2014.
- [77] Mohammad Hadi Mottaghi, Mehdi Sedighi, and Morteza Saheb Zamani. FIFA: A Fully Invertible FPGA Architecture to Reduce BTI-Induced Aging Effects. *IEEE Transactions on Computers (Early Access)*, pages 1–11, 2021.
- [78] Taniya Siddiqua and Sudhanva Gurumurthi. Recovery Boosting: A Technique to Enhance NBTI Recovery in SRAM Arrays. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, pages 393–398, 2010.
- [79] S. Kothawade, D. M. Ancajas, K. Chakraborty, and S. Roy. Mitigating NBTI in the Physical Register File through Stress Prediction. In *Proceedings of the IEEE 30th International Conference on Computer Design*, pages 345–351, 2012.

- [80] Helen-Maria Dounavi, Yiorgos Sfikas, and Yiorgos Tsiatouhas. Aging Prediction and Tolerance for the SRAM Memory Cell and Sense Amplifier. *Springer Journal of Electronic Testing*, 37:65–82, 2021.
- [81] A. Calimera, M. Loghi, E. Macii, and M. Poncino. Dynamic Indexing: Leakage-Aging Co-Optimization for Caches. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(2):251–264, 2014.
- [82] E. Gunadi, A. A. Sinkar, N. S. Kim, and M. H. Lipasti. Combating Aging with the Colt Duty Cycle Equalizer. In *Proceedings of the 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 103–114, 2010.
- [83] Jeonghee Shin, V. Zyuban, P. Bose, and T. M. Pinkston. A Proactive Wearout Recovery Approach for Exploiting Microarchitectural Redundancy to Extend Cache SRAM Lifetime. In *Proceedings of the 35th International Symposium on Computer Architecture*, pages 353–362, 2008.
- [84] Ilya Tuzov, Pablo Andreu, Laura Medina, Tomas Picornell, Antonio Robles, Pedro López, Jose Flich, and Carles Hernández. Improving the Robustness of Redundant Execution with Register File Randomization. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pages 1–9, 2021.
- [85] A. Valero, F. Candel, D. Suárez-Gracia, S. Petit, and J. Sahuquillo. An Aging-Aware GPU Register File Design Based on Data Redundancy. *IEEE Transactions on Computers*, 68(1):4–20, 2019.
- [86] Joel Emer, Vivienne Sze, Yu-Hsin Chen, and Tien-Ju Yang. Tutorial on Hardware Accelerators for Deep Neural Networks. <http://eyeriss.mit.edu/tutorial.html>, 2019. [Online; accessed 10-August-2023].
- [87] Charles E Leiserson. Systolic arrays (for vlsi). In *Sparse Matrix Proceedings 1978*, volume 1, pages 256–282. Society for industrial and applied mathematics Philadelphia, PA, USA, 1979.
- [88] ankur6ue. Understanding Matrix Multiplication on a Weight-Stationary Systolic Architecture. <https://www.telesens.co/2018/07/30/systolic-architectures/>, 2018. [Online; accessed 10-August-2023].
- [89] Murugan Sankaradass, Venkata Jakkula, Hari Cadambi, Srimat Chakradhar, Igor Durdanovic, Eric Cosatto, and H.P. Graf. A massively parallel coprocessor for convolutional neural networks. pages 53–60, 07 2009.
- [90] Srimat Chakradhar, Murugan Sankaradas, Venkata Jakkula, and Srihari Cadambi. A dynamically configurable coprocessor for convolutional neural networks. *SIGARCH Comput. Archit. News*, 38(3):247–257, jun 2010.

- [91] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. Shidiannao: Shifting vision processing closer to the sensor. *SIGARCH Comput. Archit. News*, 43(3S):92–104, jun 2015.
- [92] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1737–1746, Lille, France, 07–09 Jul 2015. PMLR.

A | Appendix A: Training and Testing of a Neural Network

For a neural network to work it must first be trained, the objective of the training is to find the weights such that the predictions of the network are as accurate as possible, this is equivalent to finding the weights w such that $\frac{\partial E}{\partial w} = 0$ where E is a function that measures the performance of the network, also known as error or loss function. Unfortunately, there is no closed solution to obtain the optimal value of w , so in practice iterative methods with optimal local solutions, such as gradient descent are used; This method is described below.

First, the weights are initialized with random values, and labeled data is reserved for training, this dataset is called training dataset and is decomposed into batches. For each batch the predicted value by the network $g(x)$ is obtained and compared with the actual label y , this comparison is carried out by means of the loss function, generally for regression we use mean squared error, equation A.1a and for cross-entropy classification, equation A.1b.

$$E = \sum_{n=1}^N (y_n - g(x_n))^2, \quad (\text{A.1a})$$

$$E = - \sum_{n=1}^N y_n^T \cdot \log(g(x_n)), \quad (\text{A.1b})$$

Where N corresponds to batch size, $g(x_n)$ and y_n are the prediction and the target value, respectively, which are real values in the case of regression or a one-hot encoded vector in classification.

After each batch is processed, the weights are updated using equation A.2 where η is called the learning rate and controls how large the steps are in the direction of minimization.

$$w^{k+1} = w^k - \eta \frac{\partial E}{\partial w} \Big|_{w^k} \quad (\text{A.2})$$

This is repeated multiple times; usually another data set is used to determine when to stop iterating, thus avoiding overfitting in the training data. After the network has been trained, it

can be used to predict new data; this process is called testing or inference.

B | Appendix B: Computational Cost of Convolutional Neural Networks during Inference

Neural networks are characterized by being demanding of computational capacity; during training, this is usually not a problem because this process only occurs once and commonly on machines with greater computing power than the final device; on the other hand, the inference is usually performed in embedded devices where resources are limited. Therefore, it is relevant to study two important metrics in a neural network: number of multiplication and addition operations (MACs) and number of accesses to memory; below we analyze these values for each layer.

Taking into account the parameters defined in Table 1.1 and according to 1.1 to process each output activation, $C_i \cdot k_h \cdot k_w + 1$ MACS are needed, therefore the required number of MACs required by the convolution layer and denoted as C_{macs} is:

$$C_{macs} = N \cdot W_o \cdot H_o \cdot F \cdot (C_i \cdot k_h \cdot k_w + 1) \quad (B.1)$$

Each MAC requires 4 memory accesses (1 weight read, 1 activation read, 1 partial sum read and 1 updated partial sum write), so in the worst case this convolutional layer will require $4 * C_{macs}$ memory accesses, taking for example the first layer of VGG16 a popular CNN architecture and assuming a batch of 8 samples we have $(N, W_i, H_i, W_o, H_o, F, k_w, k_h, C_i) = (8, 224, 224, 224, 224, 64, 3, 3, 3) \implies \approx 720$ millions of MACS then it would require 2880 Millions of memory accesses which is clearly prohibitive even for high computation devices, fortunately, this type of layers have a high data reusability, three reuse patterns are presented below.

- **Convolutional Reuse:** Each subgrid of the activation map uses the same filters, and if the stride is smaller than the size of the kernel, the subgrids share activations with each other.
- **Filter Reuse:** All activation maps within a batch use the same filters.
- **FMap Reuse:** All filters use the same activation maps.

Figure B.1 shows graphically the 3 types of reuses listed.

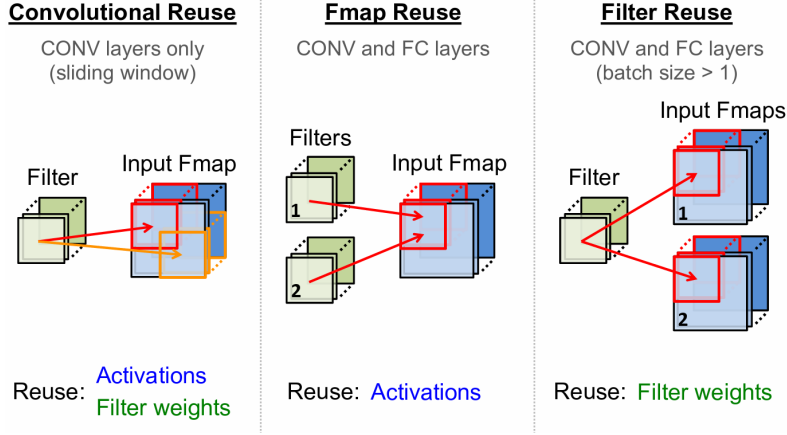


Figure B.1: Data and weight reuse in convolutional layers, from [86]

If a device is able to take full advantage of data reuse, avoiding rereading the same activations or weights and keeps in local memory of fast access the temporary sums, then it would only need to read each activation of the input feature map and each weight once, hence $B \cdot W_i \cdot H_i \cdot C + K_w \cdot K_h \cdot F$ reads, in the previous example this is equivalent to 1.2M reads 2400 times less than the original 2880 million.

Another method of computing the convolution is to take the matrix representation of the convolution operation and process it as a matrix multiplication using algorithms designed for this purpose.

B.1. Depthwise Convolution Layer

Using the parameters defined in Table 1.1 and according to 1.3 to process each output activation, $k_h \cdot k_w + 1$ MACS are needed, therefore the required number of MACs required by the convolution layer and denoted as DWC_{macs} is:

$$DWC_{macs} = N \cdot W_o \cdot H_o \cdot C_i \cdot (K_h \cdot K_w + 1) \quad (B.2)$$

The number of MACs is F (number of filters) times less than the one required in a convolutional layer; however, this decrease in operations leads to no more reuse of input feature map activations (FMap reuse) which limits the opportunities for the reduction of memory accesses required by the convolutional layer.

B.2. Dense Layer

Consider a generic layer with N inputs and M outputs, then one simply needs $(M + 1) * N$ MACs, and thus $4(M + 1) * N$ memory accesses. Unlike the convolutional layer, the Dense layer has no reuse of the convolutional type, and therefore only the other two types can be exploited.

B.3. Pooling Layer

Considering a pooling layer with filter size $K \times K$ each output activation requires reading K^2 activations and computing a $\max()$ function on K^2 inputs for MaxPooling or K^2 MAC's in the case of Averagepooling, the number of output activations equals the number of input activations divided by K^2 . Typically the read/write cost of these layers can be eliminated if the operation is performed in conjunction with the processing of a convolutional layer; i.e., instead of writing the convolutional layer outputs and reading them back to the pooling layer, pooling is performed directly on the convolution outputs.

B.4. Activation function and normalization operation

The normalization operation during inference is simply a transformation of the data, according to equation 1.6 we can see that a read of the input activation, a subtraction and division for the normalization, and a MAC for the displacement-scaling and a write for each output activation is enough.

The mathematical cost of the activation function varies greatly depending on your choice, for Relu a $\max()$ operation is enough, on the other hand, the sigmoid and hyperbolic tangent functions being complex trigonometric operations usually take multiple cycles. As for readings, all of them only need one read per output activation plus the writing of the same.

As in the case of pooling, the read/write cost of both operations can be eliminated if processed in conjunction with the convolutional or dense layer that precedes it.

C | Appendix C: Systolic Arrays

A systolic array is a type of spatial architecture first introduced in [87], consisting of a homogeneous array of processing elements communicating with each other; In its original form, each PE independently computes a partial result based on the data received from its upstream neighbors, stores the result within itself, and passes it downstream, the name systolic derives from systole as an analogy to the regular pumping of blood by the heart. Figure C.1 shows an example of a systolic array implementing three matrix multiplications $Y^a = W \times A$, $Y^b = W \times B$, $Y^c = W \times C$, in this case PEs communicate to the right the input of the systolic array and to the bottom the partial results, also the data of the matrix W have been previously loaded into PE registers.

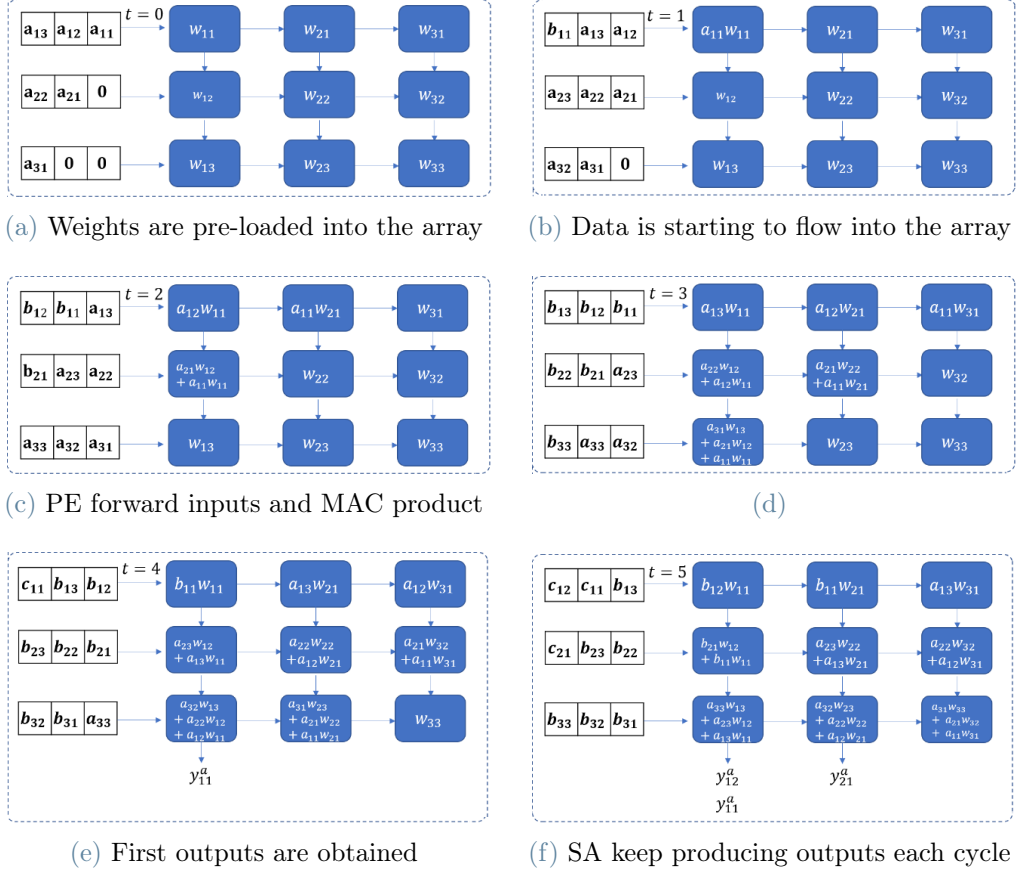


Figure C.1: 2-D Systolic array used for matrix multiplication [88].

C.1. Systolic Dataflows for Convolution

In order to maximize the different types of data reuse mentioned in B, multiple ways of mapping a convolution to a systolic array have been developed; Each of these dataflows will be presented below; to do so consider a two-dimensional systolic matrix as shown in Figure C.1 and the detailed nomenclature of Table C.1.

Table C.1: Systolic Array parameters

Parameter	Description
R, C	Number of rows and columns, respectively
i, j	indices of the PE within the array, $0 < i < R, 0 < j < C$
$I1_{i,j}$	left entry to the PE of indices i, j
$I2_{i,j}$	top entry to the PE of indices i, j
$RF_{i,j}$	data stored in the RF of the PE of indices i, j
$O1_{i,j}$	right output of the PE of indices i, j
$O2_{i,j}$	bottom output of the PE of indices i, j
IR_i	input to row i of the systolic array
IC_j	input to column j of the systolic array

C.1.1. Weight Stationary

Each filter is preloaded in a different column of the systolic array and stored in the RF of each PE, unrolled convolution windows from the input matrix are entered into the array from the left, each PE communicates to its left the inputs to the array and to the bottom the partial sums; Equation C.1 shows the configuration during the preloading of weights, and Equation C.2 the configuration for the convolution computation.

$$I2_{i,j}[t] = O2_{i-1,j}[t-1]; \quad I2_{0,j}[t] = IC_j[t] \quad (C.1a)$$

$$RF_{i,j}[t] = I2_{i,j}[t] \quad (C.1b)$$

$$O2_{i,j}[t] = I2_{i,j}[t] \quad (C.1c)$$

$$I1_{i,j}[t] = O1_{i-1,j}[t-1]; \quad I1_{i,0}[t] = IR_i[t] \quad (C.2a)$$

$$I2_{i,j}[t] = O2_{i,j-1}[t-1]; \quad I2_{0,j}[t] = 0 \quad (C.2b)$$

$$O1_{i,j}[t] = I1_{i,j}[t] \quad (C.2c)$$

$$O2_{i,j}[t] = I1_{i,j}[t] * RF_{i,j}[t] + I2_{i,j}[t] \quad (C.2d)$$

This type of dataflow maximizes convolutional and filter reuse, its obvious disadvantage is that it requires extra cycles to preload the weights; on the other hand, the output activations are conveniently collected in the last row of the array; examples of architectures that implement this dataflow can be found in [89, 90].

C.1.2. Output Stationary

In this type of dataflow, the inputs are entered into the systolic array from the left while the weights from above, each PE calculates an output activation independently and the partial sums are stored in the RF, the communication between PEs is limited to forwarding the weights and activations that enter the array; equation C.3 shows the operation performed by each PE.

$$I1_{i,j}[t] = O1_{i-1,j}[t-1]; \quad I1_{i,0} = IR_i \quad (C.3a)$$

$$I2_{i,j}[t] = O2_{i,j-1}[t-1]; \quad I2_{0,j} = IC_j \quad (C.3b)$$

$$RF_{i,j}[t] = I1_{i,j}[t] * I2_{i,j}[t] + RF_{i,j}[t-1] \quad (C.3c)$$

$$O1_{i,j}[t] = I1_{i,j}[t] \quad (C.3d)$$

$$O2_{i,j}[t] = I2_{i,j}[t] \quad (C.3e)$$

This dataflow minimizes the cost of accumulation; its disadvantage is that it requires extra cycles to obtain the output activations from each PE once they have been calculated, alternatively an auxiliary structure can be connected in which each PE deposits its activation once it is completed. Examples of architectures that implement this data flow can be found in [91, 92].

C.1.3. Input Stationary

Same principle as weight stationary, only that under this dataflow the input matrix is preloaded into the systolic array and the unrolled filters are fed into the array.

Figure C.2 shows a possible internal structure for a PE that can implement both weight stationary and output stationary dataflow, note that both outputs and RF input are multiplexed, the type of dataflow to be executed will define the value that the control signals take.

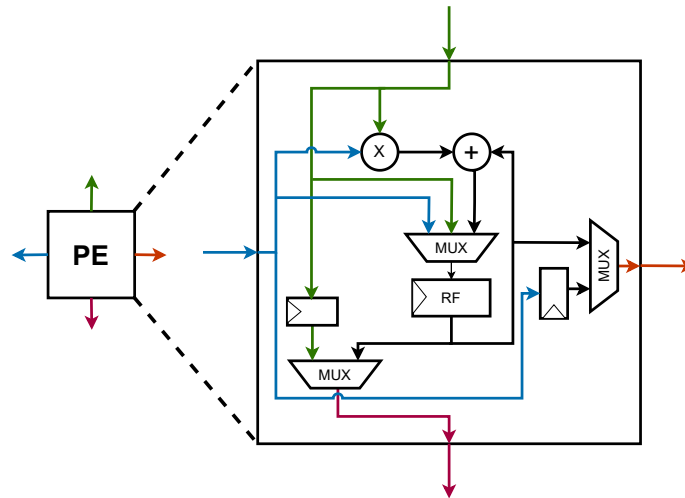
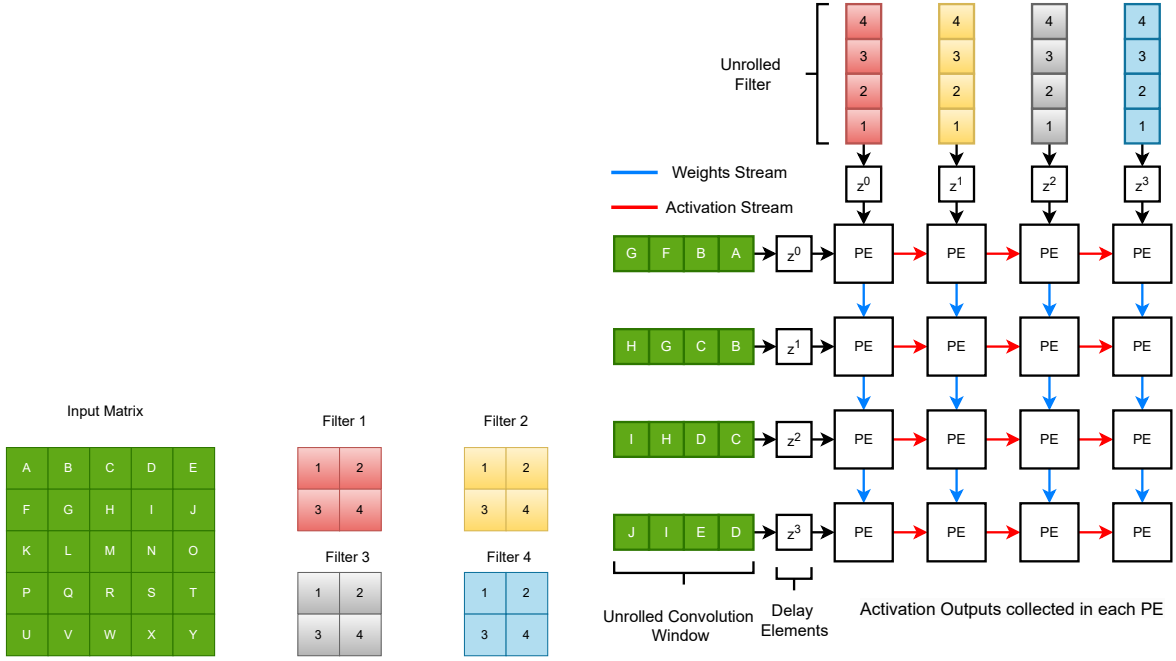
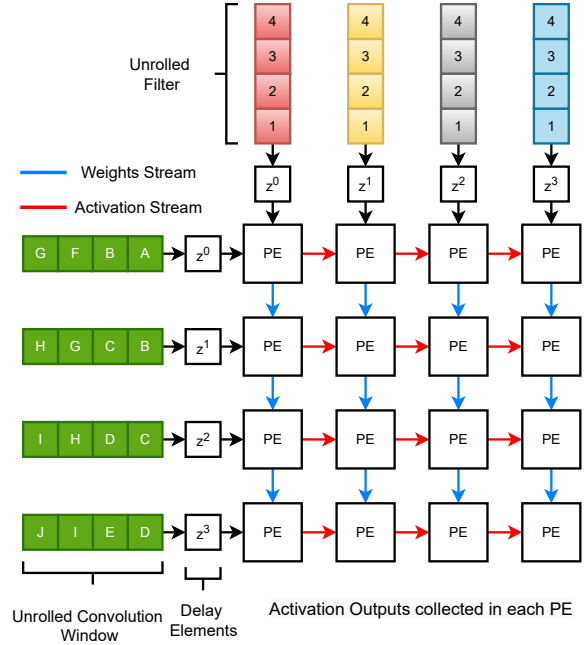


Figure C.2: Possible PE internal structure, mux control signals are omitted for simplicity.

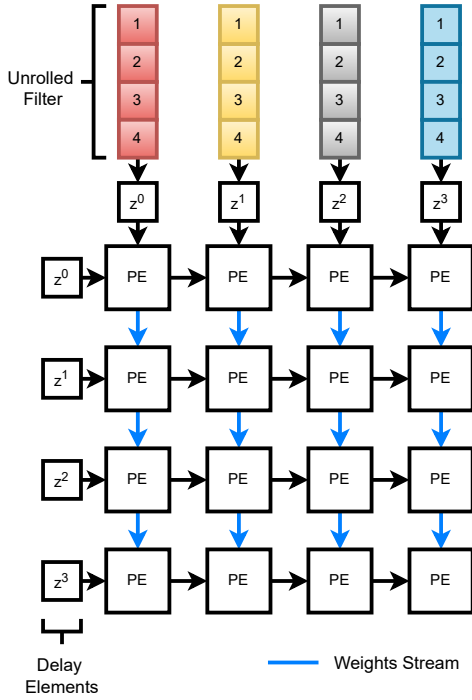
The following is an example of processing the first row of a one-channel convolution with four filters of size 2×2 with unitary stride in a 4×4 PE systolic array for the weight stationary and output stationary dataflows.



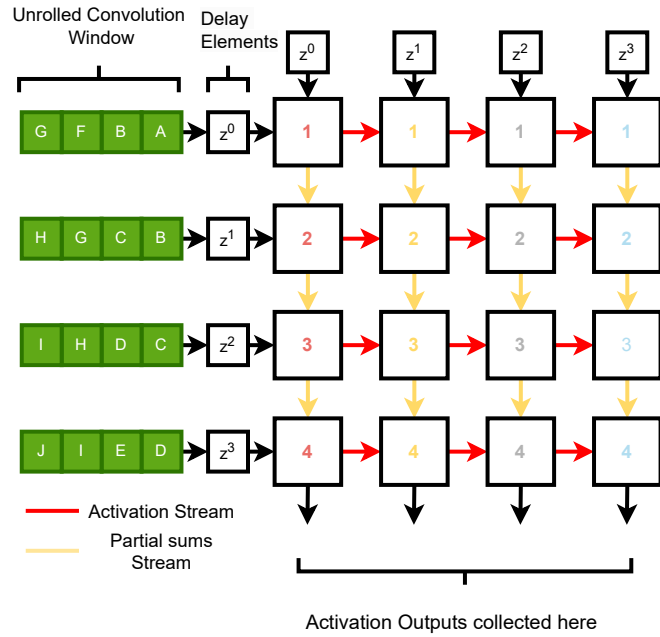
(a) Input Matrix and Filters of the convolution



(b) Output Stationary dataflow



(c) Weight Stationary: Pre-load phase



(d) Weight Stationary: Running phase

D | Appendix D: Fault Injection

In order to inject faults into a neural network and evaluate its performance, the value of the activations must be modified at layer level. Due to the operating restrictions of tensorflow, this modification cannot be carried out in run-time, but only in design phase, for which a new operation called "Aging" is added to the data flow graph.

The operation of aging is modelled by the pseudo-code described in the algorithm D.1, in which the layer takes as input the output activations of the previous layer, a vector with the indices of the activations with faults together with the bits to be modified; "Aging" converts the input activations into their corresponding binary values in fixed-point format and modifies the logical value of each bit that has a fault, following a stuck-at fault model. Finally, the faulty values are converted to their corresponding floating point value so that they can be processed by future layers. Since this operation will only be performed during inference, it is not necessary to define its differentiation for training.

Algorithm D.1 Aging custom layer

Require: A : Activations, $Active$: Active flag, I : indices of faults and faulty bits

Ensure: A_f : Faulty activations

```

1: if  $Active == True$  then
2:   Convert  $A$  in fixed point representation  $A_b$  {i.e  $0.0 \rightarrow 16b0000\_0000\_0000\_0000$ }
3:   for index and fault_info  $(i, F)$  in  $I$  do
4:     for bit position and stuck at value  $(p, sa)$  in  $F$  do
5:        $A_b[i][p] = sa$ 
6:     end for
7:   end for
8:   Convert faulty  $A_b$  back into the faulty float representation  $A_f$ 
9:   return  $A_f$ 
10: else
11:   return  $A$ 
12: end if

```

Similar to what is done for quantization (see chapter 3) aging operations are added to the network, in this case it is added after each operation/layer in which its output triggers are written to the I/O buffer of an accelerator, as shown in Figure D.1

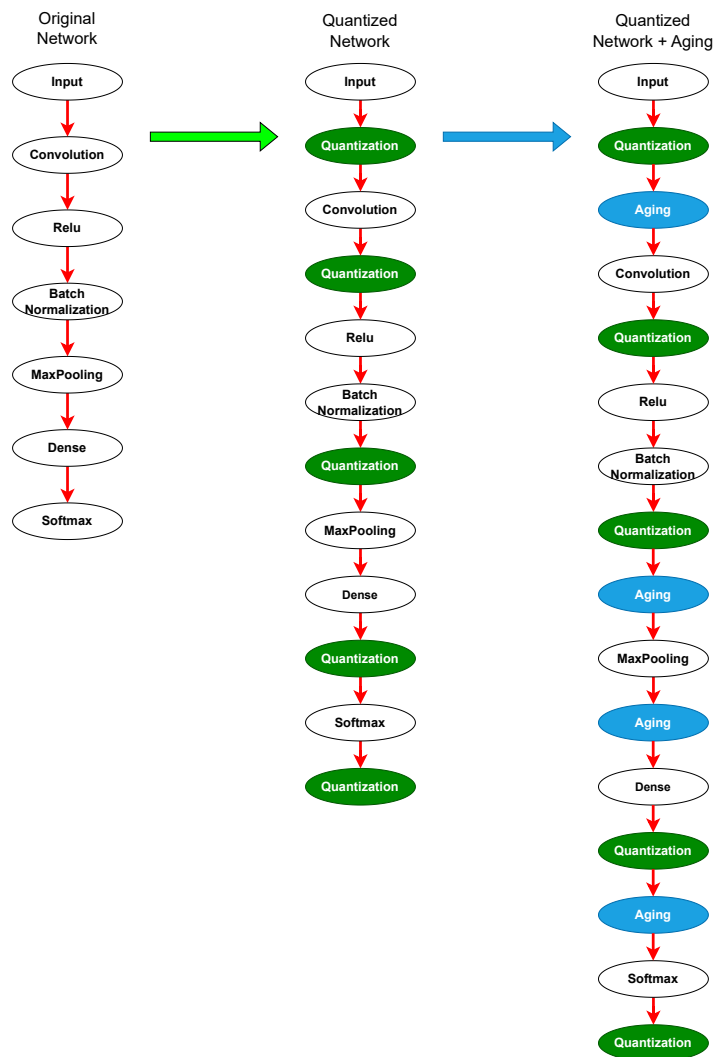


Figure D.1: Fault injection in a network using the incorporation of Aging layers.

List of Figures

1	Accuracy Drop due to faulty bit injections	2
1.1	classification of ML algorithms	6
1.2	Examples of neural network schematics	7
1.3	Neuron structure	8
1.4	Example of Convolution operation	9
1.5	Convolution as matrix multiplication	11
1.6	Depth-wise convolution example	11
1.7	MaxPooling example	12
1.8	Dense layer as a matrix multiplication.	13
1.9	Typical structure and memory layout of spatial CNN accelerators.	15
1.10	Three different ways to vectorize a 4-D feature map into a 1-D vector.	16
1.11	Structure of Eyeriss accelerator	16
1.12	Structure of TPU	17
1.13	Implementation of a 6T SRAM cell	20
2.1	Overview and dataflow of the baseline CNN accelerator.	25
2.2	Baseline bank organization	27
3.1	Quantized Network	31
3.2	Quantized Network Top-1 accuracy in function of the bit distribution	32
3.3	Loss function varying the number of bits of activations in PilotNet.	32
3.4	Output processing in an toy 3×3 systolic array	36
3.5	Dataflow in an toy 3×3 systolic array	37
3.6	Non-sequential layers used in SqueezeNet and DenseNet benchmarks.	39
3.7	Processing of sequential and non-sequential convolutions	40
3.8	‘0’ duty cycle distributions on each bit position of the 16-bit active activations.	41
3.9	Normalized bit flips on each bit position of the 16-bit active activations	42
3.10	Normalized number of accesses of the networks	43
4.1	Proposed Gated-CNN design	46
4.2	Driving example of the CNN-Gated component	48

5.1	Longest duty cycle distributions across all the benchmarks	52
5.2	Normalized highest bit flip and access peaks of Gated-CNN	53
5.3	Longest duty cycle distributions using fixed and adjusted buffer size	55
5.4	Normalized bit flip and access peaks for fixed and adjusted buffer size	56
5.5	Normalized dV_{th} for every cell transistor	58
B.1	Data and weight reuse in convolutional layers	80
C.1	2-D Systolic array used for matrix multiplication	83
C.2	Possible PE internal structure, mux control signals are omitted for simplicity. . .	85
D.1	Fault injection in a network using the incorporation of Aging layers.	88

List of Tables

1.1	Parameters used in the context of a convolutional layer	10
1.2	Main parameters of the NBTI and HCI aging model	21
2.1	Overview of the studied CNN benchmarks	24
2.2	Software versions	28
3.1	Quantization details for each benchmark.	33
C.1	Systolic Array parameters	84

