TU WIEN Informatics

Robust and Energy Efficient Deep Learning Systems

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

Muhammad Abdullah Hanif
Registration Number 11946414

to the Faculty of Informatics

at the TU Wien

Advisor: Prof. Dr. Muhammad Shafique, PhD

The dissertation has been reviewed by:

Assoc. Prof. Dr. Morteza Biglari-Abhari

Prof. Dr. Smail Niar

Vienna, 28th November, 2023

Digitally signed by Muhammad Abdullah Hanif
Date: 2023.11.28 19:10:22 +04'00'

Muhammad Abdullah Hanif

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Muhammad Abdullah Hanif

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 28. November 2023

Digitally signed by Muhammad Abdullah Hanif
Date: 2023.11.28 19:14:28 +04'00'

Muhammad Abdullah Hanif

Acknowledgements

First and foremost, I would like to present my cordial gratitude to my advisor Prof. Dr. Muhammad Shafique for his continuous guidance, support, scientific feedback, and encouragement during the course of my PhD. His guidance and endless support enabled me to increase the depth and breadth of my knowledge and shape my research and career path. Following his guidance, I have published papers in various top-ranking conferences and journals, and have also secured the ACM SIGDA Best PhD Forum Presentation Award in the 24^{th} IEEE/ACM Design Automation & Test in Europe (DATE) conference.

I would also like to thank my proficiency evaluation committee members, Prof. Dr. Theocharis Theocharides and Prof. Dr. Wolfgang Kastner, for their valuable technical feedback.

I would like to present my special thanks to Prof. Dr. Ulrich Schmid (head of Embedded Computing Systems (ECS) group), Prof. Dr. Andreas Steininger (director of the Vienna PhD School of Informatics) and Ms. Edeltraud Sommer for their endless support in all institute-related matters. I am also very grateful to Ms. Clarissa Schmid for her continuous help and support in all administrative steps related to my studies at TU Wien. I would also like to thank all my colleagues at the Computer Architecture and Robust Energy-Efficient Technologies (CARE-Tech.) and ECS groups for their encouragement and support.

I dedicate very special thanks to Florian Kriebel for all the guidance and support and for always being a huge source of motivation throughout my stay at TU Wien.

I am also extremely grateful to Prof. Dr. Rehan Hafiz and all my previous teachers, mentors and institutes who significantly contributed to my knowledge and showed me the path of research.

Lastly, I would like to express my gratitude and deepest love to my parents and siblings, who are always there sharing my pains and joys.

Abstract

Due to their unprecedented capability to discover patterns in data, Deep Learning (DL) algorithms have emerged as a powerful and dominant set of techniques for solving challenging problems that fall under the domain of Artificial Intelligence (AI). The models trained using these algorithms, i.e., Deep Neural Networks (DNNs), are nowadays being used in almost every industry for various applications, including safety-critical applications, e.g., autonomous driving, healthcare, and security & surveillance. For safety-critical applications, reliability against hardware-induced faults (e.g., soft errors, device aging, and manufacturing defects) is one of the foremost concerns, as faults at critical locations in a system can significantly degrade its application-level accuracy. The high overheads of conventional redundancy-based fault-mitigation techniques (e.g., dual-/triple-modular redundancy, instruction duplication, and error-correcting codes) coupled with the compute-intensive nature of DNNs limit their applicability for DNN-based applications, especially embedded applications. Therefore, alternative approaches are required that can exploit the intrinsic characteristics of these networks to offer improved resilience at low overhead (in terms of energy, area, and performance). Moreover, the intrinsic resilience characteristics of DNNs can also be exploited for introducing carefully-crafted designer-induced approximations to further improve the energy efficiency of the systems and compensate for the overheads of fault-mitigation techniques.

To enable highly robust and energy-efficient DL systems, this PhD work aims at exploiting the unique error-resilience characteristics of DNNs to mitigate the effects of hardware-induced reliability threats at low cost and to further improve the energy efficiency of DNN-based systems through judicious approximations (i.e., carefully-crafted designer-induced approximations in less-sensitive computations/neurons for trading quality for efficiency). To achieve this, this work explores opportunities at both the software and hardware levels. In particular, this work develops novel concepts for substantially reducing the frequency of critical faults by either modifying the system to have a biased fault distribution (biased towards non-critical faults) or by transforming critical faults into the ones that can be tolerated by the system due to the intrinsic resilience of DNNs. Moreover, this work also develops concepts for effectively exploiting the intrinsic resilience of DNNs to improve the energy efficiency by relaxing the accuracy bounds of intermediate computations through designer-induced approximations. In fact, this work shows that when prudently designed, approximations may be deployed without having any application-level accuracy loss, which is crucial for safety-critical systems

where energy efficiency and reliability both are important design metrics. Key highlights of the novel contributions of this PhD thesis are:

Low-cost fault and aging mitigation for DL systems: This thesis develops the following concepts and techniques for mitigating the effects of hardware-induced reliability threats at low cost by leveraging the intrinsic error-resilience characteristics of DNNs.

1. *Saliency-driven fault-aware mapping* is proposed to enable reliable execution of DNNs on the hardware accelerators with permanent faults without employing fault-aware retraining. The technique defines mapping of different parts of the given DNN on the hardware accelerator (subjected to faults) by leveraging the saliency of DNN parameters and the faults in the hardware accelerator.

2. *A framework for mitigating aging in the on-chip weight memory of DNN accelerators* is proposed. The framework jointly exploits hardware- and software-level knowledge to improve the lifetime of on-chip weight memories with negligible energy overhead. At the software-level, the effects of using different DNN quantization methods are analyzed. Based on the insights gained from the analysis, a micro-architecture is proposed that employs low-cost memory-write (and read) transducers to achieve an optimal duty-cycle in the weight memory cells to minimize aging.

Approximations for Energy-Efficient DNN Implementations: To enable highly energy-efficient DNN implementation by leveraging the intrinsic error-resilience of DNNs, the following concepts and techniques are developed in this work.

1. *Statistical techniques for error estimation of approximate adders* are proposed that enable fast yet accurate estimation of accuracy characteristics of different types and configurations of approximate adders. These techniques enable efficient design space exploration and selection of approximate modules. The insights gained from using these techniques for analyzing cascaded approximate modules help in defining effective application-specific approximations for DNNs.

2. *A cross-layer approximation methodology* is proposed that systematically combines the hardware-level and the software-level techniques together to achieve high energy efficiency. At the software level, a structured pruning technique is employed along with the quantization of inputs and network parameters to reduce the computational complexity and memory requirements of DNNs. At the hardware level, functional approximations are employed in the arithmetic modules to further improve the efficiency of DNN implementations.

3. *The concept of curable approximations for DNN hardware accelerators* is proposed to benefit from approximations without compromising application-level accuracy. The concept enables the design of high-performance DNN accelerators where approximation error(s) from one stage/part of the design is "completely" compensated in the subsequent stage/part while offering significant efficiency gains.

4. *Non-uniform post-training quantization* is proposed for energy-efficient and low-precision implementation of deep convolutional neural networks by leveraging the key insight that DNN weights and activations are mostly concentrated near zero and only a limited number of them have large magnitudes. The proposed method also exploits correlation between activation values for partial compensation of quantization errors enabling ultra energy-efficient DNN implementation.

In summary, this thesis presents several techniques for improving the robustness of DNN inference systems against hardware-induced reliability threats at low overhead cost by exploiting the unique error-resilience characteristics of DNNs. The thesis also presents techniques for improving the energy efficiency of DNN inference through carefully-crafted approximations that have minimal impact on the DNN accuracy while offering significant efficiency gains. The techniques are not restricted to a single abstraction layer. Specifically, for improving the energy efficiency, a cross-layer methodology is proposed that systematically combines software-level and hardware-level techniques to achieve higher benefits.

Kurzfassung

[Translation of the English version]

Aufgrund ihrer beispiellosen Fähigkeit, Muster in Daten zu entdecken, haben sich Deep Learning (DL)-Algorithmen als leistungsfähige und dominierende Techniken zur Lösung anspruchsvoller Probleme im Bereich der Künstlichen Intelligenz (KI) entwickelt. Die mit diesen Algorithmen trainierten Modelle, d.h. Deep Neural Networks (DNNs), werden heute in fast jeder Branche für verschiedene Anwendungen eingesetzt, darunter auch sicherheitskritische Anwendungen, z.B. autonomes Fahren, Gesundheit sowie Sicherheit und Überwachung. In sicherheitskritischen Anwendungen ist die Zuverlässigkeit gegenüber hardwarebedingten Fehlern (z.B. Soft Errors, Gerätealterung und Fertigungsfehler) eines der wichtigsten Anliegen, da Fehler an kritischen Stellen in einem System die Genauigkeit auf Anwendungsebene erheblich beeinträchtigen können. Der hohe Overhead konventioneller redundanzbasierter Techniken zur Fehlervermeidung (z.B. dual/triple modular redundancy, Befehlsverdopplung und fehlerkorrigierende Codes) in Verbindung mit der rechenintensiven Natur von DNNs schränkt ihre Anwendbarkeit für DNN-basierte Anwendungen, insbesondere für eingebettete Anwendungen, ein. Daher werden alternative Ansätze benötigt, die die intrinsischen Eigenschaften dieser Netzwerke ausnutzen, um eine verbesserte Resilienz bei geringerem Overhead (in Bezug auf Energie, Fläche und Leistung) zu bieten. Darüber hinaus können die intrinsischen Resilienzeigenschaften von DNNs auch genutzt werden, um sorgfältig entworfene, designerinduzierte Approximationen einzuführen, um die Energieeffizienz von Systemen weiter zu verbessern und den Overhead von Fehlerbehebungstechniken zu kompensieren.

Um hochgradig robuste und energieeffiziente DL-Systeme zu ermöglichen, zielt diese Dissertation darauf ab, die einzigartige Fehlerresilienz von DNNs zu nutzen, um die Auswirkungen von hardwarebedingten Zuverlässigkeitsproblemen kostengünstig zu mildern und die Energieeffizienz von DNN-basierten Systemen durch sinnvolle Approximationen weiter zu verbessern (d.h. durch sorgfältig entworfene designerinduzierte Approximationen in weniger empfindlichen Berechnungen/Neuronen, um Qualität gegen Effizienz zu tauschen). Um dies zu erreichen, werden in dieser Arbeit Möglichkeiten sowohl auf Software- als auch auf Hardware-Ebene untersucht. Insbesondere werden neuartige Konzepte entwickelt, um die Häufigkeit kritischer Fehler signifikant zu reduzieren, indem entweder das System so modifiziert wird, dass es eine einseitige Fehlerverteilung aufweist (einseitig auf unkritische Fehler ausgerichtet), oder indem kritische Fehler in solche umgewandelt werden, die

aufgrund der intrinsischen Resilienz von DNNs vom System toleriert werden können. Darüber hinaus werden in dieser Arbeit Konzepte entwickelt, um die intrinsische Resilienz von DNNs effektiv zu nutzen, um die Energieeffizienz zu verbessern, indem die Genauigkeitsgrenzen von Zwischenberechnungen durch konstruktionsbedingte Approximationen gelockert werden. Diese Arbeit zeigt, dass Approximationen ohne Genauigkeitsverlust auf Anwendungsebene eingesetzt werden können, was für sicherheitskritische Systeme entscheidend ist, bei denen sowohl Energieeffizienz als auch Zuverlässigkeit wichtige Designkriterien sind. Die wichtigsten Highlights der neuen Beiträge dieser Dissertation sind:

Kostengünstige Fehler- und Alterungsminderung für DL-Systeme: In dieser Arbeit werden die folgenden Konzepte und Techniken entwickelt, um die Auswirkungen von hardwarebedingten Zuverlässigkeitsrisiken durch die Nutzung der intrinsischen Fehlerresilienz von DNNs kosteneffizient zu mindern.

1. Es wird ein fehlerbewusstes Mapping vorgeschlagen, um die zuverlässige Ausführung von DNNs auf Hardwarebeschleunigern mit permanenten Fehlern ohne fehlerbewusstes Retraining zu ermöglichen. Die Technik definiert das Mapping verschiedener Teile des gegebenen DNNs auf den (fehlerbehafteten) Hardwarebeschleuniger unter Verwendung der Ausprägung der DNN-Parameter und der Fehler im Hardwarebeschleuniger.

2. Es wird ein Framework vorgeschlagen, um die Alterung des On-Chip-Speichers von DNN-Beschleunigern zu reduzieren. Das Framework nutzt Wissen auf Hardware- und Software-Ebene, um die Lebensdauer von On-Chip Weight-Memory mit vernachlässigbarem Energieaufwand zu verbessern. Auf der Software-Ebene werden die Auswirkungen der Verwendung verschiedener DNN-Quantisierungsmethoden analysiert. Basierend auf den aus der Analyse gewonnenen Erkenntnissen wird eine Mikroarchitektur vorgeschlagen, die kostengünstige Speicher-Schreib- (und Lese-) Wandler verwendet, um einen optimalen Duty-Cycle in den Speicherzellen des Weight-Memory zu erreichen und damit die Alterung zu minimieren.

Ansätze für energieeffiziente DNN-Implementierungen: Um eine extrem energieeffiziente DNN-Implementierung zu ermöglichen, indem die intrinsische Fehlerresilienz von DNNs ausgenutzt wird, werden in dieser Arbeit die folgenden Konzepte und Techniken entwickelt.

1. Es werden statistische Techniken zur Fehlerschätzung von approximativen Addierern vorgeschlagen, die eine schnelle und dennoch genaue Schätzung der Genauigkeitseigenschaften verschiedener Typen und Konfigurationen von approximativen Addierern ermöglichen. Diese Techniken ermöglichen eine effiziente Untersuchung des Entwurfsraums und die Auswahl approximativer Module. Die Erkenntnisse, die

aus der Anwendung dieser Techniken zur Analyse kaskadierter Approximationsmodule gewonnen werden, helfen bei der Definition effektiver anwendungsspezifischer Approximationen für DNNs.

2. Es wird eine schichtenübergreifende Approximationsmethodik vorgeschlagen, die systematisch Techniken auf Hardware- und Software-Ebene kombiniert, um eine hohe Energieeffizienz zu erreichen. Auf der Software-Ebene wird eine strukturierte Pruning-Technik zusammen mit der Quantisierung von Eingaben und Netzwerkparametern eingesetzt, um die Rechenkomplexität und den Speicherbedarf von DNNs zu reduzieren. Auf Hardware-Ebene werden funktionale Approximationen in arithmetischen Modulen verwendet, um die Effizienz von DNN-Implementierungen weiter zu verbessern.

3. Das Konzept der heilbaren Approximationen für DNN-Hardwarebeschleuniger wird vorgeschlagen, um von Approximationen zu profitieren, ohne die Genauigkeit auf Anwendungsebene zu beeinträchtigen. Das Konzept ermöglicht die Entwicklung von leistungsfähigen DNN-Beschleunigern, bei denen Approximationsfehler aus einer Stufe/einem Teil des Entwurfs in der nächsten Stufe/dem nächsten Teil "vollständig"kompensiert werden, während gleichzeitig erhebliche Effizienzgewinne erzielt werden.

4. Eine ungleichmäßige Quantisierung nach dem Training wird für eine energieeffiziente Implementierung von Deep Convolutional Neural Networks mit niedriger Genauigkeit vorgeschlagen. Dabei wird die wichtige Erkenntnis ausgenutzt, dass die Gewichte und Aktivierungen von DNNs in der Regel nahe Null konzentriert sind und nur eine begrenzte Anzahl von ihnen große Beträge aufweist. Die vorgeschlagene Methode nutzt auch die Korrelation zwischen Aktivierungswerten, um Quantisierungsfehler teilweise zu kompensieren, und ermöglicht so eine sehr energieeffiziente DNN-Implementierung.

Zusammenfassend kann gesagt werden, dass in dieser Dissertation mehrere Techniken vorgestellt werden, um die Robustheit von DNN-Inferenzsystemen gegenüber hardwarebedingten Zuverlässigkeitsproblemen bei geringen Overhead-Kosten zu verbessern, indem die einzigartige Fehlerresilienz von DNNs genutzt wird. Darüber hinaus werden Techniken zur Verbesserung der Energieeffizienz von DNN-Inferenzsystemen durch sorgfältig entwickelte Approximationen vorgestellt, die nur minimale Auswirkungen auf die Genauigkeit von DNNs haben und gleichzeitig signifikante Effizienzgewinne bieten. Diese Techniken sind nicht auf eine einzelne Abstraktionsebene beschränkt. Insbesondere wird zur Verbesserung der Energieeffizienz eine schichtenübergreifende Methodik vorgeschlagen, die systematisch Techniken auf Software- und Hardware-Ebene kombiniert, um höhere Vorteile zu erzielen.

Publications of this PhD Thesis

Conference Publications:

[1] Muhammad Abdullah Hanif, Rehan Hafiz, Osman Hasan, and Muhammad Shafique. QuAd: Design and analysis of quality-area optimal low-latency approximate adders. In *Design Automation Conference (**DAC**)*, pages 1–6, 2017. [**received a HIPEAC Paper Award**].

[2] Muhammad Abdullah Hanif, Faiq Khalid, and Muhammad Shafique. CANN: Curable approximations for high-performance deep neural network accelerators. In *Design Automation Conference (**DAC**)*, pages 1–6. IEEE, 2019. [**received a HIPEAC Paper Award**].

[3] Muhammad Abdullah Hanif, Rehan Hafiz, Osman Hasan, and Muhammad Shafique. PEMACx: A probabilistic error analysis methodology for adders with cascaded approximate units. In *Design Automation Conference (**DAC**)*, pages 1–6, 2020. [**received a HIPEAC Paper Award**].

[4] Muhammad Abdullah Hanif, Rehan Hafiz, and Muhammad Shafique. Error resilience analysis for systematically employing approximate computing in convolutional neural networks. In *Design, Automation & Test in Europe Conference & Exhibition (**DATE**)*, pages 913–916. IEEE, 2018.

[5] 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 *2021 Design, Automation & Test in Europe Conference & Exhibition (**DATE**)*, pages 729–734. IEEE, 2021. [**Best Paper Award Candidate**].

[6] Muhammad Abdullah Hanif, Giuseppe Maria Sarda, Alberto Marchisio, Guido Masera, Maurizio Martina, and Muhammad Shafique. CoNLoCNN: Exploiting correlation and non-uniform quantization for energy-efficient low-precision deep convolutional neural networks. In *2022 International Joint Conference on Neural Networks (**IJCNN**)*, pages 1–8. IEEE, 2022.

[7] Muhammad Abdullah Hanif, Faiq Khalid, Rachmad Vidya Wicaksana Putra, Semeen Rehman, and Muhammad Shafique. Robust machine learning systems: Reliability and security for deep neural networks. In *International Symposium on On-Line Testing And Robust System Design (**IOLTS**)*, pages 257–260. IEEE, 2018.

[8] Muhammad Abdullah Hanif and Muhammad Shafique. Dependable deep learning: Towards cost-efficient resilience of deep neural network accelerators against soft errors and permanent faults. In *International Symposium on On-Line Testing And Robust System Design (**IOLTS**)*, pages 1–4. IEEE, 2020.

[9] Muhammad Abdullah Hanif, Le Ha Hoang, and Muhammad Shafique. Cross-layer approaches for improving the dependability of deep learning systems. In *International Workshop on Software and Compilers for Embedded Systems (**SCOPES**)*, pages 78–81. ACM, 2020.

Journal Publications:

[10] Muhammad Abdullah Hanif and Muhammad Shafique. SalvageDNN: Salvaging deep neural network accelerators with permanent faults through saliency-driven fault-aware mapping. *Philosophical Transactions of the Royal Society A (**RSTA**)*, 378(2164):20190164, 2020.

[11] Muhammad Abdullah Hanif, Alberto Marchisio, Tabasher Arif, Rehan Hafiz, Semeen Rehman, and Muhammad Shafique. X-DNNs: Systematic cross-layer approximations for energy-efficient deep neural networks. *Journal of Low Power Electronics (**JOLPE**)*, 14(4):520–534, 2018.

[12] Muhammad Abdullah Hanif and Muhammad Shafique. A cross-layer approach towards developing efficient embedded deep learning systems. *Microprocessors and Microsystems (**MICPRO**)*, 88:103609, 2022.

Other Co-Authored Publications

Conference Publications:

[1] Vojtech Mrazek, Muhammad Abdullah Hanif, Zdenek Vasícek, Lukás Sekanina, and Muhammad Shafique. autoAx: An automatic design space exploration and circuit building methodology utilizing libraries of approximate components. In *Design Automation Conference (**DAC**)*, page 123. ACM, 2019. [**received a HIPEAC Paper Award**].

[2] Jeff Jun Zhang, Kang Liu, Faiq Khalid, Muhammad Abdullah Hanif, Semeen Rehman, Theocharis Theocharides, Alessandro Artussi, Muhammad Shafique, and Siddharth Garg. Building robust machine learning systems: Current progress, research challenges, and opportunities. In *Design Automation Conference (**DAC**)*, page 175. ACM, 2019. [**received a HIPEAC Paper Award**].

[3] Salim Ullah, Semeen Rehman, Bharath Srinivas Prabakaran, Florian Kriebel, Muhammad Abdullah Hanif, Muhammad Shafique, and Akash Kumar. Area-optimized low-latency approximate multipliers for fpga-based hardware accelerators. In *Design Automation Conference (**DAC**)*, pages 159:1–159:6. ACM, 2018. [**received a HIPEAC Paper Award**].

[4] Rachmad Vidya Wicaksana Putra, Muhammad Abdullah Hanif, and Muhammad Shafique. Sparkxd: A framework for resilient and energy-efficient spiking neural network inference using approximate dram. In *2021 58th ACM/IEEE Design Automation Conference (**DAC**)*, pages 379–384. IEEE, 2021. [**received a HIPEAC Paper Award**].

[5] Rachmad Vidya Wicaksana Putra, Muhammad Abdullah Hanif, and Muhammad Shafique. Softsnn: Low-cost fault tolerance for spiking neural network accelerators under soft errors. In *Proceedings of the 59th ACM/IEEE Design Automation Conference (**DAC**)*, pages 151–156, 2022. [**received a HIPEAC Paper Award**].

[6] Rachmad Vidya Wicaksana Putra, Muhammad Abdullah Hanif, and Muhammad Shafique. Drmap: A generic dram data mapping policy for energy-efficient processing of convolutional neural networks. In *2020 57th ACM/IEEE Design Automation Conference (**DAC**)*, pages 1–6. IEEE, 2020. [**received a HIPEAC Paper Award**].

[7] Vojtech Mrazek, Zdenek Vasícek, Lukás Sekanina, Muhammad Abdullah Hanif, and Muhammad Shafique. ALWANN: automatic layer-wise approximation of deep neural network accelerators without retraining. In David Z. Pan, editor, *International Conference on Computer-Aided Design (**ICCAD**)*, pages 1–8. ACM, 2019.

[8] Rachmad Vidya Wicaksana Putra, Muhammad Abdullah Hanif, and Muhammad Shafique. Respawn: Energy-efficient fault-tolerance for spiking neural networks considering unreliable memories. In *2021 IEEE/ACM International Conference On Computer Aided Design (**ICCAD**)*, pages 1–9. IEEE, 2021.

[9] Muhammad Shafique, Alberto Marchisio, Rachmad Vidya Wicaksana Putra, and Muhammad Abdullah Hanif. Towards energy-efficient and secure edge ai: A cross-layer framework iccad special session paper. In *2021 IEEE/ACM International Conference On Computer Aided Design (**ICCAD**)*, pages 1–9. IEEE, 2021.

[10] Bharath Srinivas Prabakaran, Semeen Rehman, Muhammad Abdullah Hanif, Salim Ullah, Ghazal Mazaheri, Akash Kumar, and Muhammad Shafique. DeMAS: An efficient design methodology for building approximate adders for fpga-based systems. In *Design, Automation & Test in Europe Conference & Exhibition (**DATE**)*, pages 917–920. IEEE, 2018.

[11] Muhammad Shafique, Theocharis Theocharides, Christos-Savvas Bouganis, Muhammad Abdullah Hanif, Faiq Khalid, Rehan Hafiz, and Semeen Rehman. An overview of next-generation architectures for machine learning: Roadmap, opportunities and challenges in the iot era. In *Design, Automation & Test in Europe Conference & Exhibition (**DATE**)*, pages 827–832. IEEE, 2018.

[12] Mohammad Taghi Teimoori, Muhammad Abdullah Hanif, Alireza Ejlali, and Muhammad Shafique. AdAM: Adaptive approximation management for the non-volatile memory hierarchies. In *Design, Automation & Test in Europe Conference & Exhibition (**DATE**)*, pages 785–790. IEEE, 2018.

[13] Alberto Marchisio, Muhammad Abdullah Hanif, and Muhammad Shafique. CapsAcc: An efficient hardware accelerator for capsulenets with data reuse. In *Design, Automation & Test in Europe Conference & Exhibition (**DATE**)*, pages 964–967. IEEE, 2019.

[14] 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. In *Design, Automation & Test in Europe Conference & Exhibition (**DATE**)*, 2020.

[15] Muhammad Abdullah Hanif, Muhammad Zuhaib Akbar, Rehan Ahmed, Semeen Rehman, Axel Jantsch, and Muhammad Shafique. MemGANs: Memory management for energy-efficient acceleration of complex computations in hardware architectures for generative adversarial networks. In *International Symposium on Low Power Electronics and Design (**ISLPED**)*, pages 1–6. IEEE, 2019.

[16] Alberto Marchisio, Muhammad Abdullah Hanif, Faiq Khalid, George Plastiras, Christos Kyrkou, Theocharis Theocharides, and Muhammad Shafique. Deep learning for edge computing: Current trends, cross-layer optimizations, and open research challenges. In *International Symposium on VLSI (**ISVLSI**)*, pages 553–559. IEEE, 2019.

[17] Alberto Marchisio, Muhammad Abdullah Hanif, Maurizio Martina, and Muhammad Shafique. PruNet: Class-blind pruning method for deep neural networks. In *International Joint Conference on Neural Networks (**IJCNN**)*, pages 1–8. IEEE, 2018.

[18] Shail Dave, Alberto Marchisio, Muhammad Abdullah Hanif, Amira Guesmi, Aviral Shrivastava, Ihsen Alouani, and Muhammad Shafique. Special session: Towards an agile design methodology for efficient, reliable, and secure ml systems. In *2022 IEEE 40th VLSI Test Symposium (**VTS**)*, pages 1–14. IEEE, 2022.

Journal Publications:

[19] Ghayoor A Gillani, Muhammad Abdullah Hanif, M Krone, Sabih H Gerez, Muhammad Shafique, and André BJ Kokkeler. SquASH: Approximate square-accumulate with self-healing. ***IEEE Access***, 6:49112–49128, 2018.

[20] Ghayoor A Gillani, Muhammad Abdullah Hanif, Bart Verstoep, Sabih H Gerez, Muhammad Shafique, and Andre BJ Kokkeler. MACISH: Designing approximate mac accelerators with internal-self-healing. ***IEEE Access***, 7:77142–77160, 2019.

[21] Muhammad Abdullah Hanif, Aditya Manglik, and Muhammad Shafique. Resistive crossbar-aware neural network design and optimization. ***IEEE Access***, 8:229066–229085, 2020.

[22] Paniti Achararit, Muhammad Abdullah Hanif, Rachmad Vidya Wicaksana Putra, Muhammad Shafique, and Yuko Hara-Azumi. Apnas: Accuracy-and-performance-aware neural architecture search for neural hardware accelerators. ***IEEE Access***, 8:165319–165334, 2020.

[23] Nivedita Shrivastava, Muhammad Abdullah Hanif, Sparsh Mittal, Smruti Ranjan Sarangi, and Muhammad Shafique. A survey of hardware architectures for generative adversarial networks. *Journal of Systems Architecture (**JSA**)*, 118:102227, 2021.

[24] Muhammad Kamran Ayub, Muhammad Abdullah Hanif, Osman Hasan, and Muhammad Shafique. Peal: Probabilistic error analysis methodology for low-power approximate adders. *ACM Journal on Emerging Technologies in Computing Systems (**JETC**)*, 17(1):1–37, 2020.

[25] Hazoor Ahmad, Tabasher Arif, Muhammad Abdullah Hanif, Rehan Hafiz, and Muhammad Shafique. Superslash: A unified design space exploration and model compression methodology for design of deep learning accelerators with reduced

off-chip memory access volume. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (**TCAD**)*, 39(11):4191–4204, 2020.

[26] Alberto Marchisio, Vojtech Mrazek, Muhammad Abdullah Hanif, and Muhammad Shafique. Descnet: Developing efficient scratchpad memories for capsule network hardware. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (**TCAD**)*, 40(9):1768–1781, 2020.

[27] Alberto Marchisio, Vojtech Mrazek, Muhammad Abdullah Hanif, and Muhammad Shafique. Feeca: design space exploration for low-latency and energy-efficient capsule network accelerators. *IEEE Transactions on Very Large Scale Integration (**TVLSI**) Systems*, 29(4):716–729, 2021.

[28] Rachmad Vidya Wicaksana Putra, Muhammad Abdullah Hanif, and Muhammad Shafique. Romanet: Fine-grained reuse-driven off-chip memory access management and data organization for deep neural network accelerators. *IEEE Transactions on Very Large Scale Integration (**TVLSI**) Systems*, 29(4):702–715, 2021.

[29] Rachmad Vidya Wicaksana Putra, Muhammad Abdullah Hanif, and Muhammad Shafique. Rescuesnn: enabling reliable executions on spiking neural network accelerators under permanent faults. *Frontiers in Neuroscience*, 17, 2023.

[30] Rachmad Vidya Wicaksana Putra, Muhammad Abdullah Hanif, and Muhammad Shafique. Enforcesnn: Enabling resilient and energy-efficient spiking neural network inference considering approximate drams for embedded systems. *Frontiers in Neuroscience*, 16, 2022.

Book Chapters:

[31] Muhammad Abdullah Hanif, Rehan Hafiz, and Muhammad Shafique. Configurable models and design space exploration for low-latency approximate adders. *Approximate Circuits: Methodologies and CAD*, pages 3–23, 2019.

[32] Muhammad Abdullah Hanif, Muhammad Usama Javed, Rehan Hafiz, Semeen Rehman, and Muhammad Shafique. Hardware–software approximations for deep neural networks. *Approximate Circuits: Methodologies and CAD*, pages 269–288, 2019.

[33] Muhammad Shafique, Osman Hasan, Rehan Hafiz, Sana Mazahir, Muhammad Abdullah Hanif, and Semeen Rehman. Approximate computing across the hardware and software stacks. In *Many-Core Computing*, pages 497–522. Institution of Engineering and Technology, 2019.

[34] Muhammad Abdullah Hanif, Vojtech Mrazek, and Muhammad Shafique. Approximate computing architectures. In *Handbook of Computer Architecture*, pages 1–41. Springer, 2022.

Awards and Achievements

1. **PhD Forum Best Poster Award** supported by EDAA, ACM SIGDA, and IEEE CEDA in the 24th Design, Automation & Test in Europe (DATE) Conference.

2. Our DNN-Life paper got shortlisted as a Best Paper Award Candidate in the 24th Design, Automation & Test in Europe (DATE) Conference.

   - 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 2021 *Design, Automation & Test in Europe Conference & Exhibition (**DATE**)*, pages 729–734. IEEE, 2021. [Best Paper Award Candidate]

3. Several **HIPEAC Paper Awards**.

   - Muhammad Abdullah Hanif, Rehan Hafiz, Osman Hasan, and Muhammad Shafique. QuAd: Design and analysis of quality-area optimal low-latency approximate adders. In *Design Automation Conference (**DAC**)*, pages 1–6, 2017. [received a HIPEAC Paper Award]
   - Muhammad Abdullah Hanif, Faiq Khalid, and Muhammad Shafique. CANN: Curable approximations for high-performance deep neural network accelerators. In *Design Automation Conference (**DAC**)*, pages 1–6. IEEE, 2019. [received a HIPEAC Paper Award]
   - Muhammad Abdullah Hanif, Rehan Hafiz, Osman Hasan, and Muhammad Shafique. PEMACx: A probabilistic error analysis methodology for adders with cascaded approximate units. In *Design Automation Conference (**DAC**)*, pages 1–6, 2020. [received a HIPEAC Paper Award]
   - Vojtech Mrazek, Muhammad Abdullah Hanif, Zdenek Vasícek, Lukás Sekanina, and Muhammad Shafique. autoAx: An automatic design space exploration and circuit building methodology utilizing libraries of approximate components. In *Design Automation Conference (**DAC**)*, page 123. ACM, 2019. [received a HIPEAC Paper Award]
   - Jeff Jun Zhang, Kang Liu, Faiq Khalid, Muhammad Abdullah Hanif, Semeen Rehman, Theocharis Theocharides, Alessandro Artussi, Muhammad Shafique,

and Siddharth Garg. Building robust machine learning systems: Current progress, research challenges, and opportunities. In *Design Automation Conference (**DAC**)*, page 175. ACM, 2019. [received a HIPEAC Paper Award]

- Salim Ullah, Semeen Rehman, Bharath Srinivas Prabakaran, Florian Kriebel, Muhammad Abdullah Hanif, Muhammad Shafique, and Akash Kumar. Area-optimized low-latency approximate multipliers for fpga-based hardware accelerators. In *Design Automation Conference (**DAC**)*, pages 159:1–159:6. ACM, 2018. [received a HIPEAC Paper Award]
- Rachmad Vidya Wicaksana Putra, Muhammad Abdullah Hanif, and Muhammad Shafique. Sparkxd: A framework for resilient and energy-efficient spiking neural network inference using approximate dram. In *2021 58th ACM/IEEE Design Automation Conference (**DAC**)*, pages 379–384. IEEE, 2021. [**received a HIPEAC Paper Award**]
- Rachmad Vidya Wicaksana Putra, Muhammad Abdullah Hanif, and Muhammad Shafique. Softsnn: Low-cost fault tolerance for spiking neural network accelerators under soft errors. In *Proceedings of the 59th ACM/IEEE Design Automation Conference (**DAC**)*, pages 151–156, 2022. [**received a HIPEAC Paper Award**]
- Rachmad Vidya Wicaksana Putra, Muhammad Abdullah Hanif, and Muhammad Shafique. Drmap: A generic dram data mapping policy for energy-efficient processing of convolutional neural networks. In *2020 57th ACM/IEEE Design Automation Conference (**DAC**)*, pages 1–6. IEEE, 2020. [**received a HIPEAC Paper Award**]

Contents

CHAPTER 1

Introduction

Over the past few years, the unprecedented capability to learn hierarchical representations directly from raw data has enabled Deep Neural Networks (DNNs) to become state of the art for many Artificial Intelligence (AI) applications [LBH15]. This evolution of DNNs has mainly been possible because of the availability of large datasets and the advancements in the parallel-computing hardware technology, which enabled the compute-intensive training of these models through backpropagation algorithms [LBH15][KSH12]. Given that DNNs have achieved (or surpassed) human-level accuracy in a wide range of applications [XLHL20][TYRW14], they are currently being used in a number of real-world systems and applications, including intelligent Internet of Things (IoT) and Cyber–Physical Systems (CPS) [MAFSG18]. Autonomous vehicles [AQBAQR17][HWT+15][MAJD+20], smart transportation systems [ZKKK19], smart healthcare systems [LKB+17][MWW+18] [ERR+19], intelligent wearable systems [RPP21], smart homes [MPC16], smart grids [ZAJ+19] and robotic systems [SBS+18] are a few examples of these real-world intelligent systems. As most of these systems are categorized as resource constrained and/or safety critical, the energy efficiency and dependability of the algorithms used at the backend arise as the foremost concerns.

1.1 Motivation

Besides intelligent IoT and CPS, the proliferation of powerful computing devices in the form of smart phones, personal computers and embedded GPU platforms has also given rise to various applications that can benefit from deep learning (see Figs. 1.1 and 1.2). Virtual assistants, content recommendation, search optimization, music composition, adding sounds to silent movies, automatic machine translation, face recognition, data restoration, route planning, visual art creation, automatic game playing and supersampling are just a few examples of such applications [LBH15][Sar21][ME19]. Although state-of-the-art DNNs have demonstrated remarkable performance in terms of accuracy

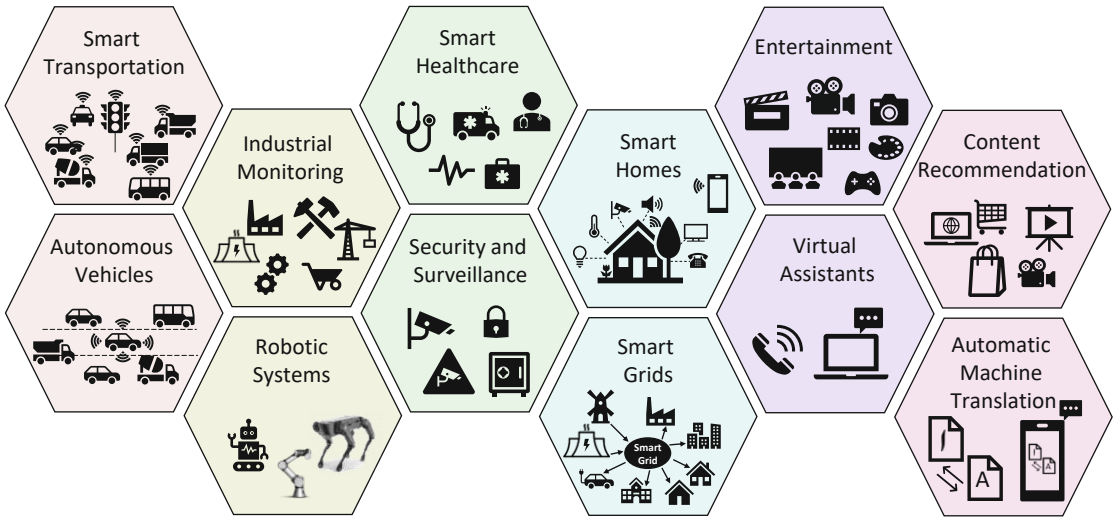


Figure 1.1: Some prominent applications of deep learning.

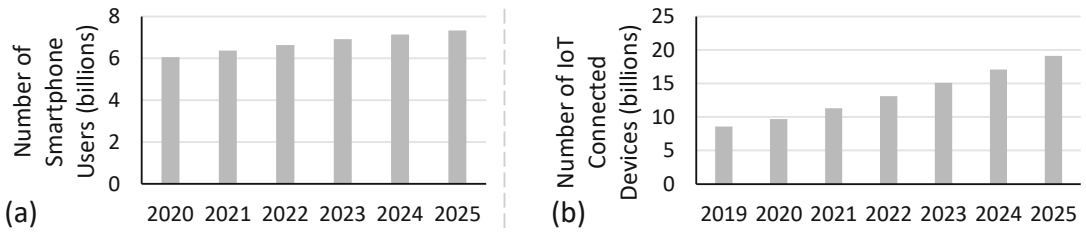


Figure 1.2: Current estimate and forecast of (a) number of smartphone users in the world and (b) number of connected IoT devices. (data source: [Vai][ban])

for all these applications, a key challenge associated with them is that the high-accuracy models are typically highly resource hungry. This is highlighted in Fig. 1.3 with the help of computational requirements, size and accuracy of different DNNs trained on the ImageNet dataset [DDS+09] for image classification. As shown in the figure, the general trend is that the models that offer high accuracy require a higher number of computations per inference. For example, the AlexNet model [KSH12] has around 60 million parameters, requires around 725 MFLOPs to process one input image and offers around 63.3% accuracy on the ImageNet validation set, while the VGG16 model [SZ14] has around 138 million parameters, requires around 15.5 GFLOPs to process one input image and offers around 74% accuracy. Both the AlexNet and the VGG16 models are based on simple convolutional and linear layers. Advancements in network architectures, i.e., introduction of residual blocks, inverted residual blocks, shortcut connections, depthwise separable convolutions and pointwise convolutions, have lead to improved accuracy results with lesser number of computations and lower memory requirements. However, the general trend is the same, as can be observed by comparing the characteristics of ResNet18 and ResNet152 models.

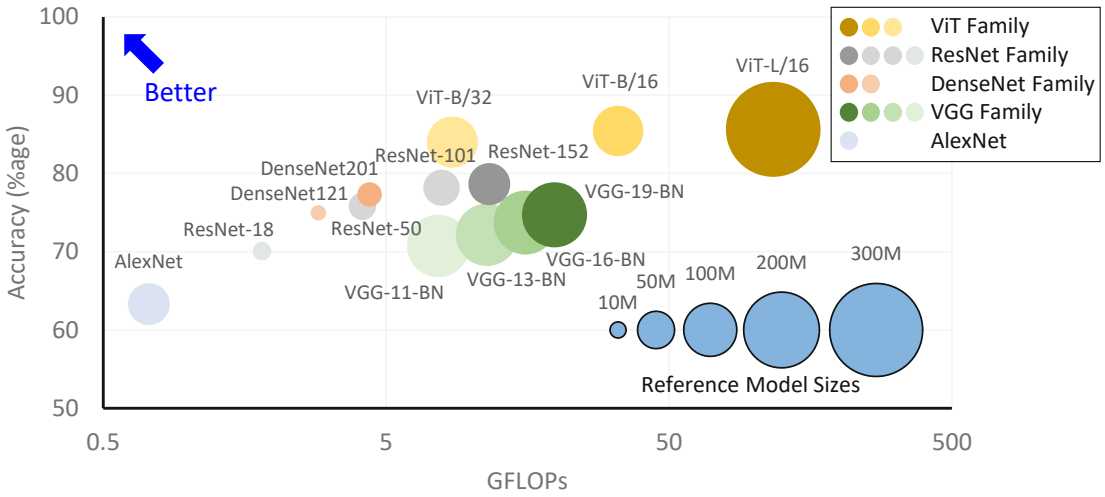


Figure 1.3: Characteristics of DNNs proposed for image classification on the ImageNet dataset. (data source: [mmc][pap])

Recently, more complex architectures have been proposed, e.g., Vision Transformers (ViT) [DBK+20], highlighting the same trend that more complex and deeper neural networks lead to better accuracy results. *The complexity (resource requirements) of high-accuracy DNNs clearly highlights the need for techniques that can improve the efficiency of DNNs without compromising the accuracy to enable their deployment in resource-constrained scenarios.*

Besides computational complexity, highlighting the dependability of DNNs is also important, as these models are nowadays commonly being used in safety-critical systems. In the past, several instances have been reported where a single bit-flip error in the system could cause catastrophic accidents. For example, investigations related to the reported cases of sudden unintended acceleration of cars manufactured by Toyota revealed that a single bit-flip error in the electronic throttle control system could cause the driver to lose control of the engine speed [EET], which in the worst case leads to fatal accidents [Yos]. Moreover, thorough analysis of Boeing 737 MAX's flight-control system, which was performed after two 737 MAX crashed, also uncovered similar flaws in the system [Leo]. These examples clearly highlight the importance of dependability of the designed systems alongside other factors (e.g., energy/power efficiency). Recent works such as [Cea19][HKP+18][Cea20] clearly show that similar dependability concerns exist in DNN systems as well due to the imperfect nature of DNNs.

Driven by the compute-intensive nature of DNNs, specialized hardware accelerators (e.g., TPU [JYP+17], MAERI [KSK18] and Eyeriss [CYES19]) are employed to offer cost-effective and efficient DNN inference. However, these accelerators experience various reliability threats, such as soft errors, device aging and manufacturing defects (for details related to reliability threats see Section 2.5). These threats manifest as bit-flip errors

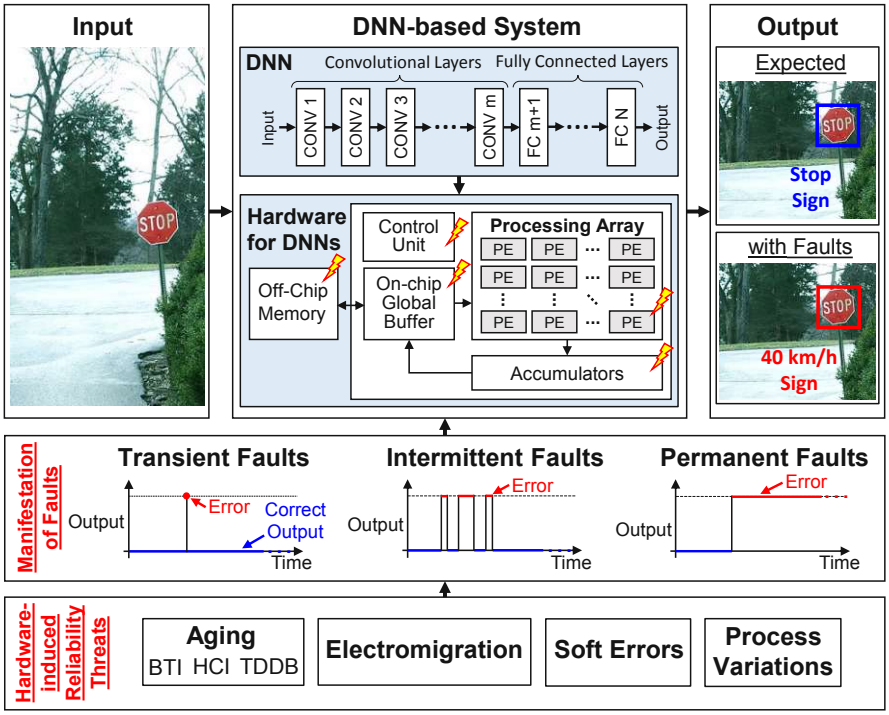


Figure 1.4: Reliability threats, their manifestation, and impact on a DNN-based system's output. (The used *stop sign* picture is from the COCO dataset [LMB+14])

at the hardware level and thereby can severely impact the functionality of the system as highlighted in the above examples; also see Fig. 1.4. *As these threats raise several safety concerns, it is crucial to ensure robustness against them.* Moreover, safety-critical systems are often required to meet industrial standards like ISO 26262 [ISO11] that allow no more than 10 failures in 10^9 hours of device operation, which further emphasizes the need for robustness against all types of reliability threats. Apart from addressing the errors that can occur at run-time, handling manufacturing defects through post-fabrication optimizations is also important, as these defects can significantly reduce the manufacturing yield, specifically for wafer-scale engines [McL19]. Therefore, *it is vital to ensure robustness of these systems against all types of technology-induced reliability threats alongside improving their energy efficiency.*

1.2 Research Challenges and Objectives

Given the compute-intensive nature of high-accuracy DNNs, the foremost challenge is how to enable their efficient deployment in resource-constrained systems. To address this challenge, several techniques have been proposed in the literature, which range from pruning and quantization at the software level all the way to functional approximations at the hardware level (see details in Chapter 2). All these techniques are based on the

fact that DNNs are (to some extent) inherently resilient to errors, and this resilience can be exploited to trade a fraction of quality for a significant amount of efficiency gain. However, DNNs respond differently to different types of errors. Moreover, the locations of errors in a DNN also have a significant impact on the accuracy. Therefore, it is crucial to classify these errors into critical and non-critical categories and identify the types that do not impact the application-level accuracy much so that they can be exploited to achieve high benefits (e.g., in terms of energy/power/area efficiency through approximations). This analysis can help in identifying the approximations that can be deployed in a system without compromising the application-level accuracy of the system, and it can also be helpful for designing fault-mitigation techniques that incur low overheads, particularly for application-specific scenarios.

As approximations can be applied at different abstraction levels and each approximation can have various configurations, tools and methods are required to efficiently and accurately estimate the error characteristics of different configurations to select the non-dominated configurations that offer the best quality-efficiency trade-offs for a given scenario. These tools/methods are expected to provide a pivotal component for enabling fast and automated design space exploration, specifically for approximate accelerators. Recently, different approximation techniques such as self-healing [GHK+18][GHV+19] and self-compensating [MHS19] approximations (including some of the works I co-authored) have also been proposed for introducing functional approximations in the computational units of hardware accelerators. However, the potential of these techniques has not been explored for DNN-based applications. Moreover, the state-of-the-art DNN pruning and quantization techniques have proven to be highly effective for improving the efficiency of DNN inference systems; however, the interaction of these optimization techniques with hardware-level approximations has not been studied so far.

In summary, related to approximations for improving the efficiency of DNN inference systems, this work focuses on the following key questions:

- How and to what extent can the resilience of DNNs be exploited to judiciously introduce approximations that can help in improving the power/energy, area and performance efficiency of a DNN inference system?
- How to systematically design approximate DNN accelerators that offer better quality-efficiency trade-off?
- How to integrate different DNN optimization techniques to achieve improved efficiency (in terms of energy/power, area and performance) under given constraints (e.g., required quality)?

Apart from energy/power efficiency, for safety-critical applications, reliability against hardware-induced faults (e.g., soft errors, device aging, and manufacturing defects) is also one of the foremost concerns, as faults at critical locations can result in a significant drop in the application-level accuracy. The high overheads of conventional

redundancy-based fault-mitigation techniques (e.g., dual-/triple-modular redundancy, instruction duplication, and error-correcting codes) coupled with the compute-intensive nature of DNNs limit the applicability of such techniques for DNN-based applications, especially embedded applications. Moreover, retraining-based mitigation techniques result in huge retraining costs, and they are also not applicable in some scenarios due to security and privacy reasons. *Therefore, alternative approaches are required that can exploit the intrinsic characteristics of DNNs to offer improved resilience at low overhead costs.* These characteristics mainly include: (1) the overall functionality of a DNN is distributed among its neurons and is not concentrated in certain neurons or parts of the network, (2) each neuron contains critical and non-critical weights, and the locations (indexes) of critical weights are not the same across neurons, and (3) each bit of a weight value has a different significance and criticality, which depends on the value of the weight as well as the number representation format used. Note that all these characteristics are, in general, different from conventional applications that are composed of different critical and non-critical modules. Towards developing low cost fault-mitigation techniques, a common perception of DNNs is that they are highly resilient to errors. However, studies have shown that DNNs are resilient to only certain types of errors (not all types) [HKP+18][ZGBG18][KHM+18b]. Thus, this resilience of DNNs can be exploited to design/modify DNN inference systems such that (in case of faults) their error distribution is biased towards non-critical errors (i.e., the ones that can be tolerated by the system) to offer high resilience against hardware-induced reliability threats.

In short, related to improving the resilience of DNNs against hardware-induced reliability threats at low overhead costs, this work focuses on the following key questions:

1. How and to what extent can the inherent resilience of DNNs be exploited to mitigate hardware-induced reliability threats?
2. How to offer efficient fault mitigation at low and moderate fault rates without involving fault-aware retraining?

In summary, this research focuses on designing cost-effective solutions for improving the robustness and energy/power efficiency of deep learning inference systems by exploiting the inherent error-resilience of DNNs at appropriate abstraction levels.

1.3 Thesis Contributions

To address the research challenges highlighted in Section 1.2, this thesis aims at exploiting the unique error-resilience characteristics of DNNs to improve the robustness of DNN inference systems against technology-induced reliability threats such as soft errors, device aging and manufacturing defects at low overhead costs. This research also intends to improve the energy efficiency of these systems through judicious approximations (i.e., carefully crafted designer-induced faults in less-sensitive neurons) that can be tolerated

due to error-resilience characteristics of DNNs. In fact, this research shows that when prudently designed, approximations may be deployed without having any accuracy loss, which is crucial for safety-critical systems where both energy efficiency and reliability are important design metrics.

An overview of the contributions of this PhD thesis and the related publications is illustrated in Fig. 1.5. The figure presents an integrated design flow for building robust and energy-efficient deep learning inference systems, with the proposed techniques highlighted in blue. To significantly benefit from the error resilience of DNNs for improving the energy efficiency and robustness of DNN inference systems, first, the resilience of DNNs to different types of errors/faults is analyzed. This resilience is then exploited to develop novel concepts and strategies to improve energy efficiency of the systems by relaxing the accuracy bounds of intermediate computations through conjoint software and hardware-level approximations (designer-induced errors/faults) that can offer high savings while having minimal or ideally no impact on the application-level accuracy of the system, i.e., without jeopardizing the safety requirements. The resilience is further exploited to improve the robustness of the system by substantially reducing the frequency of critical faults, which is achieved either by modifying the system to have a biased fault distribution (biased towards non-critical faults), or by transforming critical faults to non-critical faults. Towards this, the key novel contributions of this PhD thesis are:

Low-cost fault and aging mitigation for DNN inference systems: The following concepts and techniques are proposed as a part of this thesis for mitigating the effects of hardware-induced reliability threats at low cost by leveraging the intrinsic error-resilience characteristics of DNNs.

1. *Saliency-driven fault-aware mapping:* To enable reliable execution of DNNs on hardware accelerators having permanent faults without involving fault-aware re-training, the concept of saliency-driven fault-aware mapping is proposed in this work. The method is based on the principle of mapping non-critical (less significant) weights/parameters to the faulty locations.

2. *A framework for mitigating aging in the on-chip weight memory of DNN accelerators:* Aging in SRAM-based on-chip memory is a primary concern that impacts the lifetime of a fabricated device. To mitigate aging in the on-chip weight memory of DNN accelerators, the DNN-Life framework is proposed in this work. The framework jointly exploits software and hardware-level knowledge to improve the lifetime of SRAM-based on-chip weight memory with negligible energy and area overheads. At the software-level, the impact of different DNN quantization techniques on the aging of on-chip SRAM cells is analyzed. Based on the insights gained from the analysis, a design is proposed that employs low-cost memory-read and -write transducers to achieve the optimal duty-cycle in the weight memory cells to minimize aging.

Approximations for Energy-Efficient DNN Deployment: The following concepts

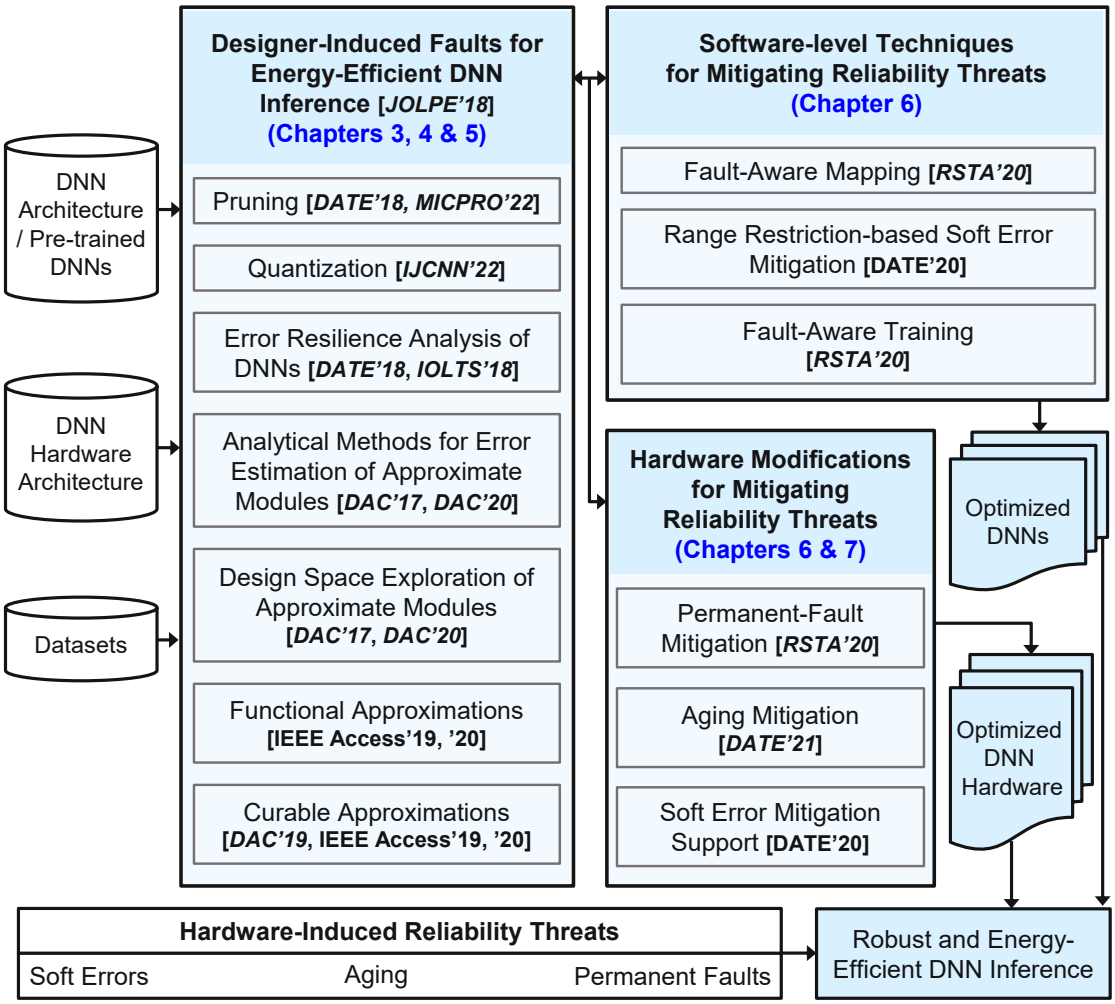


Figure 1.5: An overview of the integrated design flow for building robust and energy-efficient deep learning systems. The proposed techniques are highlighted in blue. The publications that are a part of this PhD thesis are mentioned in *italic* while other co-authored publications are mentioned in non-italic format.

and techniques are proposed as a part of this thesis to enable highly energy-efficient DNN inference by leveraging the error-resilience of DNNs through approximations.

1. *Statistical techniques for error estimation of approximate adders:* Fast-yet-accurate error estimation of different types and configurations of approximate modules is important to achieve efficient design space exploration of approximate modules. Towards this, in this thesis different techniques are proposed to compute accuracy characteristics of different types of approximate adders. The insights gained from these techniques when used for analyzing cascaded approximate modules and

approximate datapaths helped in defining application-specific approximations for DNN accelerators.

2. *A cross-layer approximation methodology:* To achieve energy-efficient DNN inference, a methodology is proposed in this thesis that systematically combines the software-level and the hardware-level approximation techniques. At the software level, a structured pruning technique is integrated with DNN quantization to reduce the computational complexity and memory requirements of DNNs. At the hardware level, functional approximation of arithmetic modules is considered to further improve the efficiency of DNN inference systems.

3. *The concept of curable approximations for high-performance accelerators:* To benefit from approximations without compromising application-level accuracy, the concept of curable approximations is proposed in this thesis. The concept enables the design of high-performance accelerators where approximation errors from one stage of the system are *completely* compensated in the subsequent stage while offering significant efficiency gains. This compensation of errors enables the system to maintain the baseline accuracy and avoid any undesirable application-level accuracy loss. This thesis also presents a case study for improving the performance of systolic array-based DNN accelerators using curable approximations.

4. *Non-uniform post-training quantization:* To achieve energy-efficient and low-precision implementation of deep convolutional neural networks, a non-uniform post-training quantization technique is proposed. The technique leverages the insight that DNN weights and activations are mostly concentrated around zero and only a limited number of values have a large magnitude. This enables the use of non-uniform quantization to simultaneously simplify the computational units (mainly Multiply-Accumulate (MAC) units) and achieve a low quantization error to offer better quality-efficiency tradeoffs than conventional quantization techniques. Alongside a novel quantization technique and the supporting hardware architecture, the proposed method also exploits correlation between activation values for partial compensation of quantization errors enabling ultra energy-efficient DNN implementation.

1.4 Thesis Outline

The thesis is organized in eight chapters, where Chapter 1 introduces the need for high performance deep learning models and the challenges associated with deploying such models in resource-constrained embedded devices for efficient and reliable inference. Chapter 2 presents the necessary background related to DNNs, hardware approximations, different reliability threats and state-of-the-art DNN optimization and fault-mitigation techniques. Chapters 3, 4, 5, 6 and 7 present concepts, techniques and evaluations of the novel contributions of this thesis. Towards the end, Chapter 8 presents a summary of the

novel contributions and findings of this thesis and also highlights some of the potential future research directions. A brief outline of each chapter is given below.

Chapter 2 - Background and Related Work: This chapter presents the background knowledge related to deep learning, prominent types of DNNs that are commonly used in the embedded systems community for benchmarking, hardware-induced reliability threats and techniques for improving the energy efficiency and robustness of DNN inference systems. In particular, Section 2.1 presents the basics of deep learning, multi-layer perceptrons and convolutional neural networks. Section 2.2 presents an overview of state-of-the-art DNN hardware accelerators. Then, Section 2.3 highlights the key design issues towards enabling efficient DNN inference. Afterwards, Section 2.4 covers the prominent software-level and hardware-level techniques used for improving the energy efficiency of DNN inference systems. Then, Section 2.5 presents an overview of different reliability threats that can affect the functionality of nano-scale CMOS devices. The section also highlights the impact such threats can have on the accuracy of state-of-the-art DNNs. Towards the end, in Section 2.6, the chapter covers different low-cost techniques designed for mitigating reliability threats in DNN inference systems without incurring huge energy, area or performance overheads. **[Parts are Published as [HHS18], [HKP+18], [HS20a], and [HHS20a] in DATE'18, IOLTS'18, IOLTS'20, and SCOPES'20, respectively]**

Chapter 3 - Analytical Models and Design Space Exploration of Approximate Modules: This chapter presents analytical models for efficient design space exploration of approximate adders. The chapter presents a generic accuracy-configurable adder model for Low-Latency Approximate Adders (LLAAs) in Section 3.1. Then, Section 3.2 covers design space exploration of LLAAs and presents the proposed Quality-Area Optimal Adder ($QuAd_o$) configurations. The coverage of the proposed accuracy-configurable adder model is presented in Section 3.3 along with the results that highlight the significance of $QuAd_o$ configurations. Afterwards, Section 3.4 highlights the limitations of $QuAd_o$, specifically, the unexplored design space of Low-Power Approximate Adders (LPAAs). To effectively compute the error estimates of LPAAs, Section 3.5 presents the proposed methodology, PEMACx, for computing the PMF of error of approximate adders that are composed of cascaded adder units. The usability of the model is highlighted in Section 3.6 which is then followed by the limitations of PEMACx in Section 3.7. A novel Data and Application-aware Error estimation Methodology for approximate adders (DAEM) is then presented in Section 3.8 to overcome the limitations of the earlier models. Sections 3.9 and 3.10 highlight the usability and limitations of DAEM (respectively), followed by the links to the open-source libraries in Section 3.11. **[Published as [HHHS17] and [HHHS20] in DAC'17 and DAC'20, respectively]**

Chapter 4 - Cross-Layer Optimizations for DNNs: This chapter presents the proposed cross-layer methodology for optimizing DNNs in Section 4.1. Then, Section 4.2 presents different case studies to highlight the effectiveness of each individual technique employed in the cross-layer methodology as well as their combined benefits. In particular, Section 4.2 presents a novel structured pruning methodology in Section 4.2.1, the impact

of various degrees of quantization on the accuracy of pruned DNNs in Section 4.2.2 and the impact of hardware-level approximations in Section 4.2.3. **[Published as [HMA+18] and [HS22] in JOLPE'18 and MICPRO'22, respectively]**

Chapter 5 - Neural Processing Arrays for Efficient DNN Inference: This chapter presents the proposed techniques for improving the performance and energy efficiency of Neural Processing Arrays (NPUs) used in DNN accelerators. Towards this, first, Section 5.1 highlights different possible real-world scenarios for optimizing DNNs. The section highlights that retraining is not possible in some scenarios, and therefore, it is important to have optimization techniques that can still be employed to enable highly energy and performance-efficient DNN inference. Then, Section 5.2 presents the impact of functional approximations of arithmetic modules in DNN accelerators on the accuracy of DNNs. Afterwards, Section 5.3 presents the proposed concept of curable approximations and how it can be employed to design an efficient neural array. Section 5.4 then highlights the potential of data-driven approximations and presents a novel data representation format to enable significant simplification of MAC units used in neural arrays. **[Published as [HKS19] and [HSM+22] in DAC'19 and IJCNN'22, respectively]**

Chapter 6 - A Low-cost Technique for Mitigating the Effects of Permanent Faults in DNN Accelerators: This chapter presents the proposed concept of fault-aware mapping for mitigating permanent faults in the computational array of a systolic array-based DNN accelerator. In particular, Section 6.1 presents the motivation behind the work. Section 6.2 presents the proposed saliency-driven fault-aware mapping methodology for salvaging DNN accelerators having permanent faults in the computational arrays. Then, the effectiveness of the proposed technique for different DNNs and fault rates is presented in Section 6.3. **[Published as [HS20b] in RSTA'20]**

Chapter 7 - Aging Mitigation for Improving the Lifetime of On-Chip Weight Memories in DNN Accelerators: This chapter presents the proposed aging mitigation framework for mitigating NBTI-induced aging in on-chip weight memories of DNN accelerators. First, Section 7.1 presents the motivation behind the work. Then, Section 7.2 presents an overview of the proposed aging mitigation framework. Afterwards, Section 7.3 presents an analysis that highlights the impact of using different data representation formats and quantization methods on the aging of SRAM-based on-chip weight memory cells. The proposed aging-mitigation scheme along with the supporting microarchitecture designed for mitigating NBTI-aging in the on-chip weight memory of DNN accelerators is then presented in Section 7.4. **[Published as [HS21] in DATE'21]**

Chapter 8 - Summary and Future Outlook: This chapter concludes the thesis in Section 8.1 and presents an outlook on potential future directions towards robust and energy-efficient DNN systems in Section 8.2.

CHAPTER 2

Background and Related Work

This chapter presents the background knowledge related to deep learning, types of DNNs and state-of-the-art works related to energy-efficient and reliable DNN inference. In particular, Section 2.1 presents the basics of deep learning and DNNs. Section 2.2 covers the basics of DNN hardware accelerators. Then, Section 2.3 highlights the key design issues towards enabling efficient DNN inference. Afterwards, Section 2.4 highlights state-of-the-art techniques for improving the energy-efficiency of DNN inference. The section covers both software-level and hardware-level techniques that can contribute towards improving the energy efficiency of DNN inference systems. Section 2.5 highlights different reliability threats, how they manifest in DNN systems and their impact on DNN accuracy. Then, Section 2.6 covers different state-of-the-art techniques for mitigating hardware-induced reliability threats. The section mainly focuses on low-cost techniques that can improve the resilience of DNNs against reliability threats without incurring huge overheads.

2.1 Deep Learning: Basics and Terminologies

2.1.1 Overview of Neural Networks

A Neural Network (NN) is an interconnected network of neurons. These neurons are arranged in the form of layers (see Fig. 2.1(a)) to learn hierarchical representations of data. A NN composed of three or more layers is typically termed as a Deep Neural Network (DNN). The basic functionality of the most common type of neuron used in state-of-the-art NNs can mathematically be written as:

$$O = f(\sum_{i=1}^{n} w_i * a_i + b) \tag{2.1}$$

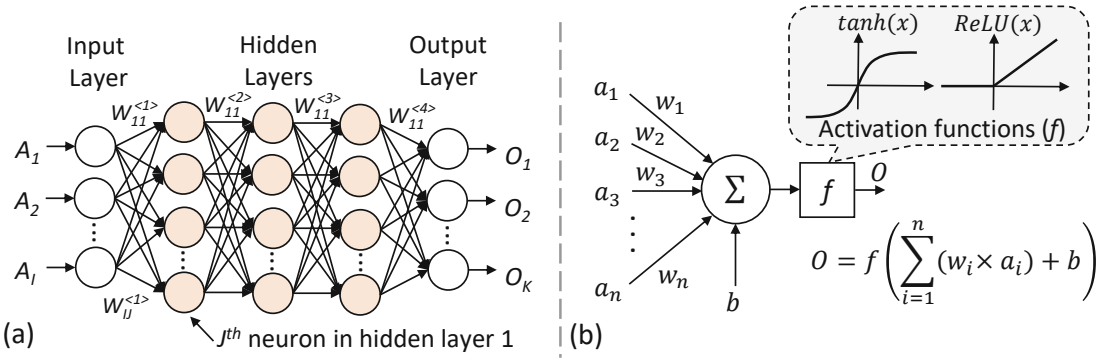


Figure 2.1: (a) Illustration of a fully-connected neural network. (b) Functionality of a neuron.

Here, O represents the output, a_i represents the i^{th} activation (input), w_i represents the i^{th} weight, b represents the bias, $f(.)$ represents a non-linear activation function (e.g., ReLU and tanh) used to introduce non-linearity in the network and n represents the total number of inputs/weights, as shown in Fig. 2.1(b).

The most common type of DNN is a fully-connected neural network, a.k.a. Multi-Layer Perceptron (MLP). An example of fully-connected network is shown in Fig. 2.1(a). As can be seen in the figure, in a fully-connected network, all the neurons in each layer are connected with all the neurons in the corresponding previous layer and all the neurons in the corresponding next layer. The weights of the network can be represented using $W_{i,j}^{<l>}$ notation, where $W_{i,j}^{<l>}$ represents the weight of the connection between neuron i of layer $l-1$ and neuron j of layer l. The functionality of each neuron in a fully-connected network can thus be defined using the following equation:

$$A_j^{<l>} = f^{<l>}(\sum_i W_{i,j}^{<l>} \times A_i^{<l-1>} + b_j^{<l>}) \tag{2.2}$$

where, $b_j^{<l>}$ represents the bias and $A_j^{<l>}$ represents the output of neuron j in layer l. $f^{<l>}(.)$ represents the activation function used for all neurons in layer l. The activation function can be any non-linear function; however, Sigmoid, Tanh, and ReLU are the most commonly used activation functions.

2.1.2 Overview of Deep Neural Network Training

The main advantage of deep learning over conventional machine learning techniques is that it enables the users to train high accuracy models directly using raw data. The models are trained using the back-propagation algorithm [LBH15]. In the back-propagation algorithm, first, a batch of data samples is passed through the network to compute the loss, which is typically based on the difference between the observed and the expected outputs of the model. The loss is then used to compute gradients for all the weights and

biases of the model. The computed gradients are then used to update the corresponding weights and biases in a way that reduces the loss. To achieve high accuracy, the training dataset is expected to be large and a true representative of the expected test set. The training set is usually divided into small batches, and one batch is used at a time to update the model parameters. This type of training process is typically referred to as mini-batch gradient descent. Note that one complete pass over the training set is defined as one epoch, and proper training of a model can require up to a few hundred epochs, depending on the complexity of the problem.

DNN training is a delicate task as it can lead to a number of problems. One of the most prominent issues associated with DNN training is over-fitting (see Fig. 2.2). Over-fitting mainly refers to the problem in which the model learns not only the important features in the training data but also the noise, which leads to reduced performance on unseen test data. To achieve high accuracy on unseen test data, it is important to avoid over-fitting and ensure that the model is able to generalize well. Various solutions have been proposed in the literature to avoid over-fitting. The most prominent one is early stopping. To detect over-fitting, a validation set is used, which is independent of the training and the test sets. The accuracy of the model on the validation set is monitored during training (evaluated usually after every epoch), and as soon as the validation accuracy starts decreasing after reaching a maximum value, the training process is stopped and the model with the best validation accuracy is forwarded as the output (see Fig. 2.3). This is mainly because the decrease in the validation accuracy corresponds to an increase in the generalization error. Other methods such as data augmentation and regularization are also commonly used to address over-fitting.

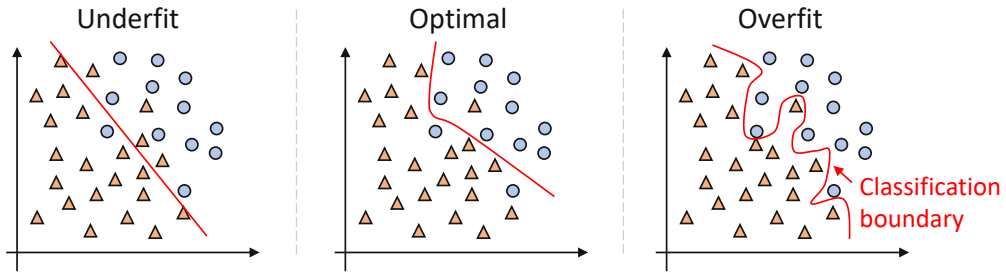


Figure 2.2: Comparison between under-fitting, optimal and over-fitting scenarios. Note, under-fitting results in high training error and high test error and over-fitting results in low training error but high test error. Only the optimal scenario results in low training error and low test error.

2.1.3 Convolutional Neural Networks

A widely used type of DNNs is the Convolutional Neural Networks (CNNs). As the name suggests, CNNs are mainly based on convolutional layers. The weights in these layers are arranged in the form of 3D filters which are traversed across the 2D space of the input activations (input feature maps) to generate the outputs. Each filter is responsible for

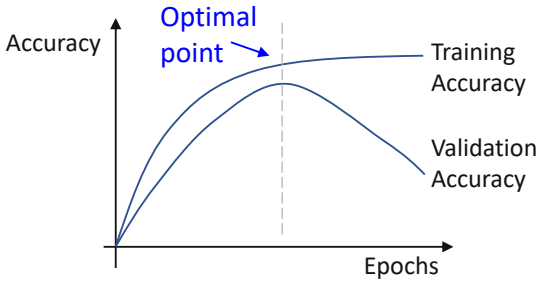


Figure 2.3: Illustration of early stopping. The validation accuracy increases to a point and after that it starts decreasing due to over-fitting.

generating one output feature map, and all the feature maps (generated using different filters of a layer) combined form the input of the subsequent layer. The dimensions of the weights of the l^{th} convolutional layer can be given as $H_f^{<l>} \times H_f^{<l>} \times M^{<l>} \times N^{<l>}$. Here, $N^{<l>}$, $M^{<l>}$ and $H_f^{<l>}$ represent the number of filters, the number of channels in each filter and the number of rows/columns of the filters of layer l, respectively. A specific weight in convolutional layer l can be represented as $W^{<l>}_{\{(r,c),(C,F)\}}$, where r, c, C and F represent the row, column, channel and the filter number, respectively. Similarly, the dimension of the output activations of convolutional layer l can be given as $x^{<l>} \times y^{<l>} \times N^{<l>}$, where $x^{<l>}$ and $y^{<l>}$ represent the number of rows and columns (respectively) in the output feature maps and $N^{<l>}$ represents the number of feature maps, which is equivalent to the number of filters in layer l. A detailed view of a convolutional layer is shown in Fig. 2.4(a), and an example CNN architecture is shown in Fig. 2.4(b).

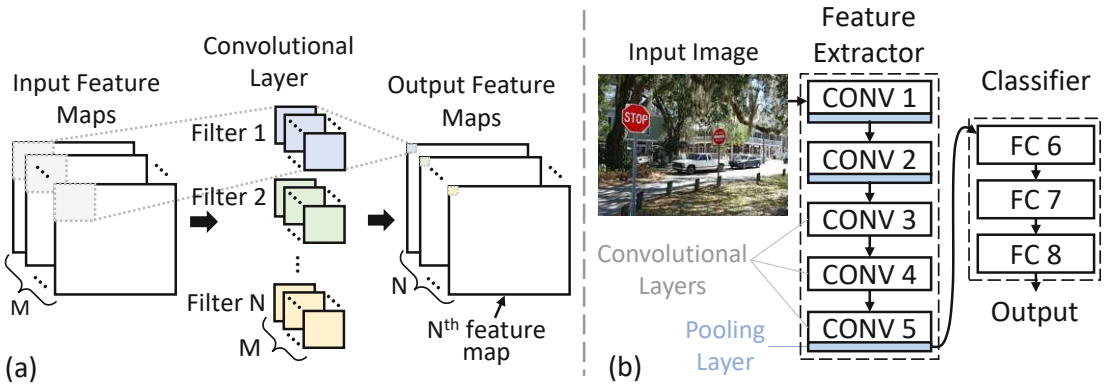


Figure 2.4: (a) Detailed view of a convolutional layer. (b) A convolutional neural network architecture for image classification application. The network is composed of five convolutional (CONV) layers and three fully-connected (FC) layers.

As illustrated in Fig. 2.4(b), a CNN contains other types of layers as well apart from convolutional and fully-connected layers. Other commonly used layers are pooling layers,

which are used to down-sample the input feature maps, and normalization layers, which are used to standardize the inputs to the next layer. Note that depending on the type of CNN, the types of convolutional layers can also be different. Other types of convolutional layers include transposed convolution and dilated convolution layers.

Apart from MLPs (discussed in Section 2.1.1) and CNNs, there are many other types of DNNs such as Recurrent Neural Networks (RNNs) and Graph Neural Networks (GNNs). RNNs are usually designed to process variable length sequences, such as speech and text, while GNNs are used to process graph data. However, without any loss of generality, this thesis mainly focuses on CNNs for evaluations as they are widely used for benchmarking embedded deep learning systems.

2.2 DNN Hardware Accelerators

DNNs fall under the umbrella of embarrassingly-parallel workloads and, therefore, can significantly benefit from hardware-level parallelism. Towards this, several specialized hardware accelerator designs have been proposed to enable resource-efficient DNN inference at the edge. Some of these accelerator designs include Tensor Processing Unit (TPU) [JYP+17], Eyeriss [CKES17], DaDianNao [CLL+14], ShiDianNao [DFC+15], Cambricon-X [ZDZ+16], FlexFlow [LYL+17], SCNN [PRM+17] and Bit Fusion [SPS+18]. These accelerators are mainly composed of (1) massively-parallel neural arrays to perform a large number of MAC operations in parallel and (2) dedicated memory hierarchy to minimize costly DRAM accesses and maximize local data reuse.

The TPU [JYP+17] is one of the prominent DNN accelerators developed by Google mainly to accelerate DNN inference process for cloud-based applications. At the core, the TPU employs a systolic array composed of 256×256 Processing Elements (PEs). Each PE contains a MAC unit and a few registers. The PEs are connected in a 2D grid-like manner to enable high local data reuse and reduce the number of costly memory accesses. Most of the case studies in this thesis are based on a similar hardware design. Fig. 2.5a presents an overview of the hardware, Fig. 2.5b presents the details of the PE design, and Fig. 2.5d together with Fig. 2.5c presents how filters are unrolled and mapped onto the processing array of the architecture. Fig. 2.5c also illustrates the architectural details of the processing array. As can be seen from the figure, each PE mainly communicates with only its immediate neighboring PEs, and the boundary PEs are the only ones that communicate with modules outside of the processing array to receive weights and activations or to output partial sums.

Fig. 2.6 presents the flow followed to process convolution layers using the array presented in Fig. 2.5. Assuming the number of columns in the array to be K and the number of rows to be N, first a set of $N \times K$ weights from K different filters are mapped onto the processing array using the vertical weight channels. The mapping is carried out using the flow presented in Fig. 2.5d, where each array column is mapped with weights from a different filter and only N number of weights from a filter are mapped at one time. The weights are then kept stationary inside the array while the activation values are fed from

the left using the activation channels. The activations have to be arranged in such a way that PEs can operate in lockstep to generate the partial sums (see Fig. 2.6c).

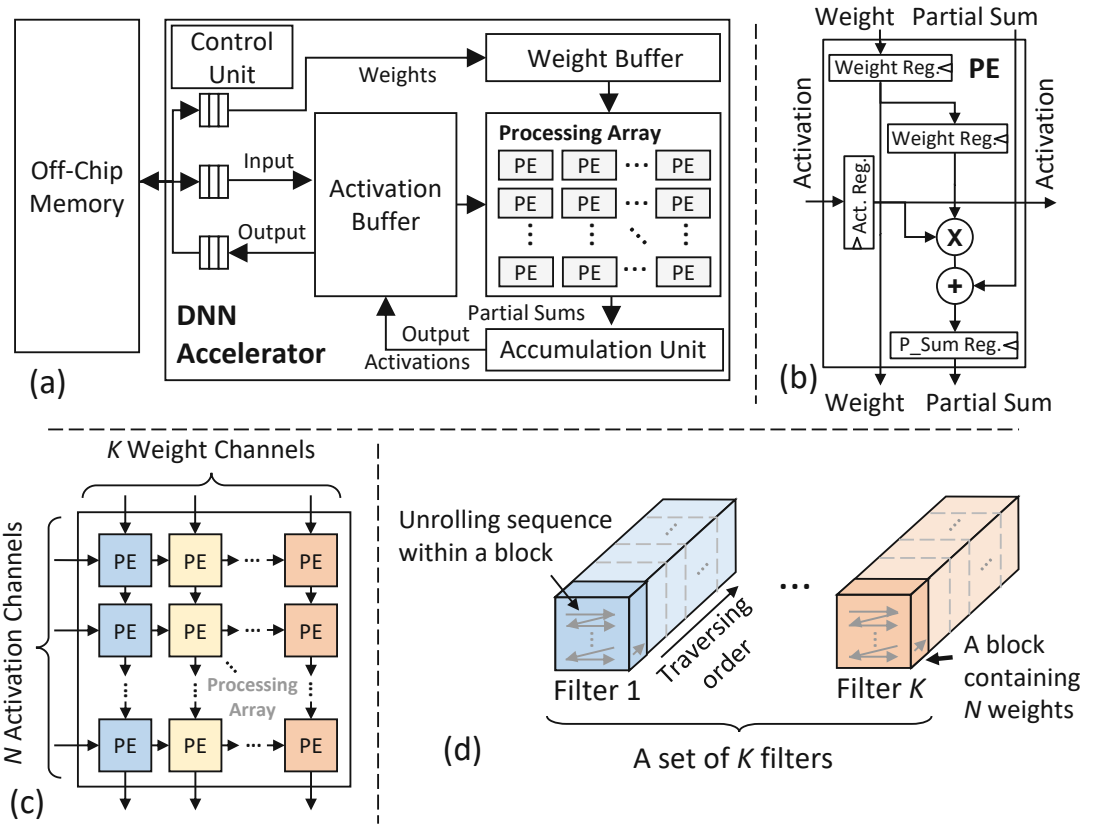


Figure 2.5: (a) Overview of a DNN hardware accelerator. (b) A detailed view of the PE architecture. (c) A detailed view of a TPU-like systolic array. The shades of the PEs together with (d) show the mapping policy adopted for such arrays.

For example, in the first clock cycle, the top left corner PE multiplies the first weight from the first filter with the first input activation from the first input set and then stores the partial sum output in its partial sum register (see Fig. 2.6c). In the second clock cycle, the PE in the second row first column receives the second input activation from the first input set, multiplies it with the second weight from the first filter, adds the product with the partial sum received from the above neighbor (which was generated in the previous clock cycle), and then stores the updated partial sum in its partial sum register. In the same clock cycle, the PE in the top left corner multiplies the first activation from the second input set with the first weight of the first filter and the PE in the first row second column multiplies the first activation from the first input with the first weight of the second filter. This way, after N clock cycles, the array outputs its first partial sum from the first column, and at its peak, it can perform $N \times K$ MAC operations in parallel and can output K partial sums per clock cycle.

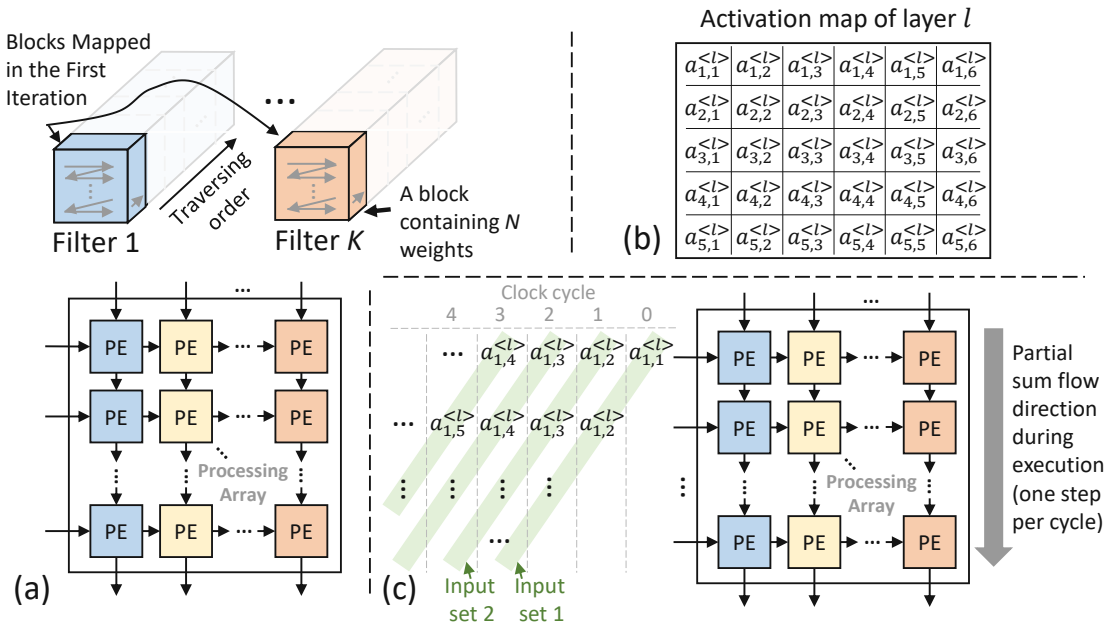


Figure 2.6: Flow for performing convolution using the hardware shown in Fig. 2.5. (a) Mapping of weights onto the processing array. (b) An example activation map. (c) Sequence of execution.

Note, if the number of weights in a filter is more than the number of rows in the array, the whole filter cannot be mapped to the array at the same time, as also illustrated in Fig. 2.6a. In such a case, the filter has to be divided into blocks of size N. The partial sums generated from the blocks are accumulated using the accumulation units (see Fig. 2.5a) to generate the final outputs.

2.3 Design Issues for Efficient DNN Inference

2.3.1 Computational Complexity of DNNs

As highlighted in Section 1.1 with the help of Fig. 1.3, high-accuracy DNNs are in general highly compute intensive as they require a large number of floating-point operations (FLOPs) to process each input sample. For example, the AlexNet model [KSH12] requires around 725 MFLOPs to process one input image while the VGG16 model [SZ14], which offers better accuracy than the AlexNet model, requires around 15.5 GFLOPs per input image. This compute-intensive nature of DNNs translates to high energy/power requirements and makes it challenging to achieve efficient deployment of these DNNs in resource-constrained scenarios. Although specialized DNN hardware accelerators are commonly used to achieve better efficiency (as highlighted in Section 2.2), the gains are usually insufficient to meet real-world constraints. Therefore, other optimization techniques are required that can reduce the computational as well as memory requirements

of DNNs without compromising their application-level accuracy.

2.3.2 Diversity in State-of-the-art DNN Architectures

Over the years, different DNN architectures have been proposed. Even within CNNs designed for image classification application, designs range from simple CNNs composed of conventional convolutional and fully-connected layers (e.g., AlexNet and VGG) to ResNets composed of residual blocks to CNNs composed of depth-wise separable and point-wise convolutional layers (e.g., MobileNet). This diversity in DNN architectures makes it challenging for the same accelerator to offer optimal efficiency gains for two different DNN architectures [CYES19]. Moreover, within each DNN, layers can have different configurations, which further adds to the difficulty. To address this challenge, configurable hardware accelerators such as FlexFlow [LYL+17] and EyerissV2 [CYES19] have been proposed in the literature. These accelerators are not only capable of supporting various configurations for the compute fabric but are also capable of supporting different dataflows. This flexibility together with works like SmartShuttle [LYL+18] that are used to optimize off-chip memory accesses, enables the user to define optimal execution policy for each individual DNN layer based on the user-defined constraints.

Given that state-of-the-art works can be employed to define and enable optimal DNN execution policy, a part of this thesis focuses on reducing the computational complexity of DNNs through software-level optimizations and designing complementary techniques that can further add to the efficiency of DNN hardware accelerators through approximations.

2.4 State-of-the-Art Techniques for Enabling Energy Efficient DNN Inference

To enable highly resource-efficient DNN inference, various optimization techniques have been proposed in the literature. The spectrum of these techniques spans the complete hardware-software computing stack. At the software level, techniques like Neural Architecture Search (NAS), pruning and quantization have been proposed to reduce the size and computational complexity of DNNs. While at the hardware level, techniques like hardware-level approximations, design of specialized DNN accelerators and aggressive voltage scaling are employed. Table 2.1 presents a summary of the key state-of-the-art techniques proposed for improving the energy and performance efficiency of DNN inference process. The details of the techniques are presented in the following subsections.

2.4.1 Software-level Optimizations

This section highlights different state-of-the-art software-level optimization techniques that can be employed to reduce the size and complexity of DNNs.

Table 2.1: Techniques for improving the energy and performance efficiency of DNN inference process.

Abstraction Layer	Type of Approximations	Related work	Brief Description	Cost	Benefits
Software	Unstructured Pruning	[HMD15] [LKD+16]	This technique iteratively removes less significant parameters from a DNN (regardless of their locations) to reduce its memory footprint and computational requirements. The technique is mainly useful for reducing the memory footprint of DNNs, and in cases where the pruning policy is not aligned with the underlying inference hardware, it can lead to energy overheads.	Loss in accuracy or high training cost	Less memory requirements and less number of computations
	Structured Pruning	[AHS17a] [NML+20] [EES+20]	It removes less significant groups of parameters from a DNN to reduce its size and computational complexity. The groups are based on some predefined templates that can lead to better accuracy-efficiency trade-offs.	Loss in accuracy or high training cost	Less memory requirements, less number of computations, and structured sparsity
	Quantization	[GAGN15] [HMD15] [JKC+18]	It converts the DNN weights and activations from floating-point to limited-precision fixed-point/integer format. It can also be performed by grouping the parameters into multiple clusters and assigning a single floating-/fixed-point value to all the parameters in the same cluster.	Loss in accuracy or high training cost	Less memory requirements and less complex hardware modules
	Neural Architecture Search (NAS)	[EMH19] [BMO+21]	It enables to search for more accurate models from a predefined search space. It mainly replaces heuristics-based manual DNN architecture design process with sophisticated push-button methodologies. Moreover, hardware-aware NAS techniques have also been proposed that take into account the execution latency, energy consumption, memory footprint, etc. of candidate options as well along with their accuracy to reach to efficient-yet-accurate DNNs.	Loss in accuracy or high training cost	Less complex and/or high accuracy DNNs
Hardware	Hardware Accelerators	[JYP+17] [CYES19] [LYL+17] [SPS+18]	In this group of works, specialized hardware is designed to accelerate the execution of compute-intensive operations involved in DNN inference. The design is based on user-defined constraints and can be for improving the latency or energy characteristics of the system.	Specialized hardware cost	Low power, energy, area, and/or latency
	Functional Approximations	[GMP+11] [KGE11] [REHS+16]	In this group of works, the functionality of arithmetic modules is simplified/approximated to reduce the hardware cost and achieve power, energy, area and/or latency benefits.	Less reliable	Low power, energy, area and/or latency
	Voltage/ Frequency Scaling	[RBKS17] [VCC+13] [ZRGG18] [BOO+23]	In this group of works, the functionality of arithmetic modules is approximated through aggressive voltage/frequency scaling that leads to timing errors in some locations.	Less reliable	Low power, energy and latency
	Data Approximations	[ATBS11] [SMBJ14] [ACV05] [YPT+16] [LPMZ11]	In this group of works, the functionality of memory subsystems is simplified/ approximated, for example, using memory cells composed of less number of transistors or by using approximate caches or DRAMs.	Less reliable	Low power, energy, area and/or latency

2.4.1.1 Pruning

High-accuracy DNNs are intrinsically over-parameterized and, therefore, have to be optimized, especially for edge-based applications. Pruning, in the context of DNN optimizations, mainly refers to removing non-essential (less important) parameters from a DNN to reduce its size and complexity. Various types of pruning techniques have been proposed in the literature, depending on the granularity and user requirements. For example, pruning can be applied at element level, where each individual weight/parameter is checked against a criterion and pruned if it falls in the non-essential category according to the defined criterion. Apart from removing individual parameters, groups of parameters, such as filters, filter kernels and even layers, can also be removed to reduce the complexity and memory footprint of DNNs. In general, element-wise pruning is referred to as fine-grained pruning, while all other types, in which groups of parameters are removed, fall under the umbrella of course-grained pruning.

Fine-grained Pruning: The most convenient and most rewarding technique in terms of reducing the number of parameters in a DNN is fine-grained (element-level) pruning, in which *each individual* parameter is checked against a criterion and removed if it falls in the non-essential category. Fig. 2.7(b) presents a pruned version of the DNN shown in Fig. 2.7(a), where pruning is achieved through fine-grained pruning. The figure clearly shows that fine-grained pruning results in unstructured sparse matrices. Earlier works towards pruning of neural networks include the Optimal Brain Damage [LDS89] and the Optimal Brain Surgeon [HS92] methods. Recently, Han et al. [HPTD15] proposed a three-step method to reduce the size and computational complexity of DNNs. First, they train a DNN to learn which connections are important. Then, they remove non-essential (unimportant) connections from the DNN. Finally, they fine-tune the remaining weights to regain the accuracy lost due to pruning. They further proposed that learning the right connections is an iterative process. Therefore, the process of pruning followed by fine tuning should be repeated multiple times, with only a fraction of weights removed at a time, to achieve higher compression ratios. Deep Compression [HMD15] employed a similar method for pruning and combined it with weight sharing and Huffman coding to significantly reduce the memory footprint of DNNs.

Course-grained Pruning: Although fine-grained pruning has the capability to significantly reduce the number of parameters (and memory footprint) of DNNs, it does not guarantee energy or latency benefits. This is mainly because (1) the network parameters are stored in a compressed format and, therefore, have to be uncompressed before corresponding operations and (2) the underlying hardware architecture (in most cases) is not specifically designed to take advantage of fine-grained sparsity in pruned DNNs. Therefore, specialized hardware accelerators are required to process data using compressed sparse DNNs. Some examples of such accelerators include EIE [HLM+16] and SCNN [PRM+17]. Yu et al. [YLP+17] argued that it is essential to align the pruning method to the underlying hardware architecture to achieve efficiency gains. A mismatch between the type of sparsity a pruning method can introduce and the organization and architecture of processing units in the hardware can lead to high overheads. Based on

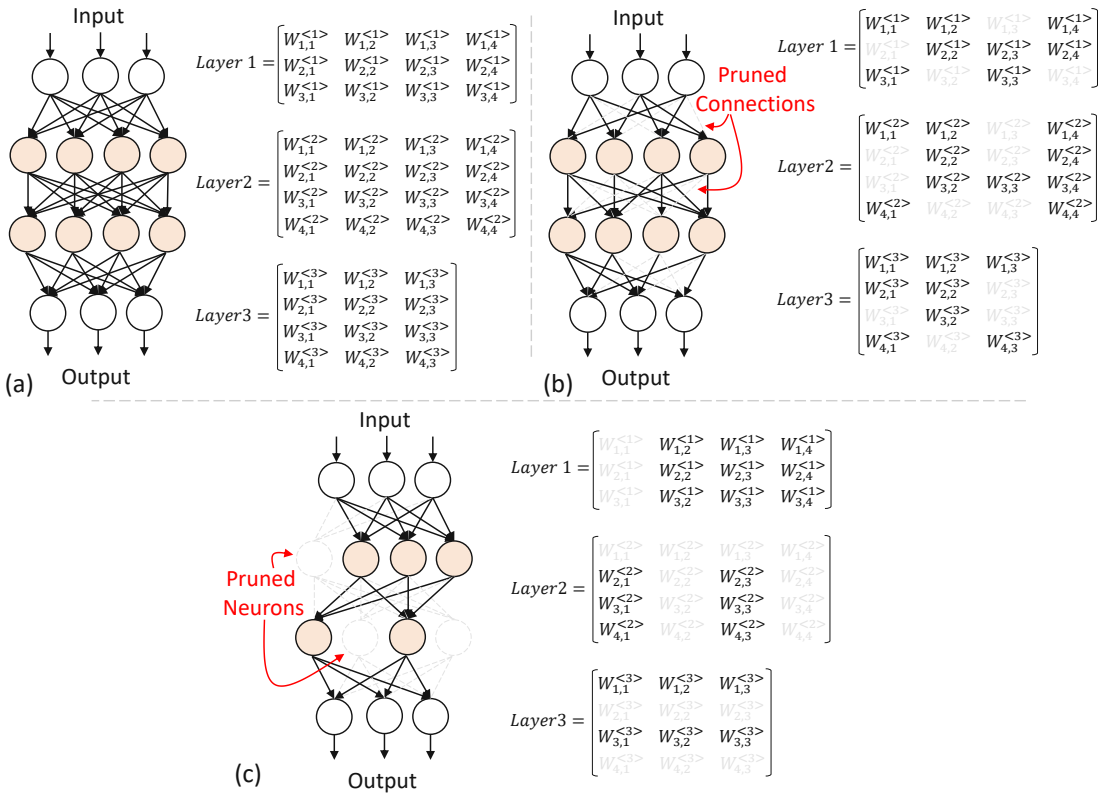


Figure 2.7: (a) An example FCNN. (b) An example of connection pruning. (c) An example of neuron pruning.

this observation, Yu et al. in [YLP+17] proposed two different pruning methods, i.e., SIMD-aware weight pruning and node pruning. SIMD-aware weight pruning maintains weights in aligned fixed-size groups to fully utilize the available SIMD units in the hardware. Node pruning removes less significant nodes from the network to reduce the computational complexity and memory footprint of DNNs while maintaining the dense matrix format of weight matrices (e.g., see Fig. 2.7(c)).

Similar to node pruning, several other structured pruning techniques have also been proposed. Anwar et al. [AHS17a] highlighted that conventional (fine-grained) pruning techniques result in irregular network structure that incurs high overhead cost and also result in under-utilization of parallel-processing units in regular hardware accelerators during inference process. To address this issue, they proposed various types of structured sparsity for CNNs, i.e., channel-wise, kernel-wise and intra-kernel strided sparsity. Fig. 2.8b presents an example of filter pruning, while Fig. 2.8c highlights the main differences between different types of structured pruning for CNNs.

Apart from the above-mentioned structured pruning techniques, pattern-based pruning [NML+20] and layer pruning [EES+20] techniques also exist in the literature. Pattern-

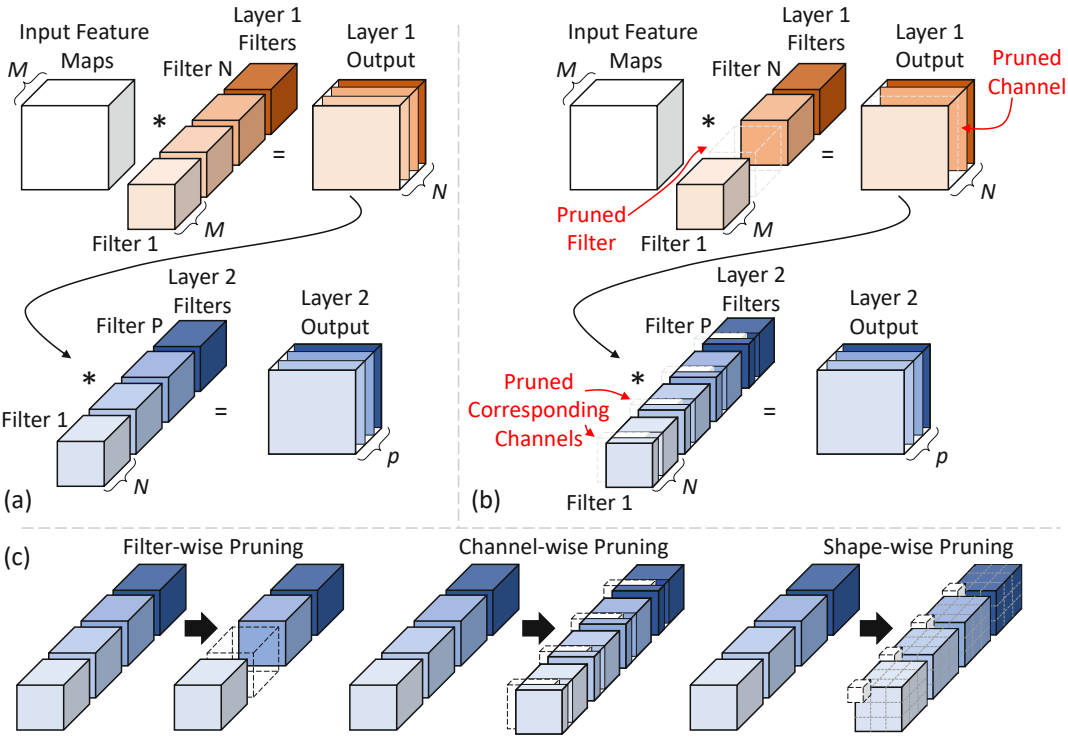


Figure 2.8: (a) Processing flow of two consecutive convolutional layers. (b) An example of filter (or channel) pruning. (c) Different types of structured pruning.

based pruning falls in between structured pruning and fine-grained pruning, as it allows to prune each filter kernel differently based on a predefined set of templates. The key advantage of pattern-based pruning over filter pruning is that it enables comparatively more compression and efficiency gain. Layer pruning techniques add another dimension to the compression by enabling the designers to drop complete layers from DNNs. Layer pruning is mainly beneficial for cases with stringent latency constraints.

2.4.1.2 Quantization

Quantization refers to the process of mapping values from a high-precision range to a low-precision range with lesser number of quantization levels. The number of quantization levels in the range defines the number of bits required to represent and store each data value. Therefore, a low-precision format with less number of quantization levels can significantly reduce the storage requirements and inference cost of DNNs.

The main goal of DNN quantization is to improve the storage and energy efficiency of DNNs without affecting their accuracy. Various techniques have been proposed to achieve this goal. The most commonly used quantization is 8-bit range linear quantization, where floating-point weights and activations are converted to 8-bit fixed-point format. The range

linear quantization further can be divided into two types: (1) asymmetric quantization; and (2) symmetric quantization. In asymmetric quantization, the minimum and maximum observed values in the float range are mapped to the minimum and maximum possible values (respectively) in the integer/fixed-point range as shown in Fig. 2.9c; however, in symmetric quantization, the float range is defined using the maximum absolute observed value as shown in Fig. 2.9b.

Non-linear (or non-uniform) quantization techniques have also been studied in the literature [JVS+19][HMD15]. These techniques are mainly inspired by the non-uniform probability distribution of DNN data structures [JVS+19]. Fig. 2.9a presents an illustration of the difference between uniform and non-uniform quantization schemes. The key advantage of non-uniform quantization over uniform quantization in the context of DNNs is that it can significantly reduce the average quantization error when both are subjected to the same number of quantization levels budget. This is mainly because the data distribution of DNN data structures is not uniform and, in general, concentrated towards zero. However, note that non-uniform quantization leads to energy and performance improvements only when used with specialized hardware, as general-purpose CPUs, GPUs and accelerators are usually designed for conventional fixed-point and floating-point number formats.

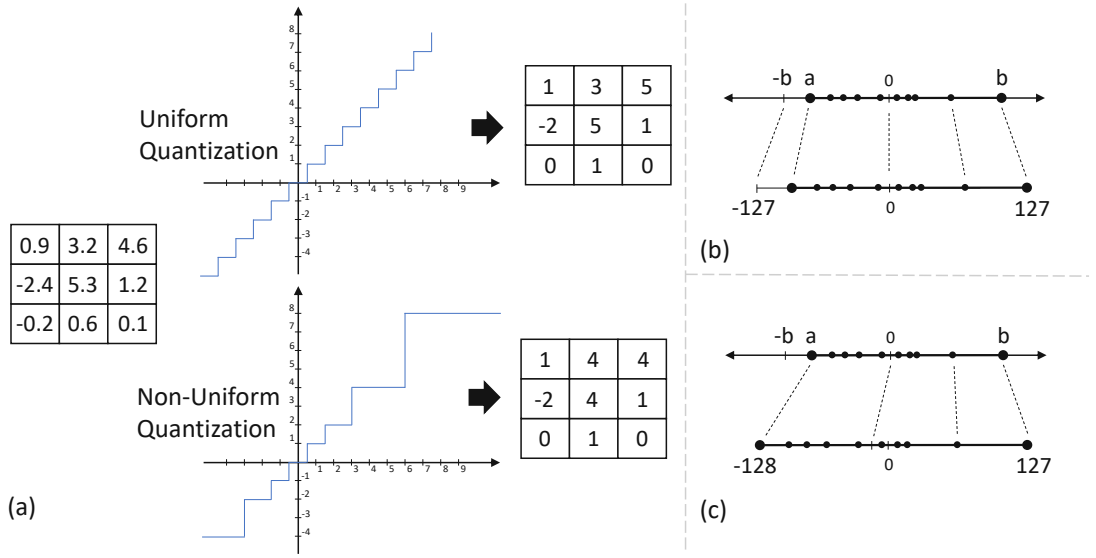


Figure 2.9: (a) Difference between uniform and non-uniform quantization. (b) 8-bit symmetric quantization. (c) 8-bit asymmetric quantization.

To push quantization to its limits, works like XNOR-Net [RORF16], Binarized Neural Network (BNN) [HCS+16] and DoReFa-Net [ZWN+16] have explored the potential of aggressive quantization. However, aggressive quantization leads to significant accuracy loss even with retraining. Mixed-precision quantization techniques have also been studied, where different precision formats can be used for different layers, filters and even channels

in the same DNN [WLL+19][GKD+21]. However, mixed-precision quantization is only useful in cases where the underlying hardware is capable of supporting different precision formats.

2.4.1.3 Neural Architecture Search

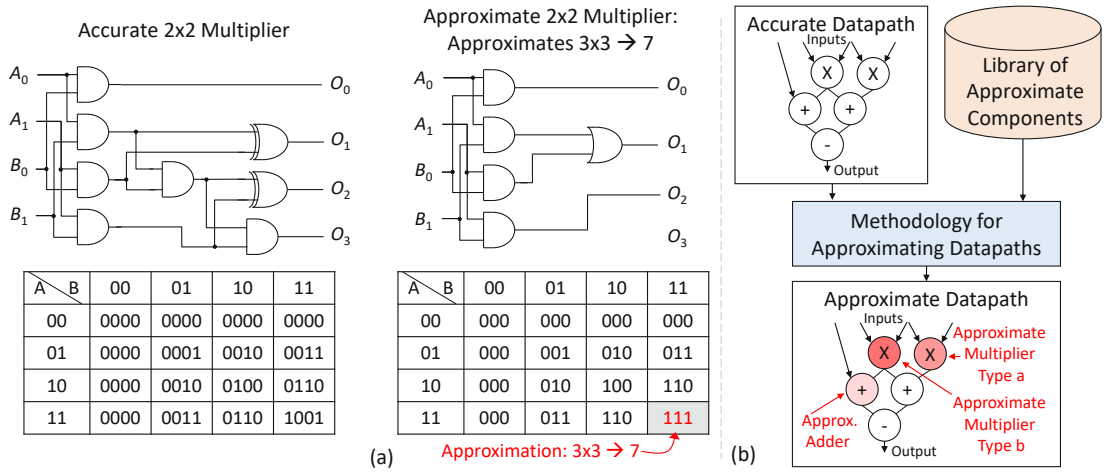
Designing DNNs manually based on heuristics is a time-consuming process. Therefore, to meet the growing demand for highly accurate and efficient models for edge-based applications, different automated Neural Architecture Search (NAS) techniques have been proposed [EMH19], including hardware-aware NAS [BMO+21]. Fig. 2.10 shows the general flow adopted for NAS. First, a search space is defined using a base architecture and the possible types of modules and connections each block in the base architecture can have. Then, the search strategy defines how the search space is traversed. The search strategy faces the exploration/exploitation dilemma, where the search strategy is expected to converge to a solution fast, while ensuring that the solution is highly effective (close to the optimal solution). The evaluation strategy defines the process for estimating the accuracy and performance metrics (such as latency, energy consumption, and memory footprint) of candidate solutions. The estimates are then sent back to the search strategy block to guide the search towards better architectures. *Note that all NAS algorithms involve training of candidate solutions and, therefore, are effective only in cases where reasonable compute resources are available.* Even though most of the state-of-the-art NAS techniques employ surrogate models for estimating the accuracy and performance metrics of candidate solutions to accelerate the search process, the complete search can still take multiple GPU days for complex problems and larger design spaces. Moreover, surrogate models bring in other challenges such as training a surrogate for each objective (in a multi-objective optimization) independently leads to false dominant solutions as each surrogate brings some approximation error. A prominent work towards addressing this challenge is Hardware-Aware Pareto-Ranking NAS (HA-PR-NAS) [BOEMN23], where a single surrogate is used to rank the models instead of estimating their accuracy and performance metrics.

Figure 2.10: Overview of Neural Architecture Search (NAS) process.

2.4.2 Hardware-level Approximations

Approximate Computing (AC) is an emerging computing paradigm that enables the designers to push the boundaries of efficiency gains beyond what can typically be achieved using conventional techniques [VCRR15][SHR+16][XKM16][Mit16]. AC achieves this by trading application quality for efficiency gains. This accuracy-efficiency trade-off can be achieved using various methods, for example, through functional approximation

of arithmetic modules (through logic simplification, as shown in Fig. 2.11) or through voltage underscaling.



A \ B	00	01	10	11
00	0000	0000	0000	0000
01	0000	0001	0010	0011
10	0000	0010	0100	0110
11	0000	0011	0110	1001

A \ B	00	01	10	11
00	000	000	000	000
01	000	001	010	011
10	000	010	100	110
11	000	011	110	111

Figure 2.11: (a) An example of functional approximation, where a 2x2 multiplier is approximated to reduce the area and energy costs. (b) A generic flow for approximating datapaths.

Functional Approximations: A number of recognition, mining and synthesis applications are inherently error resilient, and this error-resilience property of the applications can be exploited to trade a bit of quality for a significant amount of energy, performance and area efficiency. Functional approximations of the arithmetic modules in the datapaths of hardware accelerators is one of the widely explored ways for achieving quality-efficiency trade-offs. These approximations mainly involve modifications in the circuitry/architecture of the modules to improve their hardware characteristics. Towards this, a number of efforts have been carried out for designing approximate adders [GMP+11][GMRR12][SAHH15][MHH+16][VBI08], multipliers [REHS+16][KGE11] and dividers [VKAK+17]. For high-performance applications, in which latency is the foremost constraint, low-latency approximate adders have been proposed, for example, ETA-II [ZGY09], ETA-IIM [ZGY09], ACA [VBI08][KK12], GDA [YWY+13]. For low-power applications, low-power approximate modules, such as low-power approximate adders [GMP+11][MAFL10], multipliers [KGE11][REHS+16] and dividers [VKAK+17] have been proposed. An example of functional approximate of 2×2 approximate multiplier module is shown in Fig. 2.11a.

Other Types of Approximations: Apart from functional approximation of arithmetic modules, voltage scaling can also be exploited to achieve quality-efficiency trade-off in the computational units of a hardware accelerator [RBKS17][VCC+13]. Various data approximation techniques have also been proposed to reduce the energy consumption associated with memory subsystems. These techniques mainly include load value approximation [SMBJ14][SSMJ15][YPT+16], memoization [ACV05][KKS15][SJLM14], memory access skipping [SLJ+13][YPT+16], refresh rate reduction [LPMZ11] and in-

exact reads/writes [RVF+15]. Although data approximations can also significantly contribute towards efficiency gains, in this thesis, we mainly focus on precision reduction to reduce the energy consumption associated with memory subsystems and functional approximations to improve the efficiency of computational units.

Hardware-Level Approximations for DNNs: Venkataramani et al. [VRRR14] highlighted that DNNs are mostly used for error-resilient applications, and therefore, approximations can be employed to improve the efficiency of DNN inference systems under such scenarios. Towards this, AxNN [VRRR14] proposed a selective approximation technique, where less significant neurons are approximated. AxNN [VRRR14] also proposed the concept of approximation-aware training of DNNs to counter the impact of approximation errors in the system. As approximation-aware training is not possible in some scenarios, ALWANN [MVS+19] proposed a layer-wise approximation methodology to select an appropriate type of approximate multiplier for each individual layer of the given DNN without involving training. Other techniques that include BiScaled-DNNs [JVS+19] and Compensated-DNNs [JVS+18] target different data representation formats to significantly reduce the hardware complexity and energy requirements of DNNs. Apart from functional approximation of arithmetic modules in DNN accelerators, voltage-scaling techniques such as ThunderVolt [ZRGG18], GreenTPU [PBCR19] and MATIC [KHM+18a] can also be employed to increase the energy efficiency of the DNN inference process at the cost of minor accuracy loss.

2.5 Reliability Threats

State-of-the-art high-accuracy DNNs are extremely resource hungry due to their huge memory and compute requirements [SCYE17]. Therefore, specialized hardware accelerators are employed to achieve efficient DNN inference [JYP+17][CKES17]. On the one hand, accelerators fabricated using nano-scale technology can increase the efficiency of a system, while on the other, they bring unique reliability challenges. Some of these challenges are associated with the limitations of the fabrication process, while others are due to the extremely small sizes of transistors. The following text provides a brief introduction to different hardware-induced reliability threats.

- **Soft Errors** are transient faults caused by high-energy particle strikes on a chip. These high-energy particles can be alpha particles emitted from the impurities in the packaging materials of the chip or neutrons from cosmic radiations [Bau05]. These faults manifest as bit-flips in the system and can propagate to the application layer and affect the application-level accuracy of the system. External factors such as temperature and altitude profoundly impact the Soft Error Rate (SER).
- **Aging** in nano-scale electronic devices occurs due to various physical phenomena such as Bias Temperature Instability (BTI), Hot Carrier Injection (HCI), Time-Dependent Dielectric Breakdown (TDDB) and Electromigration (EM). It typically affects the hardware characteristics of different components, for instance, the

threshold voltage (V_{TH}) of transistors [KGPK08] and width/connectivity of wires. In the early stages, aging results in timing errors, and later it can even transform into permanent faults.

- **Process Variations** are deviations in the hardware characteristics from the expected values due to imperfections in the fabrication process [RTGM13]. In general, these variations manifest as timing errors in a system and are usually addressed by adding guardbands, e.g., by reducing the operating frequency of the device. Extreme variations can even lead to **permanent faults**, which affects the yield of the manufacturing process.

To highlight the impact of the above-mentioned reliability threats that manifest as bit-flip errors at the hardware level on the application-level accuracy of DNNs, Fig. 2.12 presents the accuracy results when different number of bit-flips are inserted at different bit locations of the weights of layer 1 of the pre-trained VGG-f network[1]. For this analysis, the faults are divided into two types: (1) 0 to 1 bit-flips and (2) 1 to 0 bit-flips. Moreover, single-precision floating-point format is considered for both weights and activations (see Fig. 2.12c). Fig. 2.12a presents the results related to 0 to 1 bit-flips and Fig. 2.12b presents the results related to 1 to 0 bit-flips. As can be observed from the figure, the 1 to 0 bit-flips do not affect the DNN accuracy much (see Fig. 2.12b); however, even a single 0 to 1 bit-flip at a critical location in the DNN can drastically reduce the accuracy (see Fig. 2.12a). In general, the criticality of a bit depends on its location in the weight along with the data representation format used for representing the weight values. Moreover, the location of the fault inside the network can also have a significant impact on the DNN accuracy; however, we observed from experiments that the behavior is mainly dominated by the location within the weights as that can significantly impact the dominating path inside the network and thereby the network accuracy. A detailed study of the impact of bit-flip errors in DNNs is presented in [NANK19]. *In summary, from this analysis, it can be concluded that DNNs are not resilient to all types of faults. Some types of faults result in significant accuracy degradation, while others can be exploited to design cost-effective fault-mitigation techniques or improve the overall efficiency of the system by trading quality for efficiency.*

2.6 State-of-the-Art Techniques for Mitigating Reliability Threats in DNN Systems

As highlighted in Section 2.5, hardware-induced reliability threats can significantly degrade the accuracy of a DNN-based system. Therefore, it is essential to mitigate these threats to ensure reliable results. Conventionally, techniques like aggressive guardbanding and spatial/temporal redundancy (e.g., Error Correction Codes (ECC) [LM76], Dual Modular Redundancy (DMR) [VZBT10], Triple Modular Redundancy (TMR) [LV62] and

[1]The pre-trained VGG-f is downloaded from [VL15]

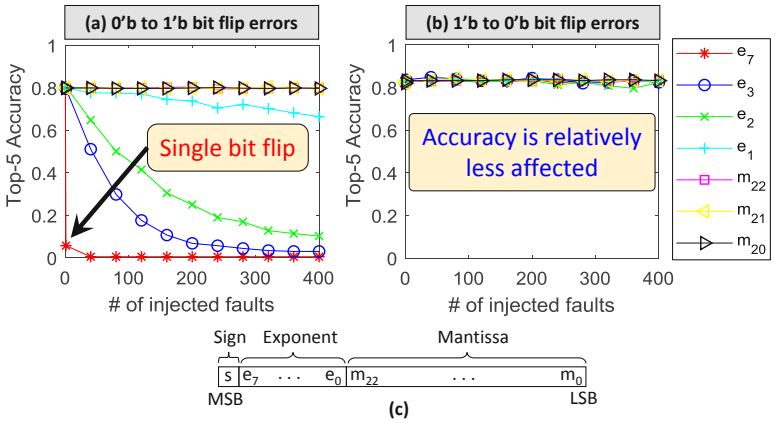


Figure 2.12: (a) and (b) show the impact of bit-flip errors on the accuracy of the VGG-f network trained for ImageNet classification; (c) Single-precision floating-point format [HKP+18], i.e., the format considered for the evaluation presented in (a) and (b).

instruction duplication [OSM02]) are used to address reliability issues. However, such techniques lead to high performance and/or energy overheads. The huge overheads of these techniques coupled with the compute- and memory-intensive nature of DNNs make them infeasible for DNN-based systems. Therefore, alternate techniques are required that can improve the resilience of DNNs against hardware-induced reliability threats at low cost. Table 2.2 presents a summary of the key low-cost techniques proposed for mitigating different types of reliability threats in DNN systems. The details of the techniques are presented in the following subsections.

2.6.1 Permanent Fault Mitigation

A major challenge associated with technology scaling is the increase in fault rates, i.e., moving to smaller technology nodes results in an increase in the number of permanent faults as well as the probability of occurrence of soft errors [Con03]. Addressing permanent faults is important as it affects the manufacturing yield of the fabrication process, which eventually affects the cost of devices [KP86]. Considering the error resilience of DNNs, one possible solution could be to simply let the faults propagate. However, prior works have shown that permanent faults in DNN accelerators can severely impact the accuracy of DNNs [ZGBG18].

To address permanent faults in systolic arrays (which are the core of most of the state-of-the-art DNN accelerators [JYP+17][CKES17][LYL+17]), Kim et al. in [KR89] and Kung et al. in [KL83] proposed techniques to enhance the fabrication yield at the cost of some performance loss. The basic idea behind these techniques is to eliminate an entire row/column for each faulty PE. This requires additional complex bypassing circuitry along with additional registers, which translates to high performance penalty, specifically at high fault rates. More sophisticated techniques have been proposed in [LJL89][EAKHAK94] that can reduce the performance loss, but at the cost of even higher design complexity.

Table 2.2: An overview of different Hardware (HW) and Software (SW) fault-mitigation techniques for deep learning inference systems.

Technique	Abstraction Layer	Related Works	Brief Description	Cost	Targeted Faults	Other Dependencies
Fault-Aware Training	SW	[KHM+18b] [ZGBG18]	It incorporates the information of faults in the training process to make the DNN aware of the faults present in the system.	Design-time cost is high	Permanent faults in memory	Fault map extraction
Fault-Aware Pruning	HW Architecture + SW	[ZGBG18]	This technique leverages the inherent resilience of DNNs to dropped computations to mitigate permanent faults in the computational array of a DNN accelerator. In case of higher number of faults, the technique is coupled with fault-aware training to offer better performance.	Run-time cost is low	Permanent faults in computational array	Modifications in HW architecture + fault map extraction
Range Restriction	SW	[Cea20] [HHS20b]	These techniques are based on the observation that faults in DNNs result in abnormal intermediate activation values. Therefore, restricting the activation values to pre-determined ranges and treating all out-of-the-range values as faulty can help improve the resilience of DNNs to faults.	Both Design-time and run-time costs are low	Transient Faults (i.e., Soft Errors)	Modifications in HW architecture in case of specialized HW
Algorithm-based Fault Tolerance	SW	[ZDL+20] [HSTK21]	This technique is based on checksum computation and is proposed to protect matrix multiplication operations. The technique has been extended to offer cost-effective error detection and correction for convolution operations as well.	Run-time cost is low	Transient faults in computational units and data corruption	No
Voltage Scaling	HW Architecture + Circuit	[ZRGG18]	This approach employs razor flip-flops to detect timing errors in the computational array of a DNN accelerator. To mitigate these errors, it exploits resilience of DNNs to dropped computations. When a timing error is detected in a PE, the computation of the next PE is dropped and the additional cycle is used to push the rectified output back into the pipeline.	Run-time cost is low	Timing errors in computational units (i.e., Aging)	No
Radiation Hardening	Circuit	[GXM14] [AGGC18]	This approach is used to make hardware less prone to radiation-induced faults using stronger cells or through designing hardware components with special characteristics that are based on resilience characteristics of DNNs. For example, SRAM cells with biased error characteristics, biased towards '0'.	Run-time cost is low	Transient Faults (i.e., Soft Errors)	No
Redundancy	HW Architecture + SW	[CW90] [Sha]	It introduces selective spatial and temporal redundancy to fortify critical components/computations against errors.	Run-time cost is moderate to high	Transient, intermittent and permanent faults	No

Another set of techniques propose to add redundancy such that each redundant PE in the architecture is dedicated for a limited region of the computing array [TH12][TF17]. These techniques offer better performance compared to the above techniques; however, as they are based on redundancy, the performance improvements are limited and the level of fault mitigation that can be achieved completely depends on the level of redundancy in the array. Along similar lines Liu et al. [LCX+21] proposed a hybrid computing architecture for fault-tolerant DNN inference. The architecture employs a set of additional (redundant) dot-product computing units to recompute all the computations mapped to faulty PEs. Note that all the above-mentioned techniques are *application agnostic*, i.e., the solutions are valid for all the applications and use cases that can benefit from systolic arrays.

As highlighted in Section 2.5, DNNs exhibit resilience to specific types of faults and this resilience can be exploited to develop low-cost fault-mitigation techniques. Towards this, Zhang et al. in [ZGBG18] proposed *Fault-Aware Pruning (FAP)* to mitigate the effects of permanent faults in DNN accelerators without affecting the performance. As the name suggests, FAP exploits resilience of DNNs to weight pruning. It achieves this by *bypassing* all the faulty MAC units in the systolic array of a DNN accelerator. This bypassing of MAC units corresponds to pruning at a higher abstraction level. Note that fault maps can be extracted at the post-fabrication testing stage using BIST (built-in-self-test)-like methods [AKS93][FFR16]. Zhang et al. [ZGBG18] also proposed hardware modifications required to realize the concept. The modifications are highlighted in Fig. 2.13. Further, they also proposed a Fault-Aware Pruning + Training (FAP+T) technique which allows to tune a model for a given faulty chip using its fault map. A generic flow for fault-aware training is shown in Fig. 2.14. Note that fault-aware retraining of DNNs is a widely used technique to mitigate faults in DNN-based systems, e.g., see [ZBG19][XXL+19][DFD+15].

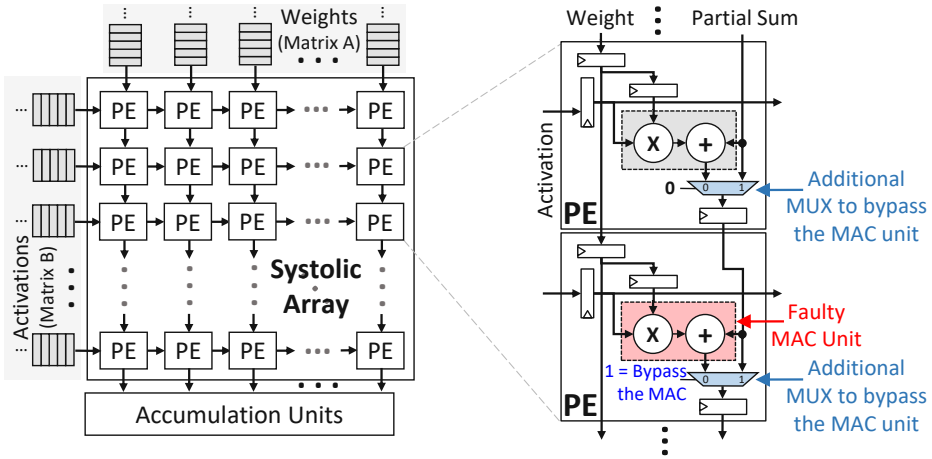


Figure 2.13: Modified systolic array design for permanent fault mitigation through fault-aware pruning

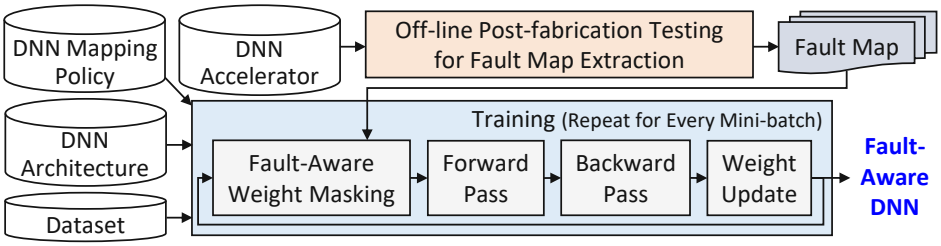


Figure 2.14: Generic flow for fault-aware (re-)training.

2.6.2 Aging Mitigation

Aging in CMOS devices manifests as timing errors. Similar to other types of errors, these errors can also significantly degrade the accuracy of DNNs as shown in [ZRGG18]. To prevent timing errors, traditionally voltage and frequency guardbands are incorporated while considering the worst-case aging effects [MSZ+11]. Adaptive techniques have been proposed to mitigate the performance degradation associated with guardbanding [MSZ+11][KKS09]; however, these techniques still result in high overheads. Moreover, all the above-highlighted techniques are *application agnostic* and do not benefit from the inherent resilience of DNNs to certain types of errors.

To mitigate timing errors in the computational array of a systolic-array-based DNN accelerator at low cost, Zhang et al. proposed TE-Drop [ZRGG18]. TE-Drop exploits resilience of DNNs to computation skipping. To detect timing errors in the array, it employs Razor flip-flops. Upon detection of a timing error in a MAC unit, instead of re-executing the erroneous MAC operation, it captures the correct output in an alternate partial-sum register operating on a delayed clock and then steals a cycle from the downstream PE by bypassing it to feed in the rectified output back in the pipeline. Fig. 2.15 presents the hardware modifications required to realize the concept. Pandey et al. proposed a different design, GreenTPU [PBCR19], to achieve the same. The GreenTPU mainly identifies error-causing activation patterns in the systolic array and prevents further timing errors from similar inputs by adaptively boosting the operating voltage of the specific MAC units that are expected to experience timing errors [PBCR19]. Note that although both the techniques, i.e., TE-Drop [ZRGG18] and GreenTPU [PBCR19], have been proposed to improve energy efficiency of DNN accelerators through voltage scaling, they can be exploited to mitigate aging as well.

Aside from aging in the computational array of DNN accelerators, NBTI aging in SRAM cells is also a serious concern. Various techniques have been proposed in the literature to address this issue. At the circuit level, structural modifications have been proposed to bring down the aging rate in SRAM cells [RSR+10][SZBP08]. For example, Ricketts et al. [RSR+10] proposed an asymmetric SRAM cell structure for workloads having biased bit distribution. However, due to high data dependence, this technique is effective only in specific scenarios. Recovery boosting through dedicated recovery-accelerating units is another method for reducing aging of SRAM cells [SG11]. However, the additional

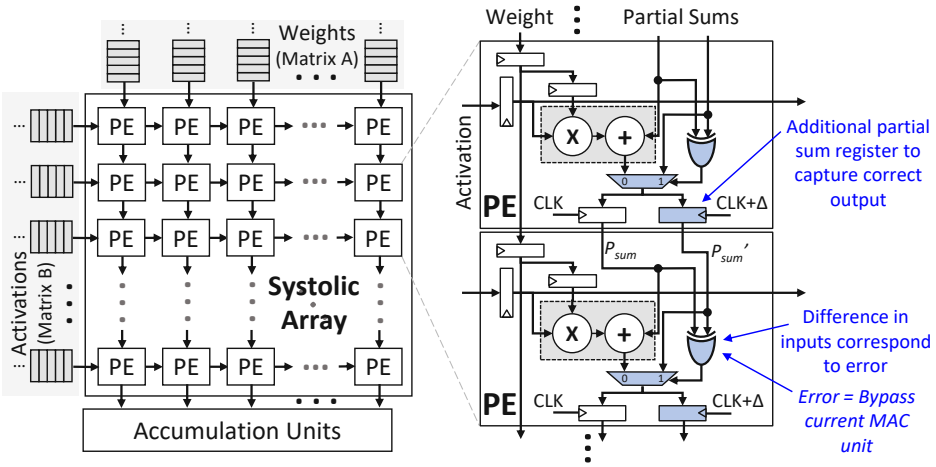


Figure 2.15: Architectural modifications required in PEs of a systolic-array-based DNN accelerator to realize TE-Drop

circuitry required in this increases the energy consumption of the system [ZSBH11]. At the architecture level, periodic inversion of data has been studied to reduce aging in on-chip SRAMs [JW12]. Although it is considered to be one of the most effective techniques, it cannot guarantee optimal aging mitigation, specifically in cases where the same data is reused periodically, e.g., in DNN-based systems. Calimera et al. [CLMP11] proposed a technique to boost recovery of unutilized memory cells. However, as the technique requires online monitoring support, it incurs high energy and area overheads. Other techniques employ circular shifts to mitigate NBTI aging [KCR11]. However, such techniques are effective only in cases where the overall distribution of bits is relatively balanced. Moreover, these techniques employ barrel shifters that result in high energy and area overheads. *In short, state-of-the-art techniques do not offer optimal level of aging mitigation for SRAM-based on-chip weight and activation memories of DNN accelerators. Therefore, novel methods are required to mitigate aging of SRAM cells used for DNN workloads.*

2.6.3 Soft Error Mitigation

Similar to other types of hardware-induced reliability threats, technology scaling results in a drastic increase in the soft error rate as well, i.e., chips fabricated using smaller technology nodes are more prone to soft errors. Various techniques have been proposed to address soft errors in DNN accelerators. Li et al. [LHS+17] studied the propagation of soft errors in DNNs during the inference stage using fault-injection experiments. The experiments indicate that using just-enough data precision can significantly reduce the impact of soft errors on the accuracy of DNNs. Based on similar observations, Chen et al. [Cea20] and Hoang et al. [HHS20b] propose to limit the activation values to pre-determined ranges, as critical faults typically result in abnormal activations with high

magnitude that can propagate to the output to cause misclassification. Moreover, Salami et al. [SUK18] propose to combine three individual mechanisms to achieve cost-effective application-aware fault mitigation. The three mechanisms are: (1) Word Masking, which sets all the bits of the corrupted register to '0'; (2) Bit Masking, which masks the faulty bit with the sign bit of the register; (3) Sign-bit Masking, which masks the sign bit with the MSB of the register. Note that Word Masking and Bit Masking are adopted from Minerva [RWA+16]. Both the works (i.e., [SUK18] and [RWA+16]) are based on the assumption that there is no limit on the number of faults that can be detected, and fault detection is achieved using Razor shadow latches, which result in only minor overhead cost [SUK18]. Apart from the above techniques, Zhao et al. [ZDL+20] proposed Algorithm-Based Fault Tolerance (ABFT) techniques to protect all types of convolution operations in DNN accelerators against soft errors using checksum techniques.

2.7 Chapter Summary

In this chapter, background knowledge related to deep learning, types of DNNs and available techniques for improving energy efficiency and reliability of DNN inference is discussed. In particular, the chapter covered the basics of deep learning, Multi-Layer Perceptrons (MLPs) and Convolutional Neural Networks (CNNs). Then, different techniques for improving the energy efficiency of DNN inference are discussed. Towards this, the chapter covered both software-level and hardware-level techniques. In software-level techniques, DNN pruning, quantization and Neural Architecture Search are covered in detail as these offer the best quality-efficiency trade-off for generating resource-friendly DNN models. In hardware-level techniques, DNN accelerators and hardware-level approximations are covered in detail. Given that DNNs are highly useful for safety-critical applications as well, the chapter also presented a brief overview of different hardware-induced reliability threats and their impact on the accuracy of DNNs. The chapter then highlighted the need for low-cost techniques for improving the reliability of DNN inference systems and covered the most prominent techniques designed for DNN accelerators.

CHAPTER 3

Analytical Models and Design Space Exploration of Approximate Modules

Approximate circuits exploit the error resilience of applications to trade-off quality for energy, area, and/or performance efficiency. Adders, being one of the fundamental operators in many processing applications, have received a significant amount of attention from the approximate computing community. Using low-power and low-latency approximate adders can have a significant impact on the energy efficiency and performance of a system. Towards improving the energy efficiency and performance of a system using approximate adders, first, this chapter presents a generic accuracy-configurable adder model for Low-Latency Approximate Adders (LLAAs) in Section 3.1. The model helps in analytically analyzing the structural properties of different LLAA configurations and identifying the optimal designs that offer the best quality-efficiency trade-offs. The analysis in Section 3.2 shows that, given a latency constraint, it is possible to effortlessly select the optimal LLAA configuration without involving any optimization strategy or numerical simulation. After the design space exploration of LLAAs, in Section 3.5, the chapter presents an analytical model for evaluating the error characteristics of Low-Power Approximate Adders (LPAAs) composed of cascaded approximate units. The analytical model is capable of covering the complete design space that can be constructed using the optimal LLAA configurations and approximate full-adder designs. This makes the proposed model highly effective for selecting the most suitable high-performance (low-latency) and low-power adder configuration for a given scenario.

Further, the chapter highlights that the data distribution of inputs can have a significant impact on the output quality of an approximate module. Therefore, in cases where input distribution is not uniform or input bits cannot be assumed independent of each other, a data-driven approach is necessary to accurately estimate the error characteristics of an

approximate module. Towards this, in Section 3.8, the chapter presents DAEM, a data and application-driven error estimation methodology. The evaluation shows that DAEM offers better error estimates compared to conventional methods in cases where input data distribution is not uniform. In the end, the chapter highlights some of the limitations of the proposed approaches and the importance of simulation-based error estimation and application-specific approximations.

3.1 Generic Accuracy-Configurable Adder Model for Low-Latency Adders

High-performance (low-latency) adders, such as fast/parallel-prefix adders, are widely used for applications that have strict latency and throughput constraints. Although these adders provide significant performance benefits, they lead to serious power and area overhead due to the presence of parallel carry generation logic. Coincidentally, most of the applications that involve an intense level of data processing are somewhat error resilient and, therefore, can benefit from the concepts of approximate computing to improve performance [VCRR15][SHR+16][XKM16][Mit16]. Towards this, several LLAA designs have been proposed, for example, ETA-II [ZGY09], ETA-IIM [ZGY09], ACA [VBI08][KK12], GDA [YWY+13], etc., that offer performance improvements beyond that of the conventional accurate adder designs. Each LLAA design has its own unique error and hardware characteristics and, therefore, performs better than other configurations under certain scenarios. Almost all LLAAs can be categorized as block-based adders, as they employ smaller sub-adders units that operate in parallel to generate the output bits. A few example LLAAs are shown in Fig. 3.1.

The availability of a vast variety of high-performance approximate adders makes it difficult for the designer to choose a suitable configuration for a particular application. To overcome this challenge, a unified model whose design space covers all/most of the LLAA configurations is necessary. The main advantage of such a model is that it can be used to build generic algorithms/methodologies for characterizing adder designs, which enables efficient design space exploration. In this context, this section first presents the *GeAr* adder model [SAHH15] and highlights its limitations. Then, the section introduces the proposed extended accuracy-configurable adder model *QuAd*, which overcomes all the limitations of the *GeAr* model and covers the complete design space of LLAAs.

3.1.1 GeAr Adder model

Let N be the length of the operands to be added. The *GeAr* adder makes use of k L-bit sub-adders that operate in parallel to compute the output. The length of the sub-adders is always less than or equal to N, i.e., $L <= N$. In *GeAr*, the sub-adders are mainly composed of two parts: (1) R-bits part, i.e., the resultant bits part, which generates *sum* bits for the adder's output; and (2) P-bits part, i.e., prediction bits part, which is used to predict the carry-in for the resultant part of the sub-adder. Each sub-adder contributes R-bits to the output, except for the first sub-adder, which contributes $L = R + P$ bits

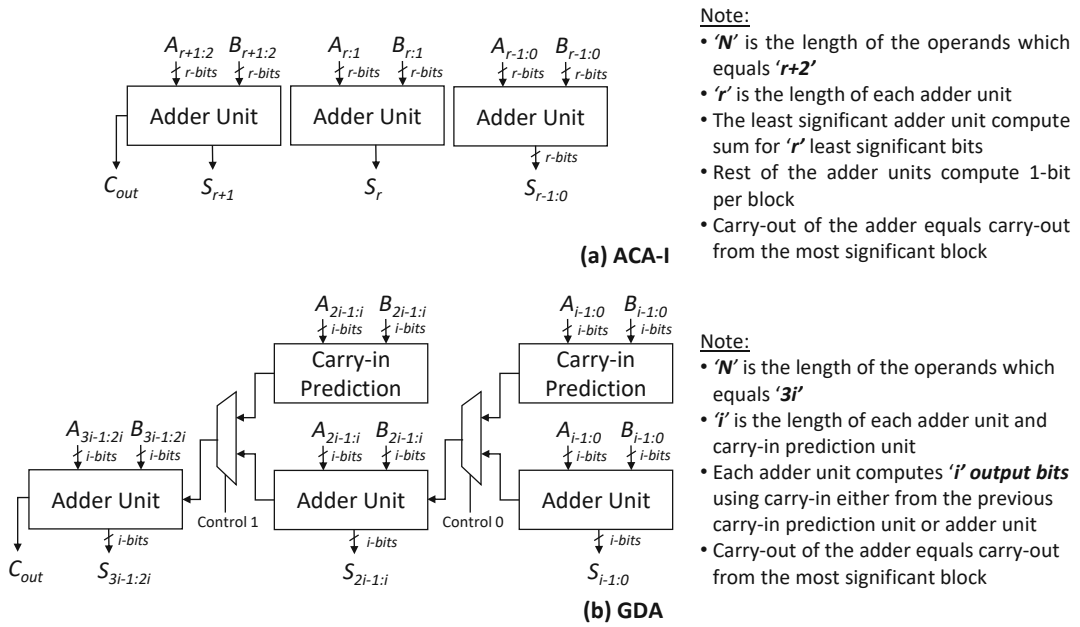


Figure 3.1: Example configurations of ACA-I (a) and GDA (b).

to the output. Based on the above description, a *GeAr* adder configuration can fully be described using three parameters, i.e., N, R, and P, and can be represented as $GeAr(N, R, P)$. The number of sub-adders (k) required for a given $GeAr(N, R, P)$ can be computed using $k = (N - P)/R$. Note, k has to be an integer for a configuration to be a valid *GeAr* configuration. A generalized architectural view of the *GeAr* adder model is presented in Fig. 3.2.

Example: To understand the functionality of the *GeAr* adder, consider $GeAr(12, 2, 6)$ configuration for adding two 12-bit numbers. Here, $N = 12$, $R = 2$ and $P = 6$. Using $k = (N - P)/R$, it can be observed that three sub-adders are required to compute the output of the adder, as shown in Fig. 3.3a. As illustrated in the figure, the first sub-adder (positioned at the least significant location) contributes eight bits (i.e, $L = R + P$ bits) to the output and all rest of the sub-adders contribute two bits (i.e., R bits) each, while using six previous bits (i.e., P bits) for predicting the carry-in to the resultant part. Note, the adder produces accurate output for all the cases where the maximum carry propagation length is less than P. An error occurs only in cases where all the P-bits of a sub-adder are in the propagate mode and the carry-out of the previous sub-adder is 1. For example, see Fig. 3.3c, which illustrates the scenario of sub-adder 2 in detail.

Limitations of the *GeAr* model: The *GeAr* model only supports configurations in which all the sub-adders have the same sub-adder length (i.e., L) and the same number of resultant (i.e., R) and prediction (i.e., P) bits. In short, adder configurations having sub-adders of different lengths and different R and P bits are not supported by the *GeAr* adder model.

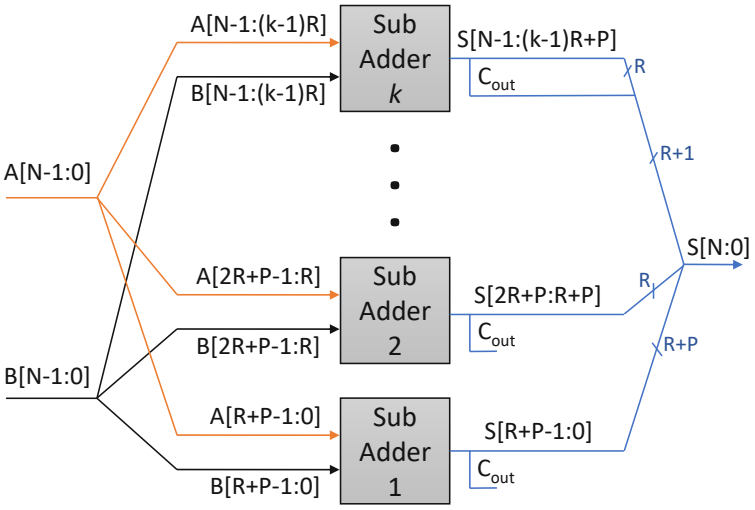


Figure 3.2: A generic N-bit GeAr adder composed of k sub-adders, where each sub-adder takes in $L = R + P$ number of bits from both the operands to generate respective R number of output bits. except for the first sub-adder which computes the output for $L = R + P$ number of bits. Also, the c_{out} of the most significant sub-adder is used as the carry-out of the adder.

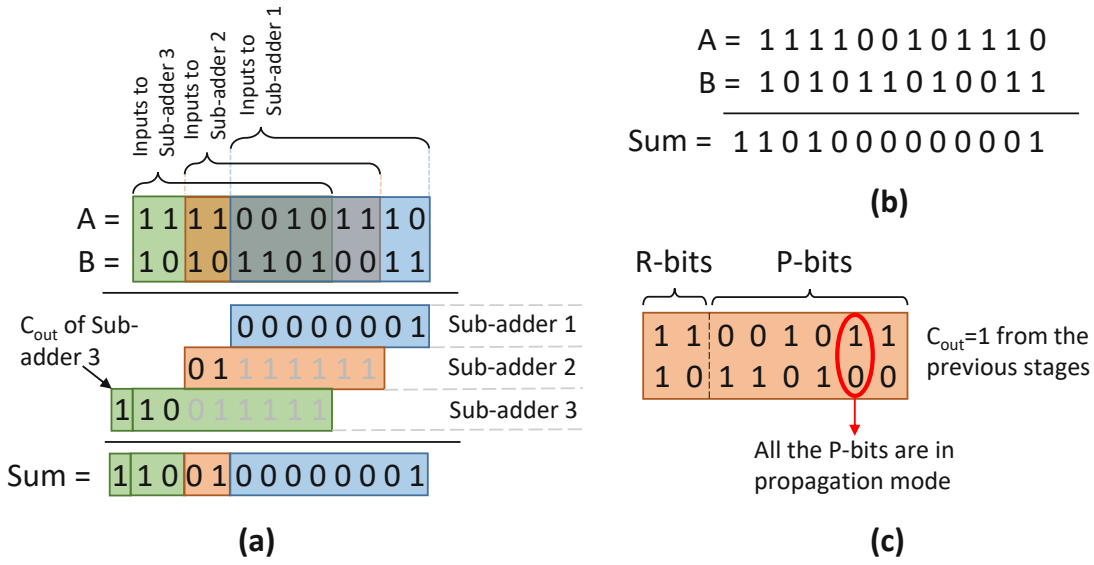


Figure 3.3: An example illustrating the functionality of low-latency adders. (a) Addition of two 12-bit operands using $GeAr(12, 2, 6)$ configuration. (b) Accurate addition of the example operands. (c) Example case of sub-adder 2 which leads to error in the output bits.

3.1.2 QuAd Adder model

The restrictions imposed on the length of the sub-adders and the number of resultant and prediction bits in the *GeAr* adder model lead to a limited design space. Therefore, to cover the complete design space of LLAAs, an extended accuracy-configurable adder model is proposed in this work, i.e., *QuAd*. Along with the configurations supported by the *GeAr* adder model, *QuAd* also supports the configurations in which sub-adders have different lengths and different number of resultant (R) and prediction (P) bits. Therefore, to define an N-bit *QuAd* adder configuration, a resultant bit vector, i.e., $R_{vect} = \{R_1, R_2, R_3, ..., R_k\}$, and a prediction bit vector, i.e., $P_{vect} = \{P_1, P_2, P_3, ..., P_k\}$, are required. Here, R_i and P_i define the number of resultant and prediction bits in the i^{th} sub-adder, respectively, and k represents the total number of sub-adders. The sum bits from the resultant parts of all the sub-adders and the carry-out from the last (k^{th}) sub-adder are concatenated to generate the $N+1$-bit output. Note, P_1 is always equal to zero, as the first sub-adder is responsible for generating the sum bits for all the least significant locations covered by the first sub-adder. Based on the above description, a generic *QuAd* configuration can be defined as $QuAd\{[R_1, R_2, ..., R_k], [0, P_2, P_3, ..., P_k]\}$. Fig. 3.4 presents a generic representation of an N-bit *QuAd* adder.

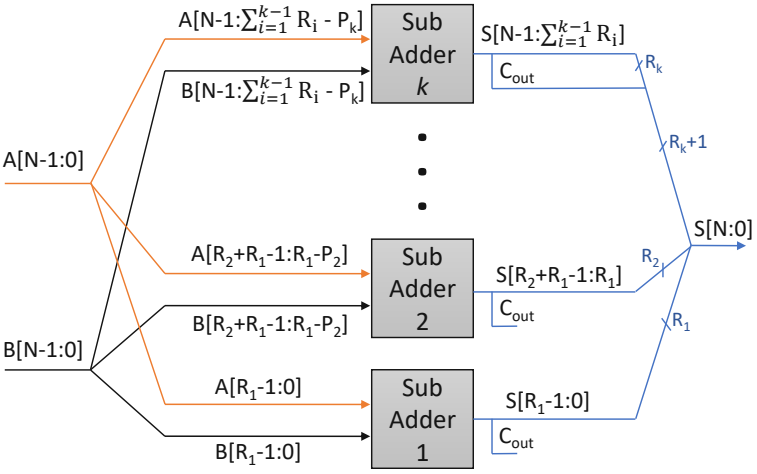


Figure 3.4: A generic N-bit *QuAd* adder composed of k sub-adders, where each i^{th} sub-adder sums two $R_i + P_i$ number of bits to generate R_i number of output bits, except for the last sub-adder which contributes $R_i + 1$ number of bits.

3.2 Design Space Exploration of Low-latency Adders for Uniformly Distributed Inputs

As highlighted in Section 3.1.2, the *QuAd* adder model significantly increases the overall design space of the low-latency adders. This section presents a systematic process for exploring the extended design space of LLAAs and reaching the final set of optimal

configurations. In the first step, the process restricts the design space of $QuAd$ to configurations that satisfy $P_i < P_{i-1} + R_{i-1}$ constraint for all $i \in 2, 3, ..., k$. Then, based on a set of properties that compare error and structural characteristics of different configurations, the process proposes that, provided a latency constraint (L_{max}), a quality-area optimal configuration ($QuAd_o$) can effortlessly be selected from the remaining design space. A flow of the steps involved in design space exploration is presented in Fig. 3.5.

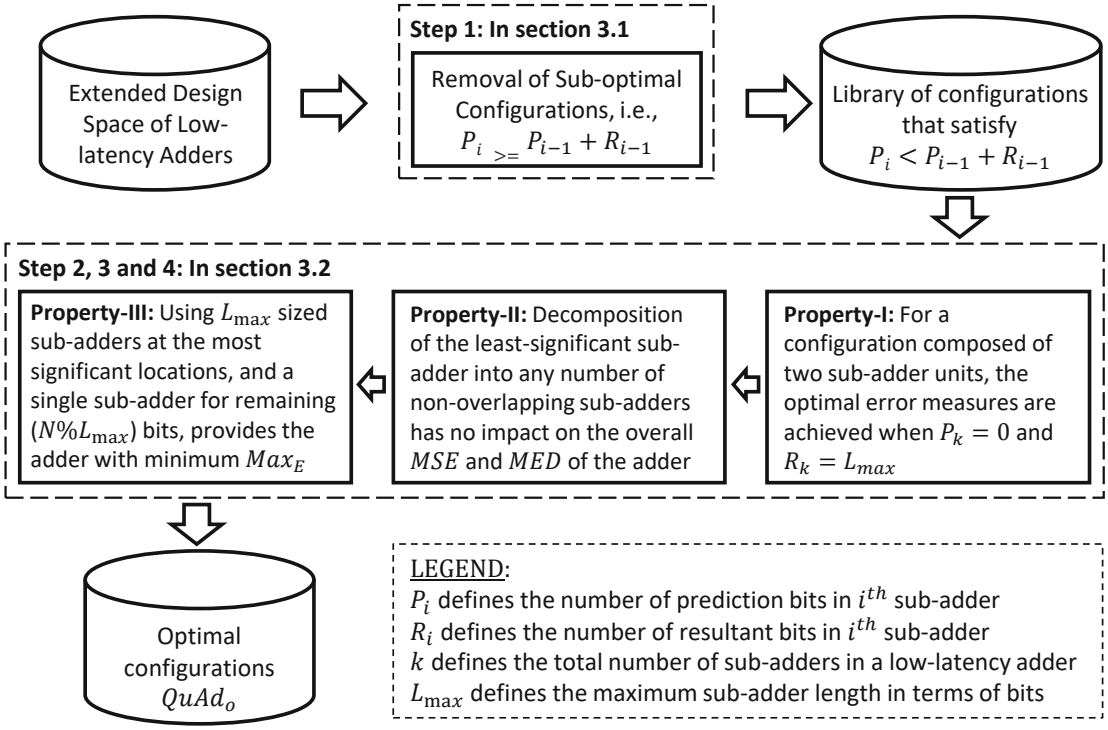


Figure 3.5: Process for exploring the design space of LLAAs covered by $QuAd$ adder model.

3.2.1 Early Design Space Reduction

This section shows that for each configurations having $P_i >= R_{i-1} + P_{i-1}$ ($\forall\ i \in 2, 3, ..., k$) there exists a configuration that satisfies $P_j < R_{j-1} + P_{j-1}$ ($\forall\ j \in 2, 3, ..., k$) and provides better/same output quality while consuming lesser resources. To illustrate the above, this section presents a comparison between the Probability Mass Function (PMF)[1] of error of three possible configuration types: (1) $P_i = R_{i-1} + P_{i-1}$; (2) $P_i < R_{i-1} + P_{i-1}$; and (3) $P_i > R_{i-1} + P_{i-1}$.

[1]A Probability Mass Function (PMF) defines the probability distribution of a discrete random variable. It defines the probability of each possible value of the random variable. For example, for a discrete random variable X, $P(X = x)$ defines the probability of X being x, where $x \in Z$.

1) $P_i = R_{i-1} + P_{i-1}$: Fig. 3.6a shows an example configuration for the case $P_i = R_{i-1} + P_{i-1}$. As can be seen in the figure, the condition $P_i = R_{i-1} + P_{i-1}$ is true for $i = 3$. The corresponding PMF of error of the configuration is shown on the right of Fig. 3.6a. The PMF of error of an approximate adder defines all the possible error values along with their occurrence probabilities. Note, the PMF of error of an approximate module provides a better understanding of the output quality of the module, as it can be used to compute most of the error metrics. The PMF of error can formally be defined as $P(E = e_w)$, where e_w corresponds to the error magnitude that can have any value between 0 and $2^{N+1} - 1$. As can be seen in the figure, the example configuration results in an error of $2^{R_1} = 2^4$ for some input combinations and no error for the rest of the combinations. The error occurs only when the input bits corresponding to the location of P_2-bits are in carry propagation mode and a carry-out is generated by the least significant $R_1 - P_2$ bits. Ideally, in the case the adder is accurate, the generated carry would propagate to the output; however, because of the truncated carry-chain in the example configuration, the carry results in an error of 2^{R_1} magnitude.

2) $P_i < R_{i-1} + P_{i-1}$: To compare the case of $P_i = R_{i-1} + P_{i-1}$ with that of $P_i < R_{i-1} + P_{i-1}$, an example configuration corresponding to the case $P_i < R_{i-1} + P_{i-1} \forall i \in \{2, 3, ..., k\}$ is presented in Fig. 3.6c. The configuration is selected such that the number of carry prediction bits used for each sum bit is the same as that of the configuration presented in Fig. 3.6a. Thus, the error distribution of both configurations is the same. However, as the number of sub-adders in $QuAd\{[4, 4], [0, 2]\}$ are less than the number of sub-adders in $QuAd\{[4, 2, 2], [0, 2, 4]\}$, the total resources consumed by $QuAd\{[4, 4], [0, 2]\}$ are also less. Therefore, Fig. 3.6c provides a resource-efficient substitute for the configuration shown in Fig. 3.6a.

3) $P_i > R_{i-1} + P_{i-1}$: Fig. 3.6b shows an example configuration which satisfies $P_i > R_{i-1} + P_{i-1}$ at least for one sub-adder. As can be seen from the figure, the prediction bits of the third sub-adder are extended even beyond the length of the second sub-adder, i.e., $P_3 > R_2 + P_2$. In this particular case, the PMF of error shows that a new error term with a magnitude of $2^{R_1+R_2} - 2^{R_1}$ is introduced in the PMF, along with the increased probability of error of 2^{R_1}. Therefore, it can be concluded that the approximation error, in this case, is significantly higher than the cases presented in Figs. 3.6a and 3.6c. Note, this increase in error is mainly due to the fact that the adder is using a lesser number of bits for predicting the carry-in for R_2-bits.

In summary, it can be concluded that:

- the configurations that satisfy the $P_i = R_{i-1} + P_{i-1}$ constraint (for any $i \in \{2, 3, ..., k\}$) provide better accuracy than similar corresponding configurations with $P_i > R_{i-1} + P_{i-1}$ (for any $i \in \{2, 3, ..., k\}$); and
- the configurations that satisfy the $P_i < R_{i-1} + P_{i-1}$ constraint ($\forall i \in \{2, 3, ..., k\}$) offer the same accuracy as compared to the similar corresponding configurations

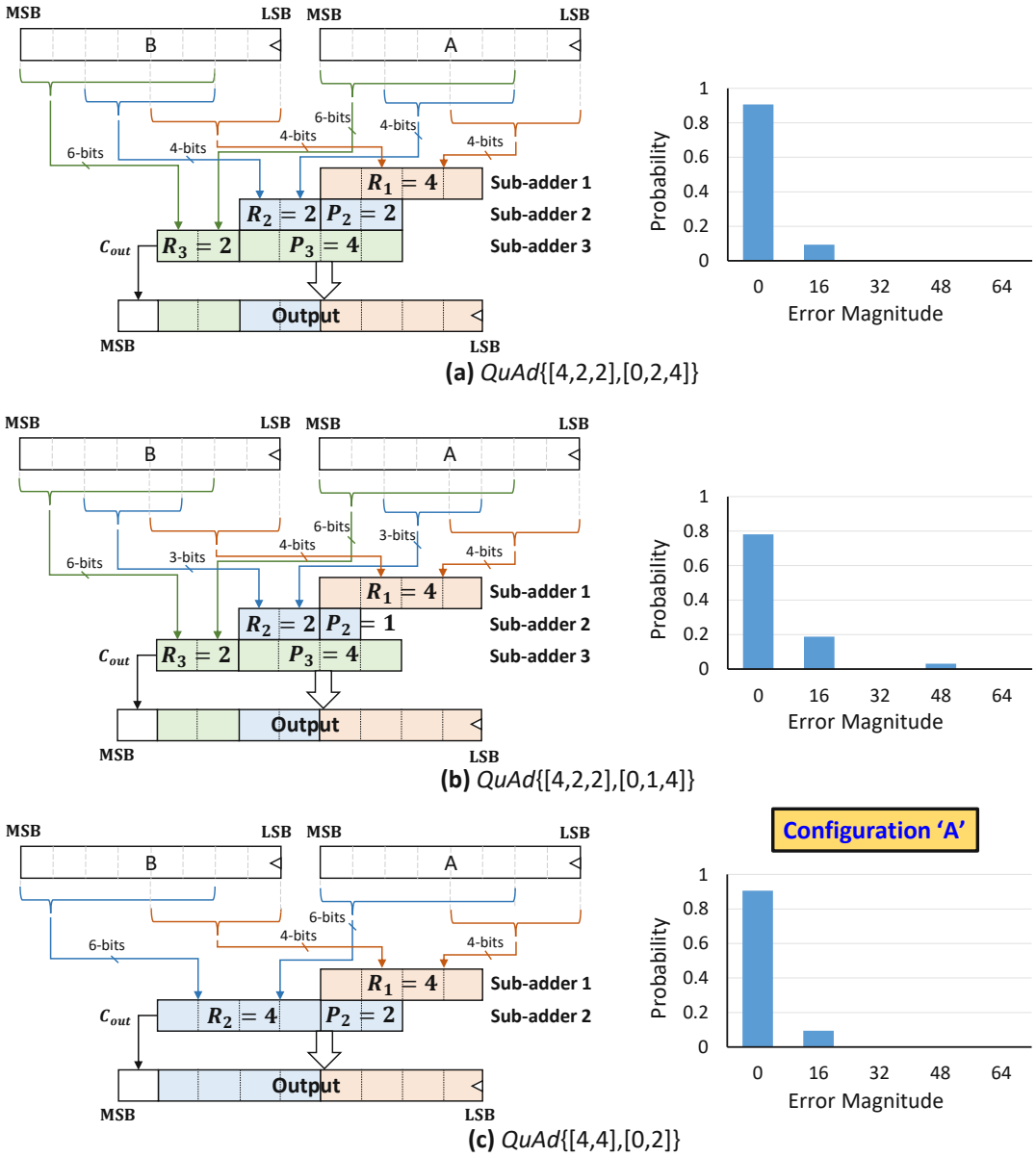


Figure 3.6: Three different *QuAd* configurations along with their respective PMFs of error. (a) $P_i = P_{i-1} + R_{i-1}$, (b) $P_i > P_{i-1} + R_{i-1}$ and (c) $P_i < P_{i-1} + R_{i-1}$.

with $P_i = R_{i-1} + P_{i-1}$ (for any $i \in \{2, 3, ..., k\}$), while requiring lesser amount of area/energy resources.

In short, for configurations that satisfy the $P_i >= R_{i-1} + P_{i-1}$ constraint (for any $i \in \{2, 3, ..., k\}$), there exists a substitute configuration with $P_i < R_{i-1} + P_{i-1}$ (for all $i \in \{2, 3, ..., k\}$) that offers the same/better output quality while consuming lesser resources. Therefore, in $QuAd$, P_i is always less than $R_{i-1} + P_{i-1}$ for all sub-adder units.

3.2.2 Quality-Area Optimal ($QuAd_o$) Configurations

Let N be the length of the operands and L_{max} be the latency constraint in terms of sub-adder length. Then, the term $QuAd_o\{N, L_{max}\}$ defines the quality-area optimal LLAA configuration. The adder makes use of $k = \lceil N/L_{max} \rceil$ number of non-overlapping sub-adders to compute the output, where $k-1$ most significant sub-adders are of length L_{max} and the least significant sub-adder has the length equivalent to the remaining number of bits, i.e., $N\%L_{max}$. To demonstrate the superiority of the $QuAd_o$ configurations over rest of the $QuAd$ configurations, three different error metrics are considered in this work, i.e., Mean Error Distance (MED) [LHL13], Mean Square Error (MSE) [GW08] and Maximum Error Magnitude (Max_E). The following text presents three properties along with their proofs to demonstrate the optimality of $QuAd_o$ configurations.

Property-I: The configurations having the least number of P-bits and the maximum possible length for the most significant sub-adder offers the lowest MSE and MED.

Consider the adder configuration shown in Fig. 3.6c that satisfies the $P_i < R_{i-1} + P_{i-1}$ constraint for all $i \in \{2, 3, ..., k\}$. The configuration is composed of two sub-adders and, therefore, has only one error term, i.e., $E_A = 2^{R_1}$, equivalent to the carry-out of sub-adder 1. The corresponding probability of error, i.e., $P[E]_A$, can be defined as the probability with which the least significant $(R_1 - P_2)$ bits generate a carry while the P_2-bits are in propagate mode. Assuming the input bits to be independent of each other and uniformly distributed, the probability of error can mathematically be represented as:

$$P[E]_A = \rho[pr]^{P_2} \times \sum_{i=0}^{R_1-P_2-1} \rho[gr] \times \rho[pr]^i \tag{3.1}$$

Here, $\rho[gr] = \rho[(a_i == 1)\&\&(b_i == 1)]$ defines the probability of carry generation and $\rho[pr] = \rho[((a_i == 1)\&\&(b_i == 0)) \,||\, ((a_i == 0)\&\&(b_i == 1))]$ defines the probability of carry propagation. Moreover, a_i and b_i represent the i^{th} bit of operands A and B, respectively.

Provided the error magnitude and the corresponding error probability, the MED and MSE of the configuration can be written as:

$$
\begin{aligned}
MED_A &= P[E]_A \times E_A \\
&= 2^{R_1} \times \rho[pr]^{P_2} \times \sum_{i=0}^{R_1-P_2-1} \rho[gr] \times \rho[pr]^i
\end{aligned} \tag{3.2}
$$

$$
\begin{aligned}
MSE_A &= P[E]_A \times (E_A)^2 \\
&= 2^{2R_1} \times \rho[pr]^{P_2} \times \sum_{i=0}^{R_1-P_2-1} \rho[gr] \times \rho[pr]^i
\end{aligned} \tag{3.3}
$$

Now, if the configuration in Fig. 3.6c is modified such that the length of the most significant sub-adder remains the same, while the number of overlapping bits between the two sub-adders is reduced by one bit, which results in configuration 'B' shown in Fig. 3.7. The corresponding error magnitude and probability of error of configuration 'B' in terms of the parameters of configuration 'A' can be written as:

$$
\begin{aligned}
P[E]_B &= \rho[pr]^{P_2-1} \times \sum_{i=0}^{R_1-P_2-1} \rho[gr] \times \rho[pr]^i \\
E_B &= 2^{R_1-1}
\end{aligned} \tag{3.4}
$$

As can be seen from Eqs. 3.1 and 3.4, the decrease in the number of prediction bits of the most significant sub-adder while maintaining its overall length results in an increase in the error probability and a decrease in the error magnitude. The MSE and MED of configuration 'B' in terms of the parameters of configuration 'A' can be written as:

$$
\begin{aligned}
MED_B &= 2^{R_1-1} \times \rho[pr]^{P_2-1} \times \sum_{i=0}^{R_1-P_2-1} \rho[gr] \times \rho[pr]^i \\
&= MED_A/(2 \times \rho[pr]) \\
MSE_B &= 2^{2R_1-2} \times \rho[pr]^{P_2-1} \times \sum_{i=0}^{R_1-P_2-1} \rho[gr] \times \rho[pr]^i \\
&= 1/2 \times (MSE_A/(2 \times \rho[pr]))
\end{aligned}
$$

Assuming inputs to be uniformly distributed, $2 \times \rho[pr]$ equals 1, as $\rho[pr] = 0.5$. Therefore, it can be said that MED of configuration 'B' is equivalent to that of the MED of configuration 'A'. However, the MSE of configuration 'B' is half of the MSE of configuration 'A'. Similar to the above case, if the number of overlapping bits between the two sub-adders of configuration 'B' are decreased further while keeping the length of the most significant sub-adder the same, further decrease in MSE can be achieved. Therefore, it can be concluded that the configuration with no prediction bits in the most significant

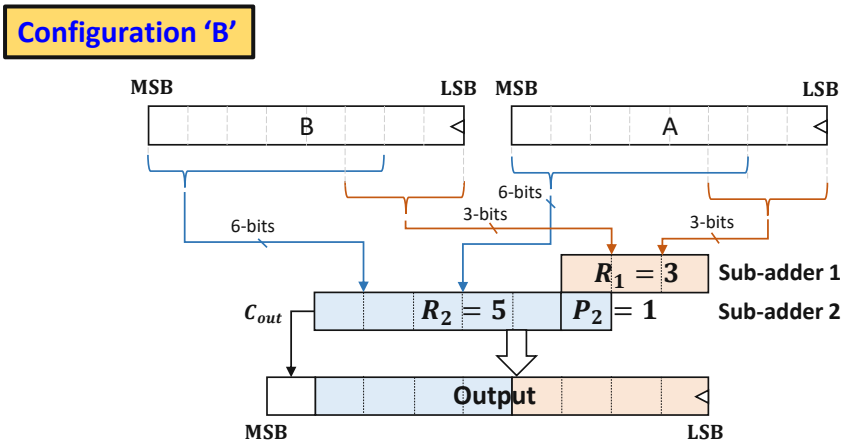


Figure 3.7: An illustrative view of $QuAd\{[3,5],[0,1]\}$, where the length of the most significant sub-adder is the same as the length of most significant sub-adder of configuration 'A' and the overlap between the sub-adders is 1-bit, i.e., $P_k = 1$.

sub-adder, i.e., $P_k = 0$, while having maximum possible length, i.e., $R_k = L_{max}$ provides optimal MSE and MED while consuming minimum area resources.

Property-II: In the case of uniformly distributed inputs, the configuration of the lower part ($N - L_{max}$ bits) of the adder does not affect the MSE and MED values.

Property-I demonstrated that in a two sub-adder configuration, the minimum MSE and MED values are achieved when there is no overlap between the sub-adders and $R_k = L_{max}$. This property, with the help of configurations 'C' and 'D' in Fig. 3.8, shows that the decomposition of the least significant sub-adder into any number of non-overlapping sub-adders does not impact the overall MSE and MED of the adder configuration.

The probability of error ($P[E]$), the error magnitude (E), and the respective MSE and MED of configuration 'C' can mathematically be represented as:

$$\begin{aligned} P[E]_C &= \sum_{i=0}^{R_1-1} \rho[gr] \times \rho[pr]^i \\ E_C &= 2^{R_1} \\ MED_C &= |E_C| \times P[E]_C = 2^{R_1} \times \sum_{i=0}^{R_1-1} \rho[gr] \times \rho[pr]^i \\ MSE_C &= E_C^2 \times P[E]_C = 2^{2R_1} \times \sum_{i=0}^{R_1-1} \rho[gr] \times \rho[pr]^i \end{aligned} \tag{3.5}$$

As $\rho[gr] = \frac{1}{2^2}$ and $\rho[pr] = \frac{1}{2}$ for inputs having uniform distribution, the $P[E]_C$ can be simplified to:

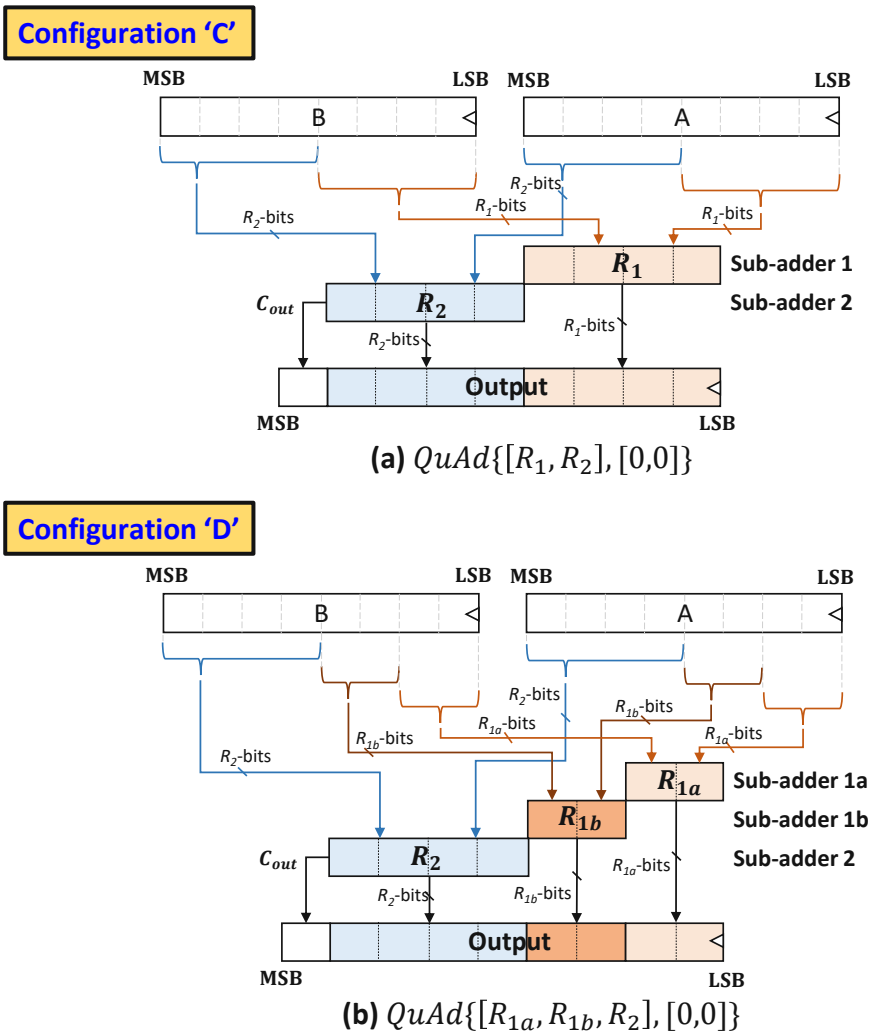


Figure 3.8: Structural comparison of two low-latency approximate adders composed of disjoint sub-adder units.

Table 3.1: Error cases of configuration 'D' along with their respective error probabilities and magnitudes.

Error Case	**Error Probability**	**Error Magnitude**
Carry-out only from R_{1a}	$\rho_{1a} - \rho_{1a} \times \rho_{1b}$	$2^{R_{1a}}$
Carry-out only from R_{1b}	$\rho_{1b} - \rho_{1a} \times \rho_{1b}$	$2^{R_{1a}+R_{1b}}$
Carry-out from both R_{1a} and R_{1b}	$\rho_{1a} \times \rho_{1b}$	$2^{R_{1a}} + 2^{R_{1a}+R_{1b}}$

$$P[E]_C = \frac{1}{2^2}(1 + \frac{1}{2} + ... + \frac{1}{2^{R_1-1}}) = \frac{1}{2^2}(2 - \frac{1}{2^{R_1-1}}) = \frac{2^{R_1}-1}{2^{R_1+1}} \tag{3.6}$$

Using Eqs. 3.5 and 3.6, MED_C and MSE_C can be simplified to:

$$\begin{aligned} MED_C &= \frac{2^{R_1}-1}{2} \\ MSE_C &= \frac{2^{R_1}(2^{R_1}-1)}{2} \end{aligned} \tag{3.7}$$

To understand the impact of dividing the lower sub-adder of configuration 'C' into parts on the MED and MSE of the configuration, consider configuration 'D' (see Fig. 3.8b) in which sub-adder 1a and 1b are non-overlapping sub-adders and have a cumulative length equivalent to the length of sub-adder 1 of configuration 'C', i.e., $R_1 = R_{1a} + R_{1b}$. The error in configuration 'D' occurs whenever there is a carry-out from R_{1a} and/or R_{1b}. The probabilities of carry-out from sub-adders 1a and 1b (i.e., ρ_{1a} and ρ_{1b}, respectively) can be written as:

$$\begin{aligned} \rho_{1a} &= \sum_{i=0}^{R_{1a}-1} \rho[gr] \times \rho[pr]^i &= \frac{2^{R_{1a}}-1}{2^{R_{1a}+1}} \\ \rho_{1b} &= \sum_{i=0}^{R_{1b}-1} \rho[gr] \times \rho[pr]^i &= \frac{2^{R_{1b}}-1}{2^{R_{1b}+1}} \end{aligned} \tag{3.8}$$

The three possible error cases of configuration 'D' along with their error probabilities and magnitudes are listed in Table 3.1.

Considering the error cases in Table 3.1, the MED of configuration 'D' can be written as:

$$\begin{aligned} MED_D = (\rho_{1a} - \rho_{1a} \times \rho_{1b}) \times 2^{R_{1a}} + (\rho_{1b} - \rho_{1a} \times \rho_{1b}) \times 2^{R_{1a}+R_{1b}} + \\ (\rho_{1a} \times \rho_{1b}) \times (2^{R_{1a}} + 2^{R_{1b}+R_{1b}}) \end{aligned} \tag{3.9}$$

Simplifying the above equation leads to:

$$MED_D = \rho_{1a} \times 2^{R_{1a}} + \rho_{1b} \times 2^{R_{1a}+R_{1b}} \tag{3.10}$$

Inserting the values of ρ_{1a} and ρ_{1b} from Eq. 3.8 leads to:

$$MED_D = \frac{2^{R_{1a}} - 1}{2} + \frac{2^{R_{1a}+R_{1b}} - 2^{R_{1a}}}{2} = \frac{2^{R_{1a}+R_{1b}} - 1}{2}$$

which is equivalent to MED_C in Eq. 3.7, as $R_1 = R_{1a} + R_{1b}$. Similarly, using the information in Table 3.1, the MSE of configuration 'D' can be written as:

$$MSE_D = (\rho_{1a} - \rho_{1a} \times \rho_{1b}) \times 2^{2R_{1a}} \times (\rho_{1b} - \rho_{1a} \times \rho_{1b}) \times 2^{2(R_{1a}+R_{1b})} + (\rho_{1a} \times \rho_{1b}) \times (2^{R_{1a}} + 2^{R_{1a}+R_{1b}})^2 \tag{3.11}$$

Simplifying the above equation leads to:

$$MSE_D = \rho_{1b} \times 2^{2(R_{1a}+R_{1b})} + \rho_{1a} \times 2^{2R_{1a}} + (\rho_{1a} \times \rho_{1b}) \times (2 \times 2^{R_{1a}} \times 2^{R_{1a}+R_{1b}})$$

Inserting values of ρ_{1a} and ρ_{1b} from Eq. 3.8 leads to:

$$MSE_D = \frac{2^{2R_{1a}+R_{1b}}(2^{R_{1b}} - 1) + 2^{R_{1a}}(2^{R_{1a}} - 1) + 2^{R_{1a}}(2^{R_{1a}} - 1)(2^{R_{1b}} - 1)}{2}$$

Further simplification of the above equation leads to:

$$MSE_D = \frac{2^{R_{1a}+R_{1b}}(2^{R_{1a}+R_{1b}} - 1)}{2}$$

which is equivalent to MSE_C in Eq. 3.7, as $R_1 = R_{1a} + R_{1b}$. Therefore, based on the above mathematical analysis, it can be concluded that the configurations in Fig. 3.8 have equal MSE and MED values.

Property-III: For a configuration composed of disjoint sub-adders, the minimum Max_E is achieved when the least possible number of sub-adders are used and the sub-adders at the most significant locations are of the maximum possible length, i.e., L_{max}.

In a configuration composed of disjoint (non-overlapping) sub-adders, the maximum error value, i.e., Max_E, is always the sum of the carry-outs from $k-1$ least significant sub-adders. Therefore, Max_E can mathematically be written as:

$$Max_E = \sum_{i=1}^{k-1} 2^{\sum_{j=1}^{i} R_j}$$

From the above equation, it can be inferred that, to minimize Max_E, it is important to minimize k (i.e., the number of sub-adders) and the value of carry-out from each sub-adder at the lower significance location than the most significant sub-adder. Therefore, a configuration having all the sub-adders (except the least significant sub-adder, in case $N\%L_{max}$ is not zero) equivalent to the maximum possible sub-adder length results in the lowest Max_E value, where all the sub-adders are disjoint. Hence, the quality-area optimal adder can be defined as:

$$QuAd_o(N, L_{max}) = QuAd\{[(N\%L_{max}), L_{max}, ..., L_{max}], [0, ..., 0]\} \tag{3.12}$$

3.3 Design Space Coverage and Performance of *QuAd*

This section presents a comparison between the design space of the *QuAd* and the existing LLAAs. The section also presents experimental results showing that the $QuAd_o$ configurations indeed offer the best quality-efficiency trade-offs. The area results are obtained by synthesizing Verilog implementation of the configurations using Xiline ISE for XILINX Virtex 6 XC6VLX75T FPGA. Note, although the results are presented for FPGA only, the *QuAd* adder model is not specific to FPGA and is equally valid for ASICs as well, where the sub-adders can be realized using any existing type of adders, e.g., Kogge-Stone Adder (KSA), etc.

3.3.1 Design Space Coverage and Exploration

To demonstrate the design space coverage of *QuAd*, without any loss of generality, the design space of 8-bit *QuAd* configurations is plotted against the design space of existing 8-bit LLAAs in Figs. 3.9 and 3.10. The design space of exiting LLAAs mainly includes GeAr [SAHH15], ACA [VBI08][KK12], ETA [ZGY09] and GDA [YWY+13]. The figures clearly show that *QuAd* not only covers all the state-of-the-art adder configurations but also span configurations that are not covered by the existing LLAAs. Moreover, the figures also highlight that *QuAd* uncovers configurations that offer better quality-efficiency trade-offs than existing adders. This is illustrated for each value of L_{max}. Note that the MSE and MED results presented here are generated using exhaustive simulations assuming uniform input distribution.

In line with the analysis in Section 3.2.2, Figs. 3.9 and 3.10 show that more than one *QuAd* configurations offer optimal MED-area and MSE-area trade-offs. Therefore, to highlight the significance of $QuAd_o$ configurations over the rest of the possible configurations that offer optimal trade-offs, the maximum error magnitude (Max_E) of all optimal configurations from Figs. 3.9 and 3.10 is plotted in Fig. 3.11. The figure clearly shows

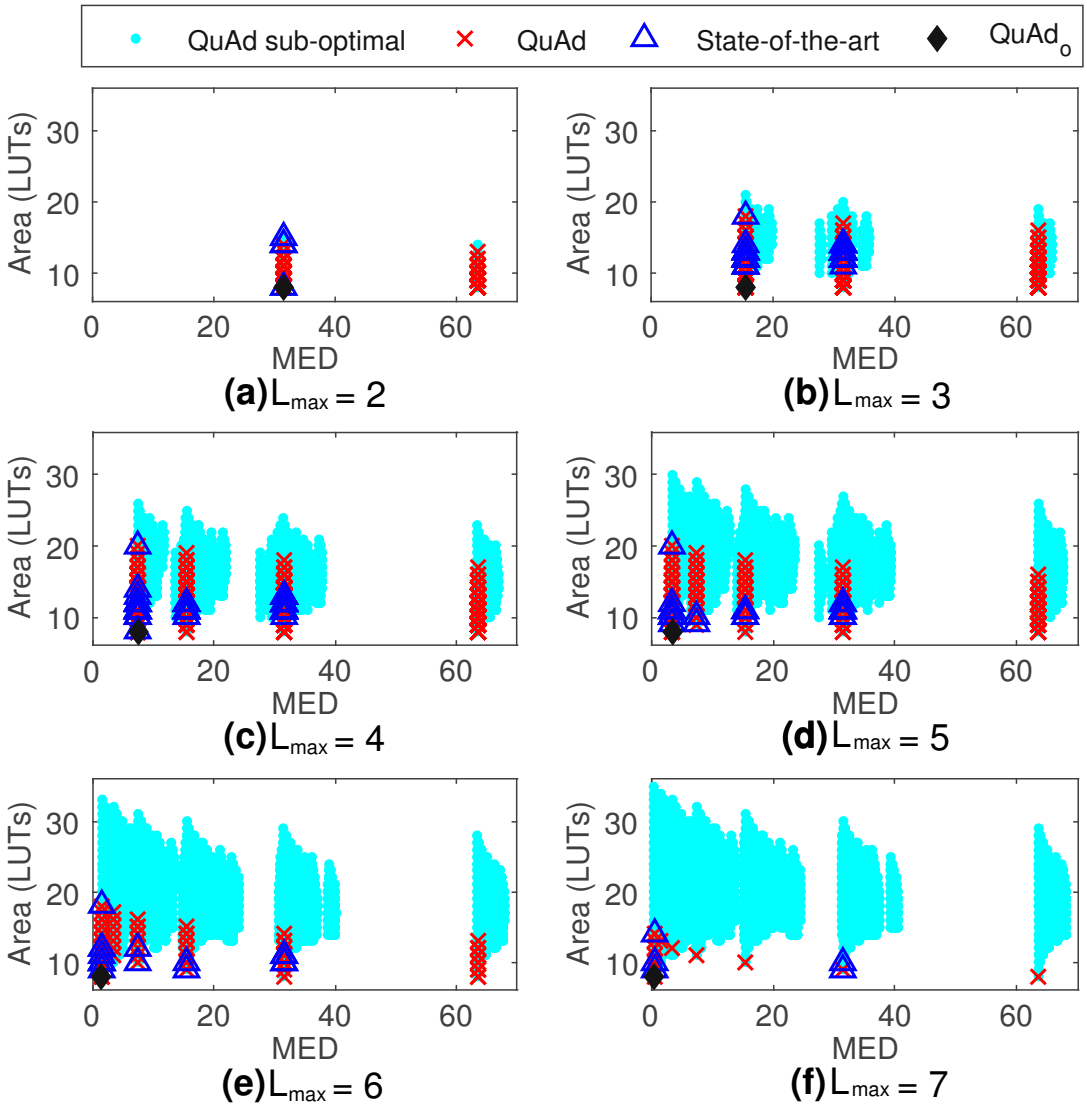


Figure 3.9: Design space of 8-bit low-latency adder for various L_{max} using MED error measure. The plot for $L_{max} = 1$ is not shown as it contains only one configuration with 8 sub-adders having $R - bits = 1$ and $P - bits = 0$.

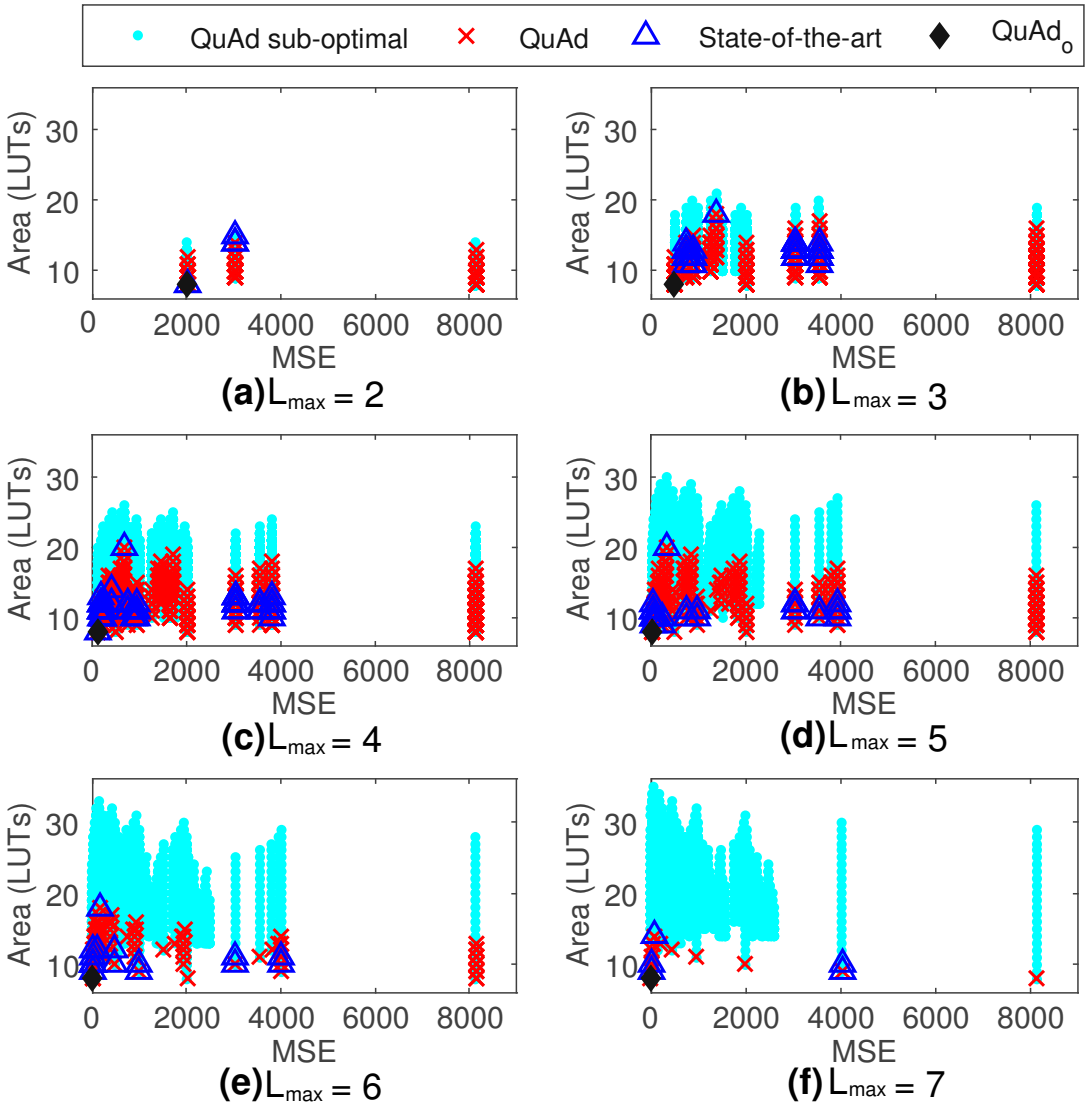


Figure 3.10: Design space of 8-bit low-latency adder for various L_{max} using MSE error measure. The plot for $L_{max} = 1$ is not shown as it contains only one configuration with 8 sub-adders each having $R - bits = 1$ and $P - bits = 0$.

that the $QuAd_o$ configurations provide minimum Max_E for each L_{max} and, therefore, should be considered optimal in terms of MSE, MED, and Max_E error measures.

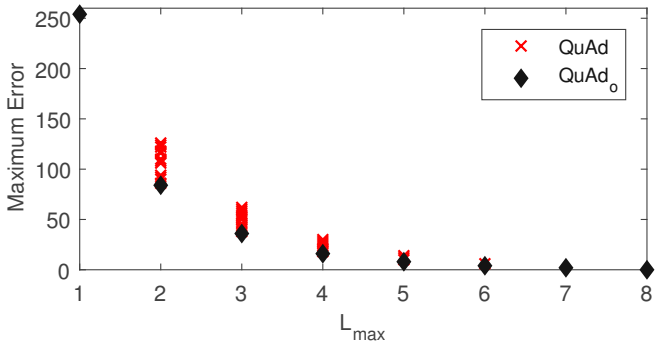


Figure 3.11: Max_E of 8-bit adder configurations that provide optimal MED and MSE results while consuming minimum area.

3.3.2 Performance in Real-World Applications

To demonstrate the effectiveness of $QuAd_o$ configurations for real-world applications, this section shows the results of low-pass image filtering and image blending applications when performed with $QuAd_o$ configurations. For image filtering, a 3x3 averaging kernel is assumed. The hardware accelerator is realized using several compression stages, as illustrated in Fig. 3.12. The compression is performed using full-adders and half-adders, while the final addition in the last stage is performed using an 8-bit approximate low-latency adder. The output of the adder is concatenated with the remaining least significant bits to get the overall sum. The sum is then divided by 9 to get the final output of the filter. The results for various configurations of LLAA are summarized in Figs. 3.13 and 3.14. As can be seen from the figures, for both cases, i.e., for $L_{max} = 4$ (Fig. 3.13) and $L_{max} = 6$ (Fig. 3.14), $QuAd_o$ configurations offer the best quality in terms of MSE as well as SSIM.

For image blending, element-wise addition of images is considered in this work. The addition is carried out using an 8-bit LLAA and the addition is followed by a division by 2 to generate the final output. Similar to the filtering application, various LLAAs are employed to study the impact of approximations on the output quality of the application. The results in Fig. 3.15 clearly show that $QuAd_o$ configuration offers optimal quality in terms of MSE and SSIM quality metrics.

To highlight the resource efficiency of the $QuAd_o$ configurations used in the above analysis, the area numbers of all the configurations are computed for XILINX Virtex 6 XC6VLX75T FPGA. The numbers are summarized in Table 3.2. The results clearly show that $QuAd_o$ configurations offer the best resource efficiency compared to the rest of the corresponding adder configurations. This, coupled with the conclusion from Figs. 3.13, 3.14 and 3.15,

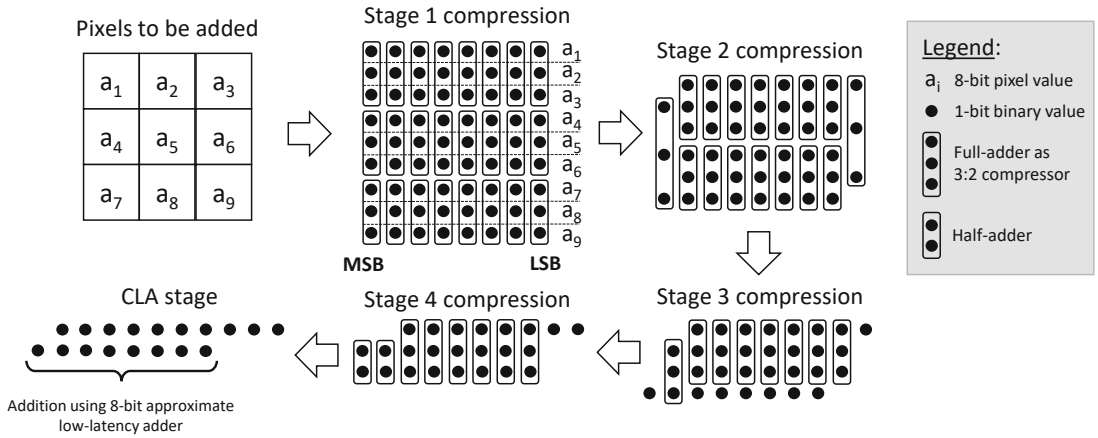


Figure 3.12: Image low-pass filtering accelerator detail.

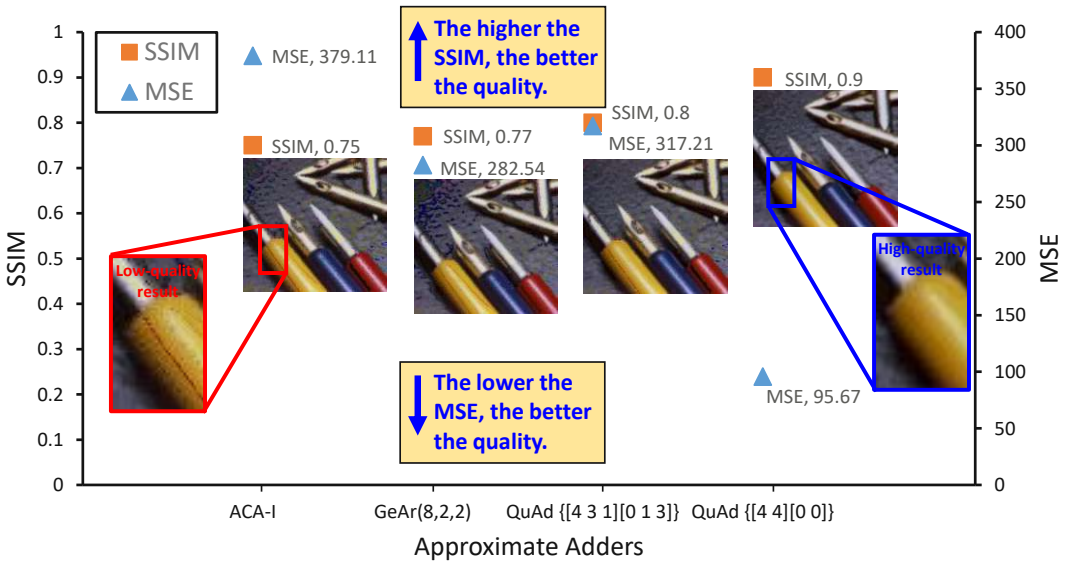


Figure 3.13: Image lowpass filtering results for various approximate low-latency adders with $L_{max} = 4$.

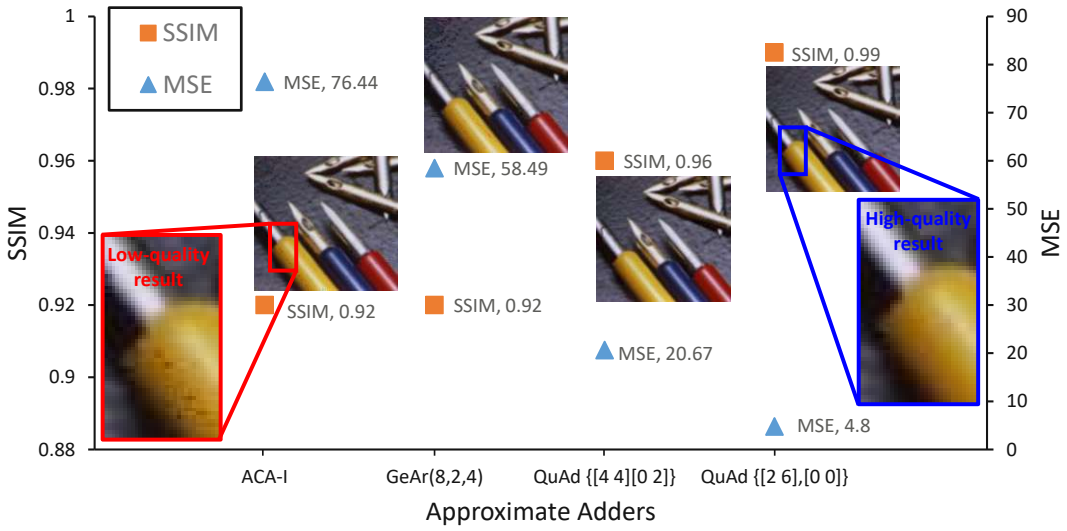


Figure 3.14: Image lowpass filtering results for various approximate low-latency adders with $L_{max} = 6$.

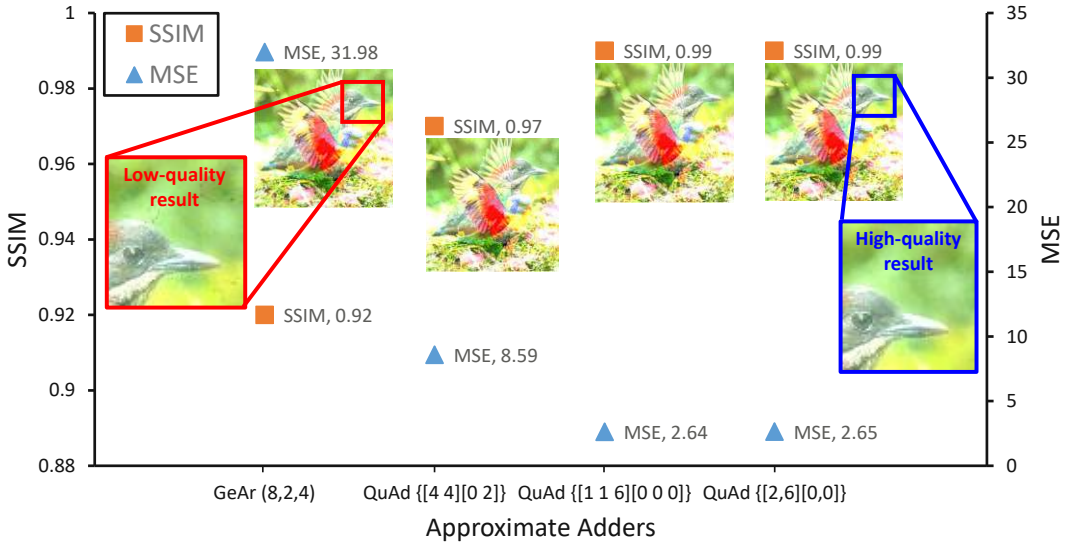


Figure 3.15: Image blending results for various approximate low-latency adders with $L_{max} = 6$.

Table 3.2: Area results for various low-latency approximate adders.

Adder Configuration	L_{max}	Area [LUTs]
ACA-I	6	18
$GeAr\{8,2,4\}$	6	12
$QuAd\{[4,4],[0,2]\}$	6	10
\$QuAd{[1,1,6],[0,0,0]}	6	8
$QuAd$\{[2,6],[0,0]\}	**6**	**8**
ACA-I	4	20
$GeAr\{8,2,2\}$	4	12
$QuAd\{[4,3,1],[0,1,3]\}$	4	12
$QuAd$\{[4,4],[0,0]\}	**4**	**8**

shows that $QuAd_o$ configurations offer the best quality-efficiency trade-off compared to other LLAA configurations.

3.4 Limitations of $QuAd$ and the Unexplored Design Space of Low-power Approximate Adders

Although $QuAd_o$ configurations offer the optimal quality-efficiency trade-offs compared to other LLAA configurations covered by the $QuAd$ model, the design space of approximate adders is not limited to LLAAs. Another class of approximate adders, i.e., Low-power Approximate Adders (LPAAs), has also gained a lot of attention due to the ever-growing demand for ultra resource-efficient computing systems. These adders are usually constructed using smaller building blocks, such as approximate full-adders. IMPACT [GMP+11]) designs are a few of the earliest approximate full-adder designs proposed for trading accuracy for power and energy efficiency. These full-adders are cascaded to build large multi-bit LPAAs, as illustrated with the help of a generic multi-stage N-bit adder in Fig. 3.16.

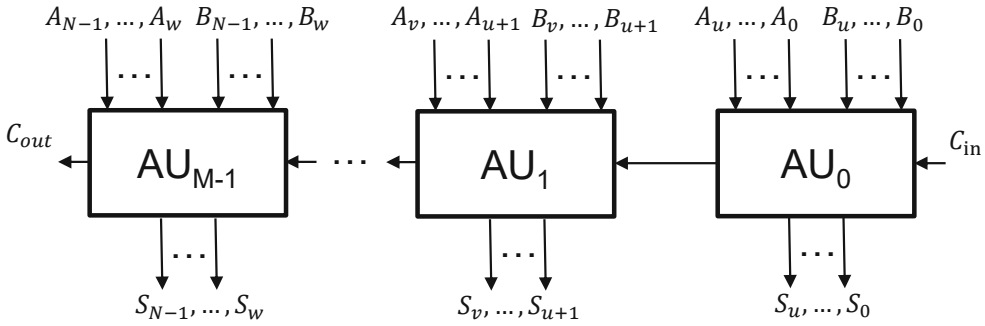


Figure 3.16: A generic N-bit adder composed of M cascaded Adder Units (AUs). The AUs can be accurate (e.g., accurate full-adders) or approximate (e.g., approximate full-adders), and approximations are typically employed at lower significance locations. The adder takes in two N-bit operands A and B and a carry-in (C_{in}) signal as inputs to generate an $N+1$-bit output, i.e., an N-bit sum (S) and a carry-out (C_{out}) signal. Each AU can be of arbitrary bit-width regardless of the bit-width of rest of the AUs.

Given the vast design space of approximate full-adders (e.g., see [AKL16][GMP+11]), it is challenging for a designer to select appropriate adder configurations for a particular application and scenario. Choosing the optimal design requires comparison between all possible configurations in terms of output quality and resource-efficiency. Although exhaustive simulations can be used to estimate quality and hardware metrics, such methods can be highly time consuming. As shown in Fig. 3.17, the exponential growth in the simulation time of an adder with the increase in the adder bit-width can make exhaustive simulation-based approaches in-feasible for practical scenarios, specifically for large adders. Hence, cost-effective error and hardware metric estimation techniques are required to enable efficient design space exploration. Towards this, various methodologies have been proposed to analyze the error characteristics of approximate adders [WLG+18][MHH+17][AHS17b]; however, such methodologies have mainly focused on statistically analyzing LLAAs, *and not the LPAAs*. Moreover, the works that focused on LPAAs targeted only specific configurations or just the *overall error occurrence probability estimation* [AHS17b]. The following section presents a novel lightweight method for efficiently computing the PMF of error of LPAAs composed of cascaded approximate units. Fig. 3.18, highlights the significance of the proposed methodology for efficient design space exploration of LPAAs. Note that the PMF of error can be used to estimate almost all types of error metrics, e.g., MSE, MED, and error rate.

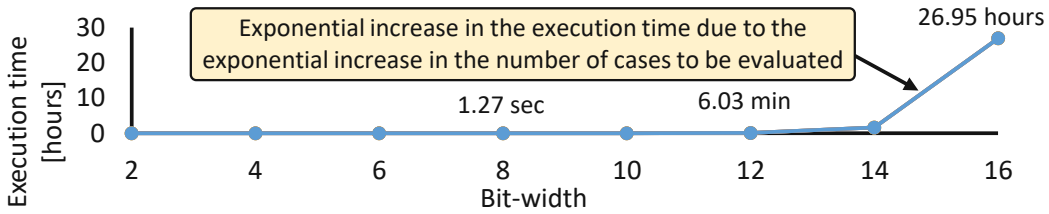


Figure 3.17: Execution time for computing PMF of error of different adders composed of smaller cascaded approximate full-adder units using exhaustive simulations.

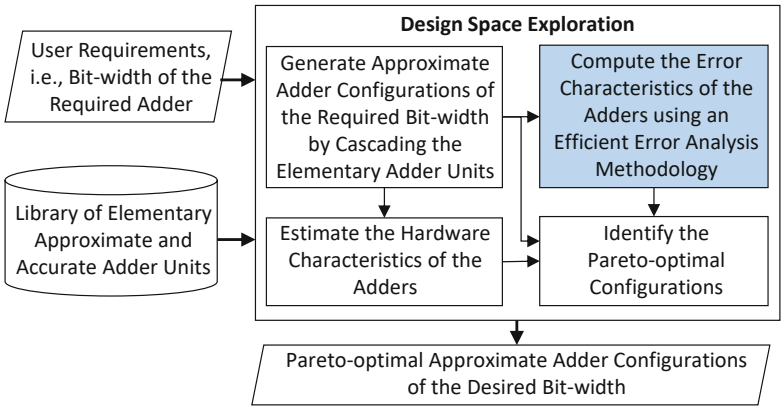


Figure 3.18: A flow for design space exploration of approximate adders composed of cascade of approximate adder units, where our novel contribution is highlighted in blue.

3.5 PEMACx: Methodology for Computing PMF of Error of Approximate Adders composed of Cascaded Adder Units

Building an accurate lightweight method for computing the PMF of error of an LPAA composed of cascaded approximate units poses two key challenges:

1. The probability distribution of the C_{in} of each AU/stage (except the first) has to be calculated. This is due to the fact that the probability distribution of the C_{in} of a stage entirely depends on the types of AUs used in the earlier stages and their corresponding input distributions.

2. The overall PMF of error of an adder composed of cascaded AUs cannot simply be computed by convolving the PMFs of error of all the individual AUs (as proposed in [MHH+17][WLG+18]). This is mainly because the probability distribution of C_{in} of each AU depends on the previous stages, and a change in the input distribution of even one of the previous stages can alter the probability distribution of C_{in} and thereby the error distribution of the current stage. Moreover, an error generated in a stage with C_{out} having a specific value will appear in the subsequent stage only in combinations where C_{in} has the same value. For example, the Type 1 adder shown in Table 3.3 produces an error of either 0 or +1 when C_{out} is 1, and an error of 0 or -1 when C_{out} is 0.

Table 3.3: Truth tables and error characteristics of prominent state-of-the-art low-power full-adders (as proposed in [AKL16][GMP+11]). The output combinations in which the resultant sum, or carry-out, or both are erroneous are highlighted in red.

Inputs			Accurate FA			Type 1			Type 2			Type 3			Type 4			Type 5			Type 6			Type 7		
A	B	C_{in}	Sum	C_{out}	$Error$	Sum	C_{out}	$Error$	Sum	C_{out}	$Error$	Sum	C_{out}	$Error$	Sum	C_{out}	$Error$	Sum	C_{out}	$Error$	Sum	C_{out}	$Error$	Sum	C_{out}	$Error$
0	0	0	0	0	0	0	0	0	1	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	0	0	-1	1	1	2	1	0	0
0	1	0	1	0	0	0	1	1	1	0	0	0	1	1	0	0	-1	1	0	0	1	0	0	1	0	0
0	1	1	0	1	0	0	1	0	0	1	0	0	1	0	1	0	-1	1	0	-1	0	1	0	1	1	1
1	0	0	1	0	0	0	0	-1	1	0	0	1	0	0	0	1	1	0	1	1	1	0	0	1	0	0
1	0	1	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	1	1	1
1	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	1	1	1	0	0	-2	0	1	0
1	1	1	1	1	0	1	1	0	0	1	-1	0	1	-1	1	1	0	1	1	0	1	1	0	1	1	0
# of Error Cases			0			2			2			3			3			4			2			2		
MSE			0			0.25			0.25			0.375			0.375			0.5			1			0.25		
MED			0			0.25			0.25			0.375			0.375			0.5			0.5			0.25		

To address the aforementioned challenges, this section presents a novel lightweight method, PEMACx, to compute the PMF of error of adders composed of cascaded approximate units. The proposed analytical method recursively computes two separate PMFs of error, i.e., one PMF of error per C_{out} value, and the probability distribution of C_{out} at each stage using the PMFs of the previous stage and the probability distribution of inputs of the current stage. Note that the PMFs at a stage contains information of the PMFs of the current stage along with the information of all the previous stages. Computing two

separate PMFs of error at each stage is important to keep track of the error distribution for each C_{in} value in the subsequent stage.

Fig. 3.19 shows the overall methodology for computing the PMF of error of an approximate adder composed of smaller cascaded approximate units. The methodology takes the adder configuration, the probability distribution of each bit of the two operands and the probability distribution of the carry-in (C_{in}) as inputs. Then, in the first step, it initializes the input PMF of error for cases with $C_{in} = 0$ as $PE_{I\{C_{in}=0\}}(0) = 1$ and for cases with $C_{in} = 1$ as $PE_{I\{C_{in}=1\}}(0) = 1$. In Step 2, starting from the first stage of the adder, the methodology recursively perform the following steps for each stage.

(a) Using the probability distribution of the corresponding input bits and the error characteristics of the current AU, the methodology computes four partial PMFs of error in parallel, i.e., one for each $\{C_{in}, C_{out}\}$ combination. The partial PMF of error for the combination $\{C_{in} = x, C_{out} = y\}$ is represented as $PPE_{CS\{C_{in}=x,C_{out}=y\}}$, where x *and* $y \in \{0, 1\}$. The partial PMFs are computed using Algo. 3.2, which is discussed in detail in Section 3.5.2.

(b) Using the probability distribution of the corresponding input bits and the functionality of the current adder stage, the methodology computes the probability distribution of C_{out} of the current stage.

(c) Using the computed partial PMFs of the current stage and the PMFs of error from the previous stage, the methodology computes two separate PMFs of error, i.e., one for each C_{out} value. The PMFs of error at the current stage for $C_{out} = 0$ and $C_{out} = 1$ are represented as $PE_{O\{C_{out}=0\}}$ and $PE_{O\{C_{out}=1\}}$, respectively. The mathematical formulation is discussed in Section 3.5.1.

After performing Step 2 for all the stages, in Step 3, the methodology combines the two PMFs of error from the last stage of the adder to get the overall PMF of error, which then can be used to compute the desired error metrics, e.g., MSE, MED, and error rate.

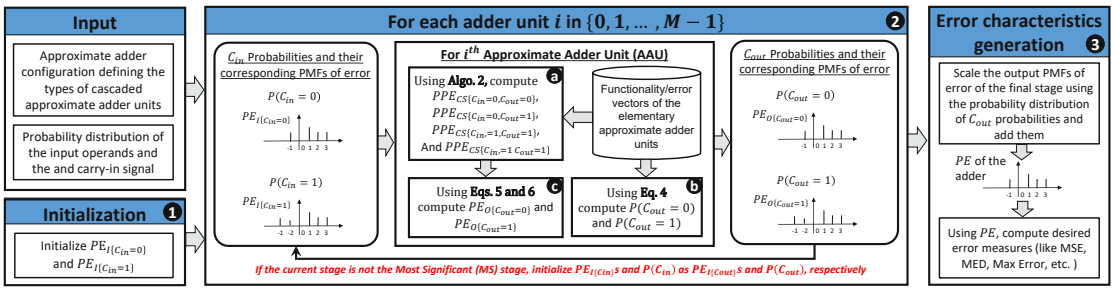


Figure 3.19: Flow of the proposed methodology for computing PMF of error and the desired error metrics of an adder composed of cascaded approximate units.

3.5.1 Mathematical Model

Assume $PE_{I\{C_{in}=0\}}$ and $PE_{I\{C_{in}=1\}}$ as the input PMFs of error for $C_{in} = 0$ and $C_{in} = 1$ cases, respectively, at the current adder stage. Further assume that the probability distribution of input bits is represented using the following terminology.

- $P(C_{in})$ is the probability of C_{in} being 1
- $P(\overline{C_{in}})$ is the probability of C_{in} being 0 and is equal to 1-$P(C_{in})$
- $P(C_{out})$ is the probability of C_{out} being 1
- $P(\overline{C_{out}})$ is the probability of C_{out} being 0 and is equal to 1-$P(C_{out})$
- $P(A_i)$ is the probability of A_i being 1
- $P(\overline{A_i})$ is the probability of A_i being 0 and is equal to 1-$P(A_i)$
- $P(B_i)$ is the probability of B_i being 1
- $P(\overline{B_i})$ is the probability of B_i being 0 and is equal to 1-$P(B_i)$

where, considering $h \leq i \leq j$, $\{A_h, A_{h+1}, ..., A_j\}$, $\{B_h, B_{h+1}, ..., B_j\}$ and C_{in} are inputs to the current stage. Given $PE_{I\{C_{in}=0\}}$, $PE_{I\{C_{in}=1\}}$ and the input probabilities, the probability of error value E at the output of a stage can be computed using the following equation, assuming independence between stages.

$$PE_O(E) = \sum_{x=-\infty}^{\infty} (PE_{I\{C_{in}=0\}}(x) \times PPE_{CS\{C_{in}=0\}}(E-x)) + \sum_{x=-\infty}^{\infty} (PE_{I\{C_{in}=1\}}(x) \times PPE_{CS\{C_{in}=1\}}(E-x)) \quad (3.13)$$

Here, $PPE_{CS\{C_{in}=0\}}$ and $PPE_{CS\{C_{in}=1\}}$ correspond to partial (un-normalized) PMF of error of the current stage for the cases with $C_{in} = 0$ and $C_{in} = 1$, respectively. However, as mentioned earlier, computing the PMF of error of the complete adder recursively requires separate PMFs of error for each C_{out} value. Therefore, the aforementioned equation is divided into two parts based on the value of C_{out}. The partial PMF of error for cases with $C_{out} = 0$ can be written as:

$$PPE_{O(C_{out}=0)}(E) = \sum_{x=-\infty}^{\infty} (PE_{I\{C_{in}=0\}}(x) \times PPE_{CS\{C_{in}=0,C_{out}=0\}}(E-x)) + \sum_{x=-\infty}^{\infty} (PE_{I\{C_{in}=1\}}(x) \times PPE_{CS\{C_{in}=1,C_{out}=0\}}(E-x)) \quad (3.14)$$

Similarly, the partial PMF of error for cases with $C_{out} = 1$ can be written as:

$$PPE_{O(C_{out}=1)}(E) = \sum_{x=-\infty}^{\infty} (PE_{I\{C_{in}=0\}}(x) \times PPE_{CS\{C_{in}=0,C_{out}=1\}}(E-x)) + \sum_{x=-\infty}^{\infty} (PE_{I\{C_{in}=1\}}(x) \times PPE_{CS\{C_{in}=1,C_{out}=1\}}(E-x)) \quad (3.15)$$

The resultant PMFs are called partial because:

$$\begin{aligned} \sum_{i=-\infty}^{\infty} PPE_{O\{C_{out}=0\}}(i) &= P(C_{out}=0) \\ \sum_{i=-\infty}^{\infty} PPE_{O\{C_{out}=1\}}(i) &= P(C_{out}=1) \end{aligned} \quad (3.16)$$

and $P(C_{out} = 0)$ (or $P(C_{out} = 1)$) is not necessarily equal to 1. The aforementioned equations can also be realized using convolution (represented using '*') and vector sum operations, as shown in the following equation.

$$\begin{aligned} PPE_{O\{C_{out}=0\}} &= PPE_{CS\{C_{in}=0,C_{out}=0\}} * PE_{I\{C_{in}=0\}} + \\ &\quad PPE_{CS\{C_{in}=1,C_{out}=0\}} * PE_{I\{C_{in}=1\}} \\ PPE_{O\{C_{out}=1\}} &= PPE_{CS\{C_{in}=0,C_{out}=1\}} * PE_{I\{C_{in}=0\}} + \\ &\quad PPE_{CS\{C_{in}=1,C_{out}=1\}} * PE_{I\{C_{in}=1\}} \end{aligned} \quad (3.17)$$

To ensure that the probability of C_{out} is used only once in the next stage, the partial PMFs are normalized by the corresponding sums, i.e.,

$$\begin{aligned} &if\ P(C_{out}=0) \neq 0\ then\ PE_{O\{C_{out}=0\}} = PPE_{O\{C_{out}=0\}}/P(C_{out}=0) \\ &\quad else\ PE_{O\{C_{out}=0\}}(E) = \begin{cases} 1 & E=0 \\ 0 & otherwise \end{cases} \\ &if\ P(C_{out}=1) \neq 0\ then\ PE_{O\{C_{out}=1\}} = PPE_{O\{C_{out}=1\}}/P(C_{out}=1) \\ &\quad else\ PE_{O\{C_{out}=1\}}(E) = \begin{cases} 1 & E=0 \\ 0 & otherwise \end{cases} \end{aligned} \quad (3.18)$$

Note, the above equations are recursively used for each stage, where the PMFs of error and the C_{out} probability are treated as input PMFs of error and C_{in} probability (respectively) of the next stage.

3.5.2 Generalized Methodology

Based on the mathematical model presented in Section 3.5.1, a generalized methodology can be formulated and then used for computing the PMF of error of an adder composed of cascaded approximate AUs. The overall algorithm is summarized in Algo. 3.1 and is composed of the following steps:

Algorithm 3.1: Pseudo-code for computing PMF of Error of an Adder

Inputs: $AACV$: Approximate Adder Configuration Vector; IPV_A: Input Probability Vector of operand A; IPV_B: Input Probability Vector of operand B; $P(C_{in})$: Probability of input carry bit; EVs: Error Vectors of all elementary adder units; $CiVs$: Carry in Vectors of all elementary adder units; $CoVs$: Carry out Vectors of all elementary adder units
Outputs: PE: Overall PMF of Error
Initialize: PE_0: $PE_0(0) = 1$; PE_1: $PE_1(0) = 1$
for $h = \{0, 1, 2, ..., M-1\}$ **do**
  Evaluate $PVIC$ using Eq. 3.19
  CiV = Carry in Vector of $AACV(h)$
  CoV = Carry out Vector of $AACV(h)$
  $EV_C = 2^{h-1} \times EV$ of $AACV(h)$
  for $i = \{0, 1\}$ **do**
    for $j = \{0, 1\}$ **do**
      Compute index vector (I), where $CiV == i \,\&\&\, CoV == j$
      Compute $PPE_{\{i,j\}}$ using I, EV_C, and $PVIC$ in Algo. 3.2
    end
  end
  $PPE_0 = PPE_{\{0,0\}} * PE_0 + PPE_{\{1,0\}} * PE_1$
  $PPE_1 = PPE_{\{0,1\}} * PE_0 + PPE_{\{1,1\}} * PE_1$
  $P(C_{out}) = [PVIC].[CoV\ of\ AACV(h)]'$
  if $h \neq M-1$ **then**
    if $1 - P(C_{out}) \neq 0$ **then**
      $PE_0 = PPE_0/(1 - P(C_{out}))$
    else
      $PE_0(0) = 1$ and 0 otherwise
    end
    if $P(C_{out}) \neq 0$ **then**
      $PE_1 = PPE_1/P(C_{out})$
    else
      $PE_1(0) = 1$ and 0 otherwise
    end
    $P(C_{in}) = P(C_{out})$
  else
    $PE = PPE_0 + PPE_1$
  end
end
return PE

1. First, Error Vector (EV), Carry-in Vector (CiV), and Carry-out Vector (CoV) are defined for each possible type of the AU. The vectors store the error value, carry-in bit and carry-out bit corresponding to each possible input combination of an elementary AU. The vectors of the approximate adder types shown in Table 3.3 are presented in Table 3.4.

2. Then, the approximate adder configuration and the probability distribution of the input bits, which are used as inputs to the method, are defined. The adder configuration to be analyzed is represented using an Approximate Adder Configuration Vector ($AACV$) of length M, i.e., $\{A_0, A_1, A_2, ..., A_{M-1}\}$, where A_i defines the type of the i^{th} AU. The probability distribution of inputs A and B is defined in terms of Input Probability Vectors ($IPVs$), i.e., IPV_A and IPV_B. The $IPVs$ define the probabil-

Algorithm 3.2: Pseudo-code for computing PPE

```
Inputs: EV: Error Vector of the sub-block; PVIC: Probability Vector of Input Combinations; I:
  Indexes
Outputs: PPE: Partial PMF of Error
Initialize: PPE: Empty
for i = {0, 1, ..., length(I) − 1} do
  |  PPE(EV(I(i))) = PPE(EV(I(i))) + PVIC(I(i))
end
return PPE
```

Table 3.4: Functional and error vectors of the approximate full-adders proposed in [GMP+11] and [AKL16].

Adder Type	EV	CiV	CoV
Accurate FA	$\{0,0,0,0,0,0,0,0\}$	$\{0,1,0,1,0,1,0,1\}$	$\{0,0,0,1,0,1,1,1\}$
Type 1	$\{0,0,1,0,-1,0,0,0\}$	$\{0,1,0,1,0,1,0,1\}$	$\{0,0,1,1,0,1,1,1\}$
Type 2	$\{1,0,0,0,0,0,0,-1\}$	$\{0,1,0,1,0,1,0,1\}$	$\{0,0,0,1,0,1,1,1\}$
Type 3	$\{1,0,1,0,0,0,0,-1\}$	$\{0,1,0,1,0,1,0,1\}$	$\{0,0,1,1,0,1,1,1\}$
Type 4	$\{0,0,-1,-1,1,0,0,0\}$	$\{0,1,0,1,0,1,0,1\}$	$\{0,0,0,0,1,1,1,1\}$
Type 5	$\{0,-1,0,-1,1,0,1,0\}$	$\{0,1,0,1,0,1,0,1\}$	$\{0,0,0,0,1,1,1,1\}$
Type 6	$\{0,2,0,0,0,0,-2,0\}$	$\{0,1,0,1,0,1,0,1\}$	$\{0,1,0,1,0,1,0,1\}$
Type 7	$\{0,0,0,1,0,1,0,0\}$	$\{0,1,0,1,0,1,0,1\}$	$\{0,0,0,1,0,1,1,1\}$

ity distribution of each individual bit of the input operands and are represented as $IPV_A = \{P(A_0), P(A_1), ..., P(A_{N-1})\}$ and $IPV_B = \{P(B_0), P(B_1), ..., P(B_{N-1})\}$. Similarly, the probability distribution of carry-in is defined using $P(C_{in})$.

3. In the next step, the Probability Vector of all the Input Combinations ($PVIC$) of the current AU is computed. Assuming independence between the input bits, $PVIC$ of the current stage can be computed using the following equation.

$$
\begin{aligned}
PVIC = \; & [P(\overline{A_h}).P(\overline{A_{h+1}}).....P(\overline{A_j}).P(\overline{B_h}).P(\overline{B_{h+1}}).....P(\overline{B_j}).P(\overline{C_{in}}), \\
& P(\overline{A_h}).P(\overline{A_{h+1}}).....P(\overline{A_j}).P(\overline{B_h}).P(\overline{B_{h+1}}).....P(\overline{B_j}).P(C_{in}), \\
& P(\overline{A_h}).P(\overline{A_{h+1}}).....P(\overline{A_j}).P(\overline{B_h}).P(\overline{B_{h+1}}).....P(B_j).P(\overline{C_{in}}), \\
& \vdots \\
& P(A_h).P(A_{h+1}).....P(A_j).P(B_h).P(B_{h+1}).....P(B_j).P(C_{in})]
\end{aligned} \tag{3.19}
$$

Here, $h \leq j$ and bits $\{A_h, A_{h+1}, ..., A_j\}$, $\{B_h, B_{h+1}, ..., B_j\}$ and C_{in} are inputs to the current stage.

4. Next, the partial PMFs of error for all combinations of $\{C_{in}, C_{out}\}$ are computed using Algo. 3.2. The algorithm only uses EV and $PVIC$ of the current AU and accumulates the probabilities of the cases which result in the same error value to generate the partial PMFs of error. Note, for each AU in the $AACV$, the EV of

the elementary adder unit is scaled as per the significance location of the adder unit, as mentioned in Line 8 of Algo. 3.1.

5. Then, the partial PMFs of error of the current stage along with the input PMFs of error associated with $C_{in} = 0$ and $C_{in} = 1$ are used for computing the partial PMFs of Error for $C_{out} = 0$ and $C_{out} = 1$ cases, as mentioned in Section 3.5.1 using Eq. 3.17. This is realized using Lines 15 and 16 in Algo. 3.1.

6. Next, $P(C_{out})$ of the current stage is computed using the dot product of $PVIC$ and CoV vectors (see Line 17 in Algo. 3.1).

7. Now, if the current stage is not the most significant stage, the partial PMFs associated with $C_{out} = 0$ and $C_{out} = 1$ are normalized using $P(C_{out})$, as mentioned in Lines 19-28 of Algo. 3.1. Then, these PMFs and $P(C_{out})$ are defined as inputs to the subsequent stage, and Steps 3-7 are repeated until the most significant stage is reached. However, if the current stage is the most significant stage, the partial PMFs are accumulated to generate the overall PMF of error of the adder configuration (see Line 31 in Algo. 3.1).

Note, the above methodology is not limited to approximate full-adder units, rather it is valid for any possible size and type of AU as long as the AUs are connected in cascade to construct large multi-bit adders, as shown in Fig. 3.16.

3.6 Model Validation and Usability

3.6.1 Model Validation

To validate the proposed methodology, the results of PEMACx are compared with exhaustive simulation results for various approximate adder configurations. All the simulations are performed using MATLAB 2018b running on an Intel Core-i5 with 16 GB of RAM. Figs. 3.20 and 3.21 show the comparison between the error measures (i.e., MSE and MED) computed using exhaustive simulations and PEMACx for different 8-bit and 12-bit approximate adders. Note, for this experiment, the inputs are assumed to be uniformly distributed, i.e., $P(C_{in}) = P(A_i) = P(B_i) = 0.5 \; \forall \; i \in \{0, 1, ..., N-1\}$. As evident from the figures, the results generated using PEMACx are exactly the same as the results achieved using exhaustive simulations. To highlight the efficiency of the proposed methodology, Fig. 3.22 presents the execution time of PEMACx for different adder lengths along with the time required to generate the same results using exhaustive simulations. As evident from the figure, PEMACx requires significantly less amount of time than exhaustive simulations. For example, for an 8-bit adder composed of cascaded approximate adder units, the exhaustive simulations require 1.272 sec to generate the PMF of error, while the proposed PEMACx methodology requires only 0.00043 sec to generate the exact same output.

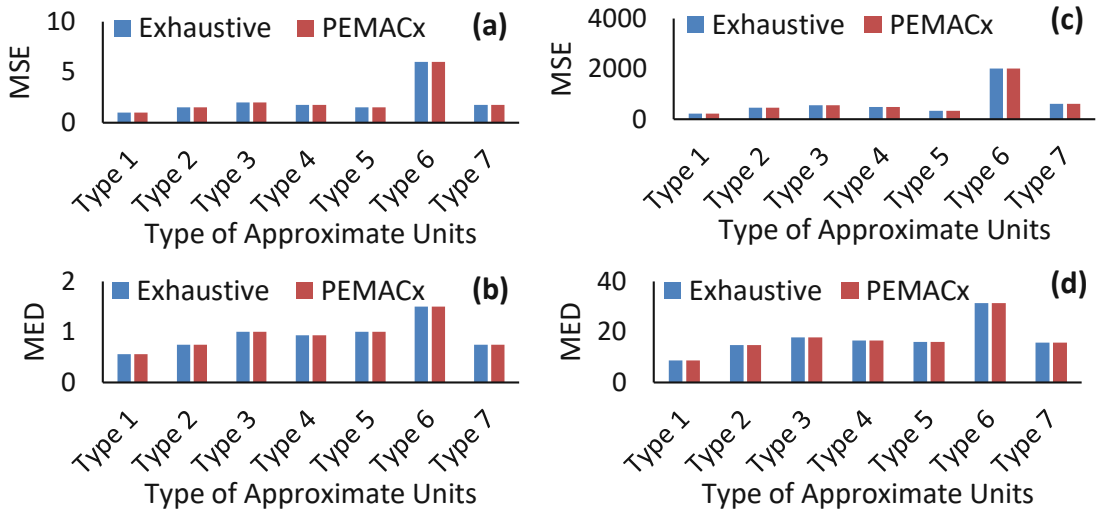


Figure 3.20: Error characteristics of different **8-bit** low-power adders composed of approximate adder types shown in Table 3.3 computed using *Exhaustive* simulations and *PEMACx*. (a) and (b) illustrate the MSE and MED, respectively, of adders with two least-significant full-adders approximated using a specific type of approximate unit. (c) and (d) illustrate the MSE and MED, respectively, of adders with six least-significant full-adders approximated.

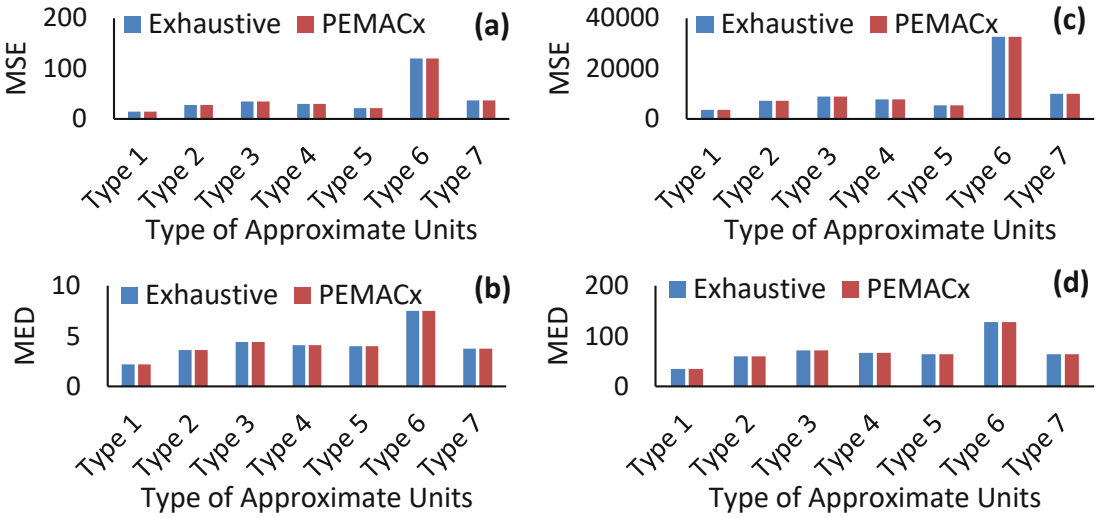


Figure 3.21: Error characteristics of different **12-bit** low-power adders composed of approximate adder types shown in Table 3.3 computed using *Exhaustive* simulations and *PEMACx*. (a) and (b) illustrate the MSE and MED, respectively, of adders with four least-significant full-adders approximated using a specific type of approximate unit. (c) and (d) illustrate the MSE and MED, respectively, of adders with eight least-significant full-adders approximated.

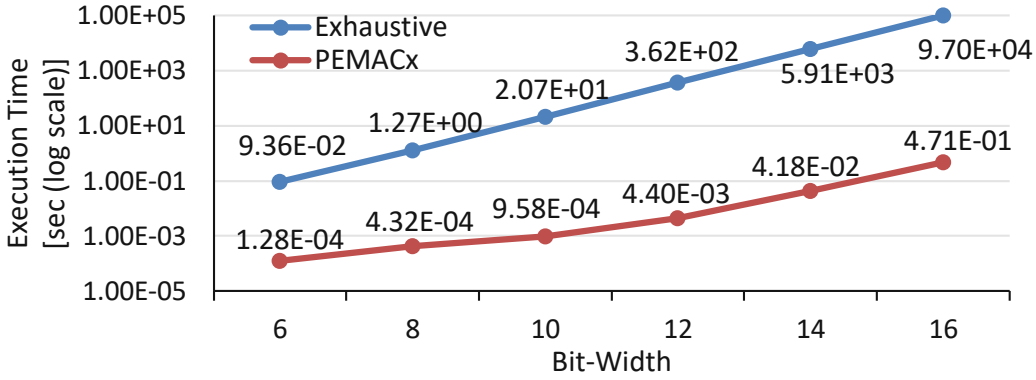


Figure 3.22: Execution time of PEMACx and exhaustive simulations for different adder lengths.

Table 3.5: Hardware characteristics of accurate and type 1-5 approximate FAs from [SHR+16].

	Accu.FA	Type 1	Type 2	Type 3	Type 4	Type 5
Power [nW]	1130	771	294	198	416	0
Area [GE]	4.41	4.23	1.94	1.59	1.76	0

3.6.2 Design Space Exploration for an 8-bit Low-power Adder

To illustrate the usability of PEMACx, it is employed for performing a design space exploration for an 8-bit low-power adder. For this exploration, only the Type 1-5 approximate full-adders shown in Table 3.3 and the accurate full-adder are used as AUs. The power and area characteristics of the full-adders are summarized in Table 3.5. For this exploration, the PEMACx flow shown in Fig. 3.18 is used for computing the error metrics and a simple additive model for power estimation of multi-bit adder configurations. The proposed methodology explored the complete design space, i.e., a total of 1,679,616 configurations, in 653.24 sec. Fig. 3.23 shows the complete design space, where the Pareto-optimal configurations are highlighted with red triangles. The Pareto-optimal configurations are the ones that have accurate FAs at the most significant locations, Type 5 approximate FAs at the least significant locations, and Type 2, 3, or 5 FA in between the two sets. Note, for this analysis, the inputs are assumed to be uniformly distributed.

3.6.3 PEMACx for Low-Latency, Low-Power Approximate Adders

To illustrate the usability of PEMACx for low-latency, low-power adders such as the $QuAd_o$ designs and the designs presented in [MAFL10], the results of PEMACx are compared with the results generated using exhaustive simulations for 8-bit and 12-bit $QuAd_o$ adders considering different L_{max} values. To enable the use of PEMACx for computing the PMF of error of $QuAd_o$ adders, another approximate full-adder unit

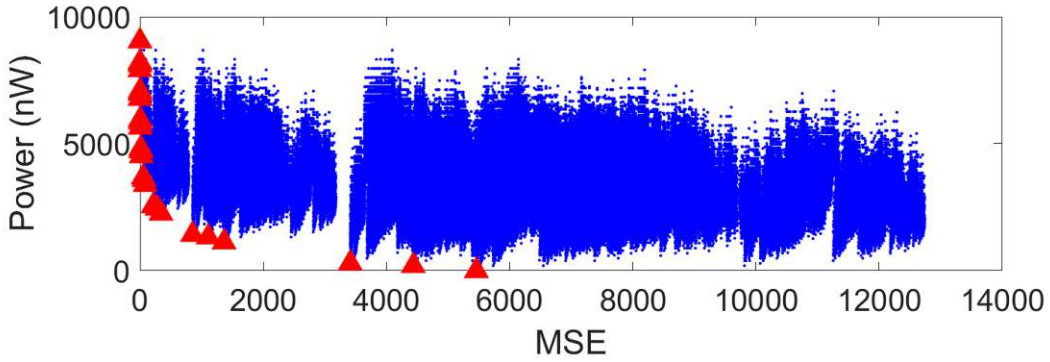


Figure 3.23: Design space of 8-bit approximate adders using accurate and Types 1-5 approximate FAs shown in Table 3.3. The pareto-optimal configurations are highlighted using red triangles.

(Type 8) is defined with $EV = \{0, 0, 0, -2, 0, -2, -2, -2\}$, $CiV = \{0, 1, 0, 1, 0, 1, 0, 1\}$ and $CoV = \{0, 0, 0, 0, 0, 0, 0, 0\}$ and then used at the MSB location of all the sub-adders, except the most significant sub-adder (see Fig. 3.24). Fig. 3.25 shows that the MSE results generated using PEMACx are the same as the ones obtained through exhaustive simulations.

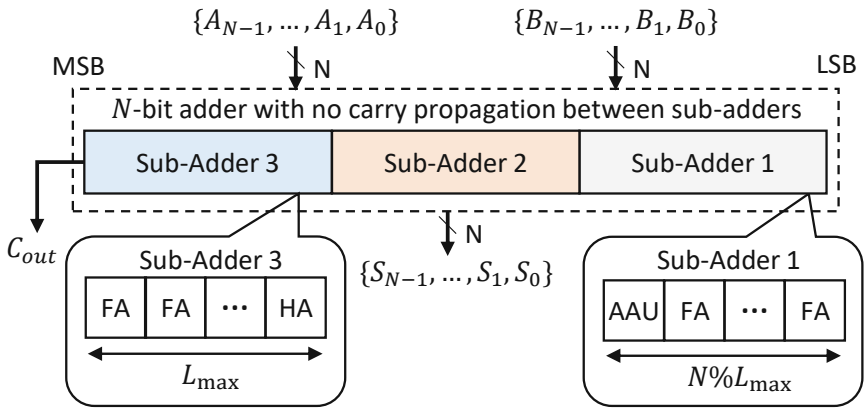


Figure 3.24: An example illustration of high-performance (low-latency) low-power adder. Each sub-adder is of L_{max} length except for the first sub-adder which is of remainder number of bits ($N\%L_{max}$). Here, HA represents a half-adder, FA represents a full-adder, and AAU is an approximate adder unit which only computes the sum bit.

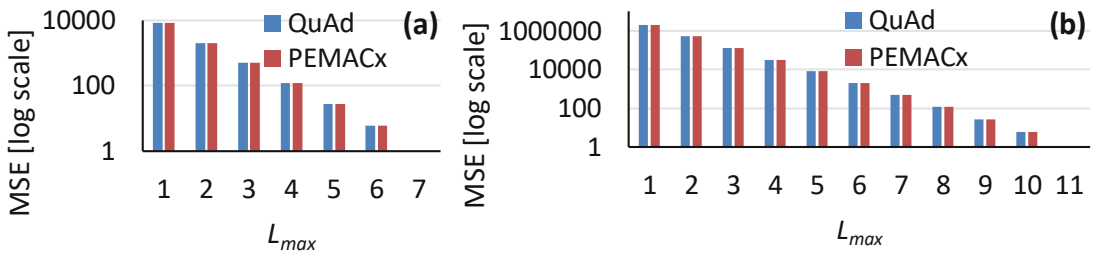


Figure 3.25: MSE of different 8-bit (a) and 12-bit (b) $QuAd_o$ adder configurations computed using exhaustive simulations and PEMACx.

3.7 Limitations of PEMACx and the Motivation for Application and Data-Aware Analysis Methodology

The QuAd analysis presented in Section 3.2.2 and the PEMACx methodology presented in Section 3.5 have two major shortcomings:

1. The operands are assumed to be independent of each other.
2. The bits of each operand are assumed to be independent of each other.

Apart from the above assumptions, the *QuAd* analysis further assumed that the inputs are uniformly distributed. Although these assumptions are in line with most of the works in the literature, such as [SS15][MHH+17][LHL15], they are not perfectly valid in all real-world use cases, and therefore, the actual output quality can deviate significantly from the estimated quality computed using these analytical models. To understand this, consider two different image processing applications with comparable workloads: a low-pass filtering application with a 4x4 smoothing kernel and a template matching application with a template size of 4x4. Both applications require the addition of 16 values that can be executed using approximate adders.

Depending on the intended use case and deployment environment of the system, the input to the approximate components may have some specific characteristics. For both the above-mentioned applications, three different types of input data are considered: (1) satellite imagery, (2) sports stadium surveillance, and (3) sky images. Note that irrespective of the input distribution, both the applications, i.e., the low-pass filtering application and the template matching application, require accumulation of data that is coming from the pixel neighborhood, and usually, in images, neighboring pixels are highly correlated due to spatial locality. For the motivational analysis, the GeAr model [SAHH15] is used, as it covers various state-of-the-art LLAAs and can be configured by defining the number of prediction bits (P) and resultant bits (R), i.e., $GeAr(N, R, P)$. Assuming parallel cascaded addition, four stages of adders are required to compute the final sum. The considered accelerator utilized GeAr(8,2,2) for the first stage additions, GeAr(9,2,3) for

the second stage additions, GeAr(10,2,2) for the third stage additions, and GeAr(11,2,3) for the final stage addition. The adder configurations are shown in Fig. 3.26. Fig. 3.27 compares the MSE computed using the analytical model proposed in [MHH+17] with the MSE computed using functional simulations for different applications and datasets. It can be observed from the figure that the MSE computed using [MHH+17] turns out to be the same value for all the cases. This is mainly because [MHH+17] assumes input bits to be independent (uncorrelated) and uniformly distributed. On the contrary, functional simulations show that MSE of the same accelerator configuration can vary across datasets and applications. *Thus, it is essential to consider the actual data distribution of inputs and the correlation between input bits and operands for accurate error estimation.*

Apart from considering the actual input data distribution, there is also a need to consider the time required for computing the error estimates. Fig. 3.27 shows that the execution time for functional simulations increases proportionally with the increase in the number of input samples. Thus, the time required for functional simulations using actual input samples can be significantly higher than the analytical method [MHH+17], which in this case requires only 5×10^{-3} seconds to compute the error estimates.

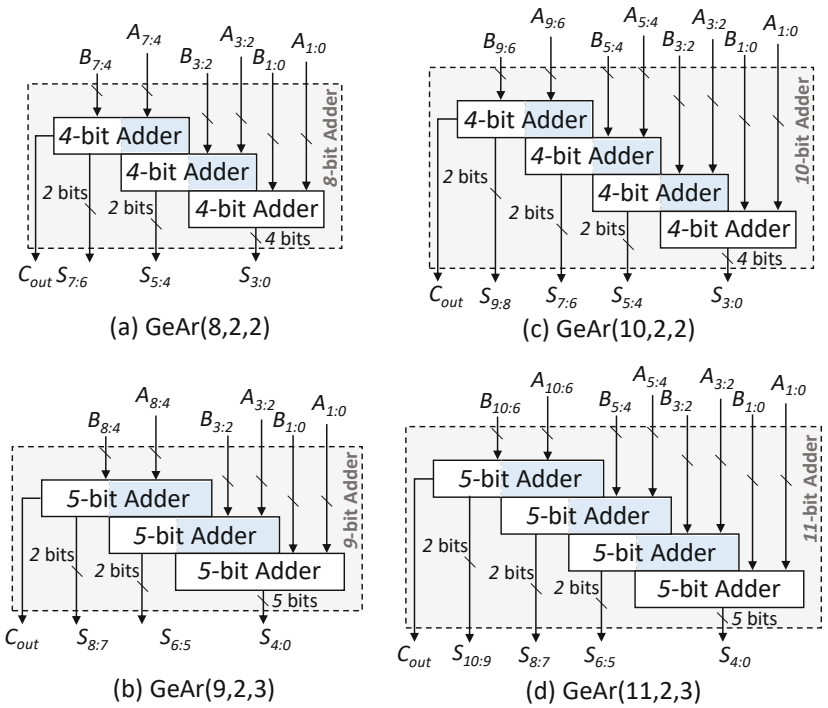
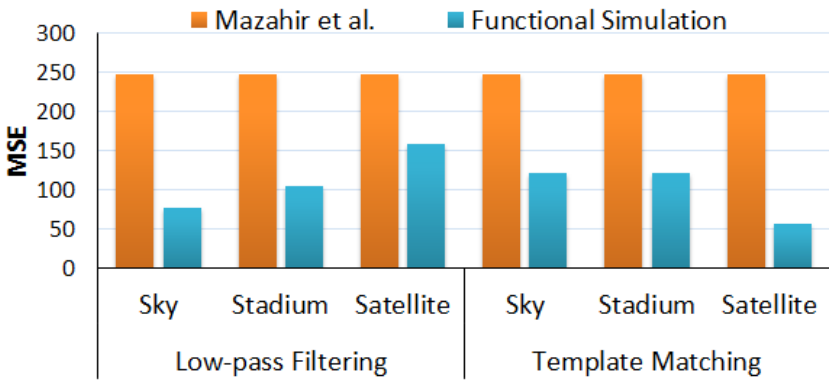
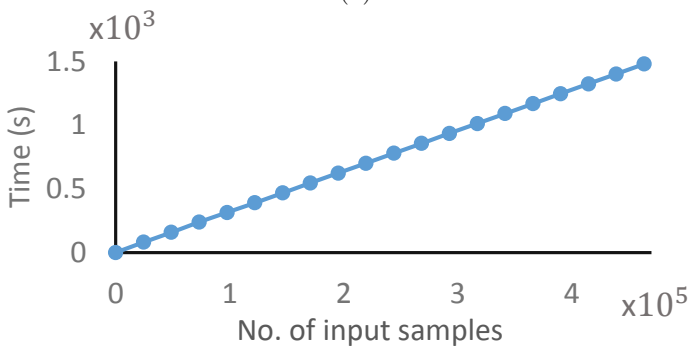


Figure 3.26: The structure of different GeAr adder configurations.

Target Research Problem: As highlighted in the above case study, there is a dire need for an analytical method that can offer fast yet accurate error estimation for approximate adders. The method should take into consideration the data and application knowledge to provide realistic (i.e., accurate in real-world settings) error estimates in a reasonable



(a)



(b)

Figure 3.27: (*a*) Comparison of MSE evaluated using functional simulations and using the analytical modeling (assuming uniform distribution). (*b*) Execution time of functional simulations for template matching application as a function of number of input samples.

amount of time. Moreover, the methodology should be generic enough to be applicable to a wide variety of approximate adders, i.e., it should not be limited to a specific set of approximate adders and/or configuration sets.

3.8 DAEM: A Data and Application Aware Error Analysis Methodology for Approximate Adders

To develop an effective data and application-aware error estimation methodology, this section first highlights different factors that must be considered to achieve a methodology that can offer a fast yet accurate estimation of error. Fig. 3.28a shows the Error Maps (E_{MAP}s) for a few notable 8-bit LLAA configurations from Speculative Adders (ACA), Segmented Adders (ESA), and Carry Select Adders (SCSA) classes. The E_{MAP} of an approximate adder is computed by populating the error values for each possible input combination. In the figure, x_1 and x_2 are the two adder operands. It can be observed that the error varies as a function of the input combination and not as a function of a

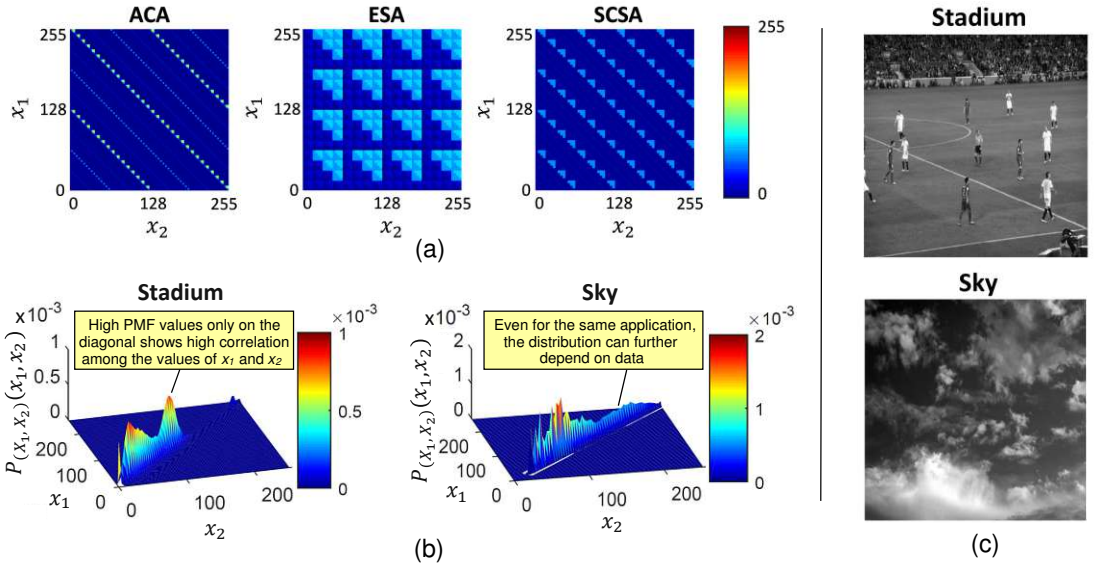


Figure 3.28: (*a*). Example E_{MAP} for 8-bit ACA with $k = 5$, ESA with $k = 2$, and SCSA with $k = 2$. The white colored locations in E_{MAP} represents accurate combinations. (*b*)Joint input probability distribution generated using neighboring pixels of two visually different sets of gray scale images shown in (c).

single operand. Thus, it is important to consider the complete E_{MAP} of approximate adders to achieve better error estimates. Apart from the $E_{MAP}s$ of approximate adders, the input distribution can also have a significant impact on the error estimates. Fig. 3.28b illustrates two sample 2D joint probability distributions generated using input data from two visually different sets of grayscale images (shown in Fig. 3.28c) for a low-pass filtering application. The distributions highlight that not all the input combinations are equally likely. Most of the high probability values are concentrated along the diagonal of the 2D joint input PMFs, which shows a significant correlation between inputs. Specifically, for applications that process data from spatial neighborhoods, the joint input probability distribution shall be highly concentrated in specific regions. In contrast, for applications that process data from independent sources, the joint input probability distribution shall be more spread out. Thus, a two-dimensional joint input probability distribution captures the probabilities as well as the interdependence of inputs and provides the data and application-specific knowledge required to develop a more accurate error estimation methodology. The next section employs 2D input PMF and $E_{MAP}s$ to develop an error estimation model for a two-operand adder. The model shall later be extended for a cascade of approximate adders.

3.8.1 Error estimation for an approximate adder

Assume the model of a standalone approximate adder, adding two operands x_1 and x_2, is defined as follows:

$$x_1 \widetilde{+} x_2 = y + eO \tag{3.20}$$

where, $\widetilde{+}$ represents approximate addition, y is the accurate output, and eO is the error induced due to approximation. Let $PMF_{(X_1,X_2)}$ be the two dimensional joint PMF representing the application-aware data dependence of the operands of an approximate adder, and it is mathematically defined as:

$$PMF_{X_1,X_2}(x_1, x_2) = P(X_1 = x_1, X_2 = x_2) \, \forall \, x_1 \, and \, x_2$$

Now, let PMF_{eG} be the probability distribution of the error generated due to the approximate addition of x_1 and x_2. PMF_{eG} is mathematically defined as:

$$PMF_{eG}(e) = P(eG = e) \quad \text{where} \quad e \in Z^* \wedge e \leq 2^{n-1}$$

PMF_{eG} can be computed by using the joint input PMF, i.e., PMF_{X_1,X_2}, and the pre-computed error map, i.e., E_{MAP}, of the specific approximate adder being used as explained in Algorithm 3.3. In particular, PMF_{eG} is computed by looking up E_{MAP} for all the instances of similar error values and summing the probabilities of all the combinations of inputs (stated in PMF_{X_1,X_2}) that result in the same error value. PMF_{X_1,X_2} can either be provided by the application engineer to the methodology or populated by logging the instances of input combinations of the adder, assuming all modules in the system to be accurate.

For a standalone approximate adder with inputs that are error free, the error introduced by the approximate adder itself is the only source of error in the system. Hence, the PMF of output error, i.e., PMF_{eO}, can be defined using the following equation.

$$PMF_{eO} = PMF_{eG}$$

Algorithm 3.3: Computing PMF_{eG} for a single approximate adder

1 **Inputs:** E_{MAP} i.e. Error Map of a specific adder AND PMF_{X_1,X_2} i.e. the joint PMF of inputs of a specific adder
2 **Outputs:** PMF_{eG} i.e. PMF of error generated
3 E_{vals} = Set of all unique error values in E_{MAP}
4 PMF_{eG} declaration as Empty Sparse Vector
5 **for** *each* E_{val} *in* E_{vals} **do**
6 | Points = $(E_{MAP} == E_{val})$
7 | $PMF_{eG}(E_{val})$ = Sum $(PMF_{X_1,X_2}$(Points))
8 **end**
9 **return** PMF_{eG}

3.8.2 Error estimation for an intermediate approximate adder

To develop a scalable error analysis methodology that could be extended for cascaded additions, consider an approximate adder that can be located anywhere in an adder tree. For example, consider adder 3 in Fig. 3.29a, which adds the outputs of adder 1 and adder 2. As per Eq. 3.20, the inputs to adder 3 are $y_1 + eO_1$ and $y_2 + eO_2$, i.e., the accurate outputs y_1 and y_2, as well as the approximation errors eO_1 and eO_2. Therefore, it can be said that the error at the output of adder 3 is composed of two different error terms:

1. *Error generated by adder 3* itself, which is characterized by PMF_{eG_3}. PMF_{eG_3} can be computed by using the data and application-aware joint input PMF_{Y_1,Y_2} with Algorithm 3.3.
2. *Error propagated to adder-3* by adders 1 and 2. This error is defined by PMF_{eP}, and it can be computed by convolving input error PMFs, assuming the errors to be independent of each other, similar to the assumption considered in [LGLG08]:

$$PMF_{eP_3} = PMF_{eO_1} * PMF_{eO_2} \tag{3.21}$$

Here, PMF_{eP_3} represents the PMF of the propagated error for adder 3, PMF_{eO_1} and PMF_{eO_2} relates to the PMF of error generated by adder 1 and adder 2, which are connected to the inputs of adder 3 as shown in Fig. 3.29, and '*' represents the convolution operator. Assuming the generated and propagated errors to be independent of each other, the PMF of cumulative output error of adder 3 can be mathematically written as:

$$PMF_{eO_3} = PMF_{eG_3} * PMF_{eP_3} \tag{3.22}$$

The computed PMF of error at each adder node n, i.e., PMF_{eO_n}, can be used to compute other error metrics such as MSE and MED using the following equations:

$$MED_{O_n} = |E_n|.P_{eO_n}, \tag{3.23}$$
$$MSE_{O_n} = |E_n|^2.P_{eO_n} \tag{3.24}$$

Here, E_n represents the magnitude vector of PMF_{eO_n} and P_{eO_n} represents the corresponding probability vector of PMF_{eO_n}.

The above analysis provides error estimates for an intermediate approximate adder placed within an accelerator or adder tree. Since design space exploration requires careful evaluation of multiple approximate adder variants for each adder node, the problem at hand is to devise an effective and practical methodology that systematically computes the cumulative error due to approximations at the output of the accelerator/adder tree.

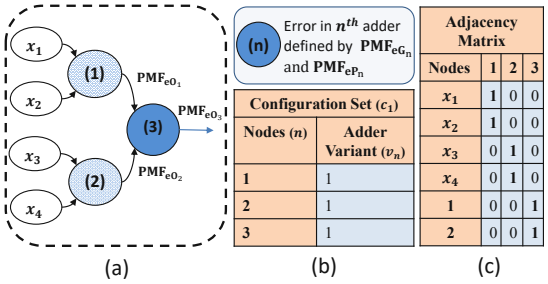


Figure 3.29: (a) Error generated and propagated by an n^{th} adder. Here, $n = 3$, i.e., the 3^{rd} adder is adding the sums obtained from the 1^{st} and 2^{nd} adders. (b) A configuration set (c_t) is defined using the type of approximate adder (adder variant) used at each adder node. In the illustrated example of Fig. (a), each adder is using the same type of approximate adder, i.e., Type 1. Thus, the adder variant 1 is mentioned for each adder node in Fig. (b). (c) Adjacency matrix provides the connectivity of the adder nodes.

3.8.3 Methodology

Building on the analysis presented in Sections 3.8.1 and 3.8.2, this section presents a systematic methodology for Data and Application-aware Error Analysis (DAEM) for an application kernel composed of $N-1$ adders for adding N operands. DAEM allows any adder variant (from the supported adders) to be evaluated for each of the $N-1$ adders. Note that, for a case with V adder variants, a total of $T = V^{(N-1)}$ configurations are possible. Thus, a t^{th} configuration set, c_t, can be specified by an array, c_t={v_1,v_2,...,v_{N-1}}, where $v_n \in \{1, 2, ..., V\}$ and represents the adder variant of the n^{th} enumerated adder. In general, DAEM requires sample data, application code and error maps (E_{MAP}) of the V approximate adder variants to provide an estimate of error metrics (e.g., PMF of error, MSE, and MED) for all the T configuration sets. Fig. 3.30 illustrates the complete flow of DAEM. The steps are explained in detail in the following paragraph.

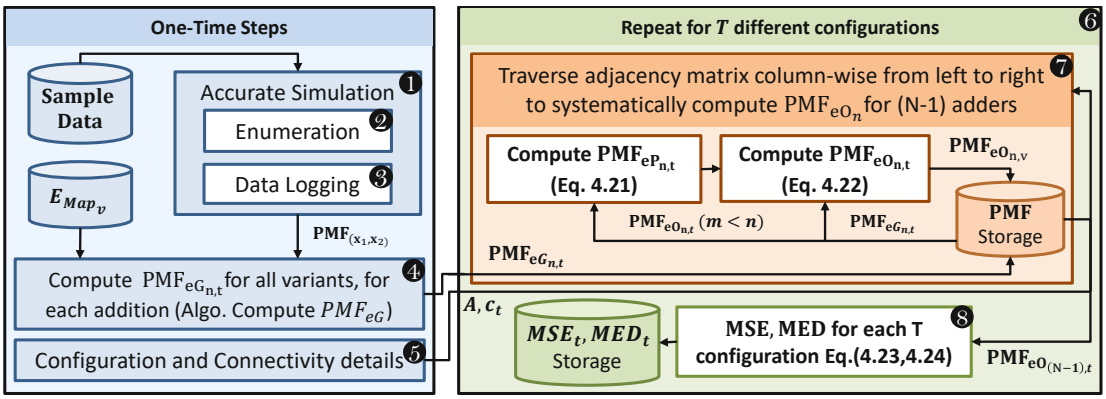


Figure 3.30: Proposed data and application-aware error analysis methodology for approximate adders.

First, a simulation of the accurate version of the application kernel is required using scenario-specific realistic sample data (Step 1) to generate 2D joint input probability distributions for all the adders in the application kernel. To achieve this, DAEM requires accurate simulation code for the application with two trivial modifications. Firstly, a code snippet is added that enumerates the adders from 1 to $N-1$ (Step 2 in Fig. 3.30) in their execution order (for example, see Fig. 3.29(a)). These node IDs are later used in Step 5 to generate an adjacency matrix (A), similar to the example shown in Fig. 3.29(c). Secondly, as the accurate simulation runs, the inputs to each adder have to be logged (Step 3) to build the corresponding 2D joint input probability distribution PMF_{X_1,X_2}. The 2D joint input probability distributions enable the generation of PMF_{eG} for each adder in each configuration set, i.e., $PMF_{eG_{n,t}}$, as explained in Section 3.8.1. The adjacency matrix (A), the computed $PMF_{eG}s$, and the list of configuration sets are passed to Step 6. Step 6 runs T times to compute the error metrics for T configurations. For each configuration, the adjacency matrix (A) is traversed column-wise from left to right and PMF_{eO_n} is evaluated (Step-7). $PMF_{eO_{N-1,t}}$, the PMF of output error of $N-1^{th}$ node, defines the PMF of the overall error of the accelerator, and hence, it is used in Step 8 to compute the MSE and MED error metrics of the t^{th} configuration set.

3.9 Usability and Effectiveness of DAEM

This section presents the evaluation results of DAEM achieved for multiple application scenarios. The scenarios include a few synthetic applications/data and a few representative kernels from image and video processing. For each of these cases, the estimated error measures are compared with those obtained from the state-of-the-art error estimation model of [MHH+17] and those obtained using functional simulations. For the analyses, different variants of GeAr adder [SAHH15] are considered as it is a well-cited and generic model that can be configured to mimic the functionality of a majority of the prominent state-of-the-art LLAAs including ACA-I, ACA-II, ETAII, and GDA.

3.9.1 Comparison with state-of-the-art [MHH+17]

First, the effectiveness of DAEM is evaluated for a synthetic scenario where error metrics of a standalone LLAA are estimated considering different input distributions. The evaluation is performed for five different configuration sets (T=5). The adder variants considered for this evaluation are: GeAr(8,1,4), GeAr(8,1,5), GeAr(8,2,2), GeAr(9,2,3) and GeAr(10,2,4). The configurations are illustrated in Fig. 3.31.

The three different input distributions considered are:

1. Uniformly distributed uncorrelated inputs.
2. Uncorrelated Gaussian distribution with $\mu = [127; 127]$ and $\Sigma = [10; 01] \times 10^3$.
3. Correlated Gaussian distribution with $\mu = [127; 127]$ and $\Sigma = [10.95; 0.951] \times 10^3$.

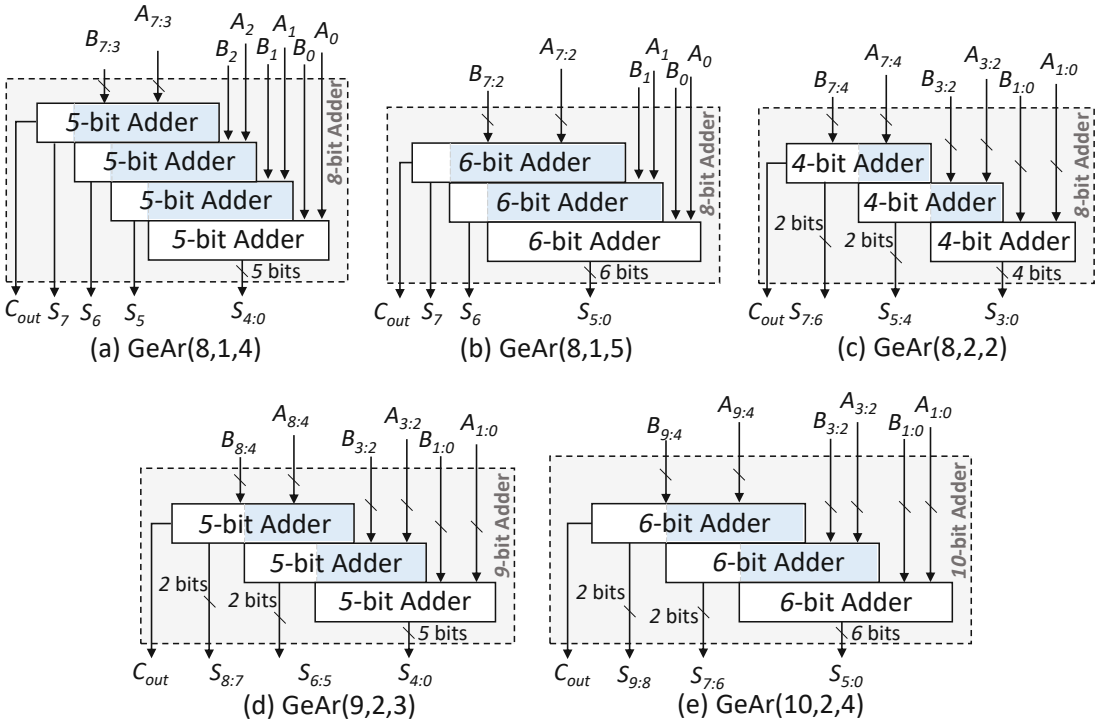


Figure 3.31: The configurations of the GeAr adder model used for evaluation using synthetic data.

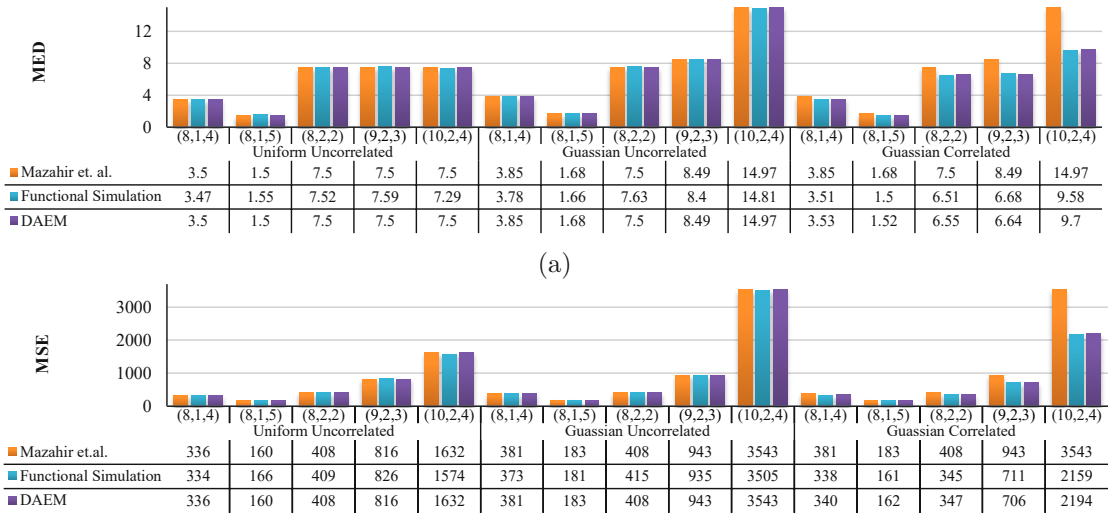


Figure 3.32: Comparison of DAEM, analytical model by Mazahir et al. [MHH+17] and functional simulations. The MSE and MED results are obtained for 6 different configurations of GeAr for three different input distributions. Each GeAr configuration is represented in its generic form (N, R, P).

Fig. 3.32 presents a comparison between the error measures estimated using DAEM and the analytical model of [MHH+17] for all three input distributions. Since DAEM operates on 2D joint input PMFs, these distributions can be directly employed to compute the PMF of error. However, since the analytical model of [MHH+17] operates only on 1D input PMFs, the marginal PMFs calculated using the 2D joint input distribution are used for computing the error estimates. To generate the reference values for the analysis, time-consuming Monte-Carlo simulations are performed. For these simulations, 100 000 input combinations are sampled from each of the above-mentioned three distributions to create three different datasets. These datasets are then used for functional simulations to compute the reference MSE and MED error measures.

The estimated MED (Fig. 3.32a) and MSE (Fig. 3.32b) for both the uncorrelated input distributions illustrate that the proposed model can offer accuracy equivalent to that of the state-of-the-art method [MHH+17]. In fact, the error estimated by both DAEM and [MHH+17] matches the results of the functional simulations for the uncorrelated input distributions. However, as illustrated in the previous sections, in some real-world applications, the inputs of adders are correlated and therefore resemble the Gaussian correlated distribution in the sense of correlation between inputs. For the Gaussian correlated case, DAEM generates accurate estimates of MSE and MED for all the considered configurations, while [MHH+17] fails to produce correct results. Thus, it can be said that DAEM outperforms state-of-the-art techniques by leveraging 2D joint input distribution.

To highlight the significance of DAEM for real-world applications, Fig. 3.33 presents a comparison between MSE values estimated using three different techniques for two different applications realized using approximate accelerators. The results are presented for three different datasets: Sky, Stadium, and Satellite. The applications considered are (1) low-pass filtering using a 4x4 kernel and (2) template matching using a 4x4 kernel. The adder tree used for this analysis is composed of approximate adders where GeAr(8,2,2) is used for the first stage additions, GeAr(9,2,3) is used for the second stage additions, GeAr(10,2,2) is used for the third stage additions, and GeAr(11,2,3) is used for the final stage. Each dataset contains 10 images of the corresponding landscape. Note that all these images are collected using Google Images and Google Maps. For DAEM, the required 2D joint input PMFs are computed for each case using 5 randomly selected images from the corresponding dataset, while the rest of the 5 images are used for generating the ground truth using functional simulations. Fig. 3.33 clearly shows that in all cases, the MSE estimated using DAEM is more realistic than the estimates generated by [MHH+17]. It can also be observed from the figure that the error estimates provided using DAEM follow the same trend as that of the functional simulations. Thus, it can be concluded that DAEM results in better error estimates than state-of-the-art approaches.

Although, the error measures computed using DAEM are better compared to the state-of-the-art method, the estimated values do not exactly match the results achieved through functional simulations. This deviation from the correct values can be attributed to two

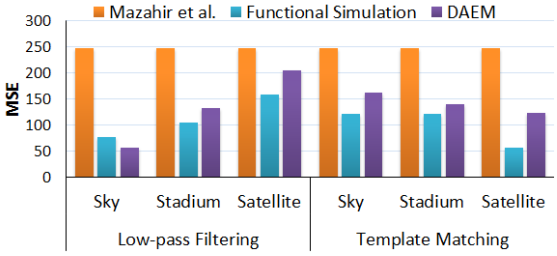


Figure 3.33: Comparison of Error in Estimated MSE for two image processing applications for three datasets.

main reasons:

1. Approximation errors in earlier stage adders may trigger changes in the joint input PMFs of the adders in the subsequent stages. Capturing these variations requires a significantly complex and compute-intensive method. Therefore, to keep the methodology lightweight, these variations are ignored.

2. In a real-world setting, the input distribution at test time can be significantly different from the distribution observed and used during designing (in Step 3 of Fig. 3.30) for error estimation.

3.9.2 DAEM in video processing

Videos captured in a static setting, such as surveillance videos, provide a common use case where the background is mostly static and hence the overall visual information across frames is almost the same. Because of this, a short chip acquired from such a static camera can be used as a representative sample for the whole video and hence can be used as training data for the DAEM methodology. Thus, the error measures predicted by DAEM over the training dataset are expected to stay the same for the testing data. To illustrate this, 4x4 average filtering is applied on each frame of a video using approximate adders. For this analysis, the *container* video, which is a standard input in the video processing domain and is available at http://trace.eas.asu.edu/yuv/, is considered. The video is captured using a static camera. For evaluation, per frame MSE for 10 different configuration sets (T=10) is computed for a total of 300 frames. Fig. 3.34 shows the results acquired for each configuration set. It can be observed from the figure that the relative standing of MSE for each configuration set is mostly consistent. *Thus, DAEM can be efficiently employed to relatively rank the suitability of a configuration set for approximation.* This can further be employed to perform the design space exploration and hence can aid in the selection of the most suitable configuration set.

To demonstrate the efficacy of DAEM for static videos, an average filtering application with a kernel size of 4x4 is considered. For this evaluation, 120 different configuration sets (T=120) are considered. Fig. 3.35 shows the MSE estimates acquired using DAEM and

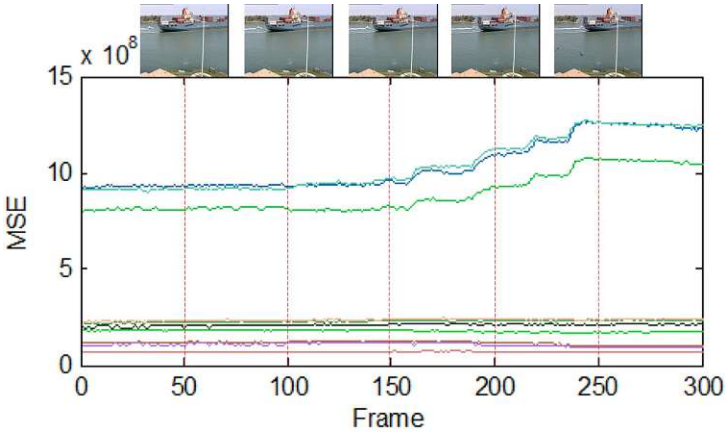


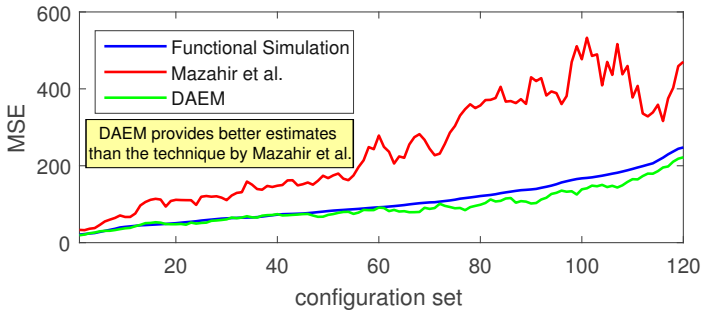
Figure 3.34: Variations in per frame MSE across frames of standard container video for 4x4 approximate low pass filtering application. The results are illustrated for 10 different configuration sets.

compares them with [MHH+17] and functional simulations. The results are generated for two standard videos ('Container' and 'Coastguard'). In each case, 5 initial frames of the video are used as the training set for DAEM to define the 2D joint input PMFs. The results are generated for frames 6-25. For illustrative purposes, the configuration sets are sorted based on their MSE values achieved through functional simulations and then plotted in the figure. *The figure clearly shows that the results of DAEM are almost in complete accordance with the simulated results and are better than the state-of-the-art approach.*

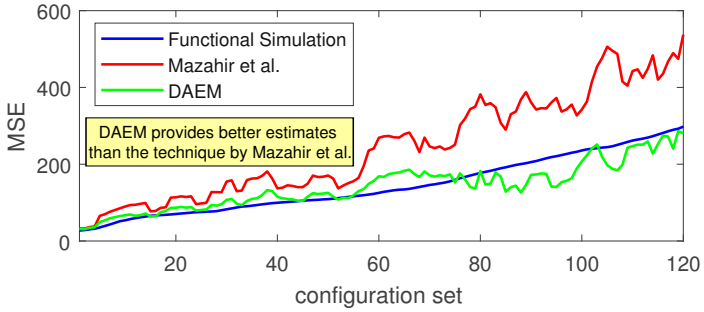
3.9.3 Time Requirements for Error Estimation using DAEM

DAEM is composed of two main computing blocks as shown in Fig. 3.30. The first block comprises Steps 1-5, which are required to be executed only once. Note, it is assumed that E_{MAP}s are generated beforehand as they only depend on the approximate adder variants. The simulations performed for generating joint input probability distributions take most of the time in this block. The second block comprises Steps 6-8. These steps are required to be repeated for each configuration set. Therefore, Steps 6-8 are executed T times in total; however, these steps are not computationally intensive and incur very low computational cost.

Table 3.6 presents a comparison between the time taken by DAEM, the method proposed by Mazahir et al. in [MHH+17] and exhaustive simulations to generate the low-pass filtering results presented in Fig. 3.35. For this comparison, T was assumed to be 120, i.e., the simulations were performed for 120 different configurations. It can be observed from the table that DAEM falls in between exhaustive simulations and [MHH+17], i.e., it requires very less amount of time compared to exhaustive simulations and much higher time compared to [MHH+17]. This is mainly because [MHH+17] does not involve any



(a)



(b)

Figure 3.35: Comparison of DAEM with state-of-the-art [MHH+17] and simulated results for 4x4 low-pass filtering application for two standard videos: (a) Container (b) Coastguard.

functional simulations while DAEM requires simulations in the earlier steps to build 2D joint input probability distributions for the adders. The table clearly shows that Steps 1-5 consume most of the execution time, i.e., 71.687 sec, while the time required for the remaining steps (6-8) to evaluate all T configuration sets is quite close to the time required for [MHH+17]. Hence, for very large T, DAEM can achieve execution time comparable to [MHH+17] while offering significantly better error estimates.

Table 3.6: Timing comparison of error estimation schemes with simulations (sec)

t	Mazahir [MHH+17]	Simulations	DAEM
10	0.005	18,416.00	71.687 + 0.006
30	0.015	55,245.00	71.687 + 0.018
60	0.031	110,497.00	71.687 + 0.036
90	0.047	165,746.00	71.687 + 0.054
120	0.062	220,995.00	71.687 + 0.073

3.10 Limitations of DAEM

The scalability of DAEM can be limited in certain scenarios, depending on the following two key factors:

1. Hardware accelerators are usually composed of cascaded modules, where the output of one module is used as input to the next. In such a case, approximations in earlier stages can affect the joint input probability distribution of later stages. Therefore, based on the level of approximations deployed in earlier stages, the estimated joint input probability distributions of the later stages can deviate from the estimated distributions (estimated through profiling), which indirectly affects the accuracy of error estimates. Hence, the proposed approach can be considered highly effective for shallow accelerators, i.e., accelerators having less number of cascaded stages, but may not produce reliable results for deeper datapaths where the number of cascaded stages is large.

2. The number of possible input combinations of a computing module increases exponentially with the increase in the input bit-width. This translates to huge memory requirements for the error maps and the joint input probability distributions and thereby limits the applicability of DAEM in such scenarios. For example, to store an error map of an 8-bit approximate adder in 8-bit integer format, a total of 2^{16} bytes (i.e., 64 KB) of memory is required; however, for a 16-bit adder, the memory requirement comes out to be 2^{32} bytes (i.e., 4 GB).

3.11 Open-source Libraries

To facilitate further research and development in the area of approximate computing and for the sake of reproducibility, the implementations of both *QuAd* and *PEMACx*, along with the supporting snippets, have been open-sourced and are available at `https://sourceforge.net/projects/quad-code/` and `https://github.com/mahanif/PEMACx`, respectively.

3.12 Chapter Summary

This chapter covered multiple error estimation and design space exploration methods for approximate adders. First, it presented *QuAd*, a configurable adder model that covers the entire design space of low-latency approximate adders. The analysis of *QuAd* configurations showed that, given a latency constraint, it is possible to effortlessly select the optimal low-latency approximate adder configuration ($QuAd_o$). Although $QuAd_o$ offers quality-area optimal designs, the design space covered by *QuAd* only includes low-latency adders. Therefore, to efficiently explore the design space of low-power adders, the chapter, after *QuAd*, presented PEMACx, a methodology to efficiently compute PMF of error of approximate low-power adders that are composed of smaller cascaded

approximate adder units. Note, all types of error metrics such as MSE, MED, and ER can be computed using the PMF of error. Hence, PEMACx enables the users to efficiently quantify the quality of low-power approximate adders based on the user-defined error metric. The chapter demonstrated the effectiveness of the proposed methodology for analyzing the error characteristics of low-power, as well as a class of high-performance (low-latency), adders. Further, the chapter highlighted that the input data distribution could have a significant impact on the output quality of an approximate adder. Therefore, in cases where the input distribution is not uniform or the input bits cannot be assumed independent of each other, a data-driven approach is necessary to accurately estimate the quality of an approximate module. Towards this, the chapter presented DAEM, a data and application-driven error estimation methodology. The evaluation showed that DAEM is suitable for shallow datapaths and cannot produce highly accurate results for deeper circuits. Hence, it is useful for simple audio-visual applications such as filtering but cannot be used for large dot-product operations involved in, for example, DNNs. This points towards the significance of simulation-based evaluation of complex applications and also the need for specialized approximation/optimization techniques that can effectively trade application-level accuracy for efficiency without involving intermediate error estimation.

CHAPTER 4

Cross-Layer Optimizations for Deep Neural Networks

As highlighted in Chapter 2, different hardware-level and software-level techniques have been proposed in the literature to improve the energy efficiency of DNNs. To significantly benefit from the error-resilience of DNNs, these techniques can be integrated in a systematic manner. Towards this, this chapter first presents a methodology that combines the most effective hardware-level and software-level approximation/optimization techniques in a cross-layer framework (see Section 4.1). Then, multiple case studies are presented in Section 4.2 that highlight the effectiveness of each individual technique employed in the framework as well as their combined benefits.

4.1 Cross-Layer Optimization Framework for DNNs

Exploiting a single approximation type at a single abstraction layer results in limited benefits. Multiple types of approximations from different abstraction layers can be combined to achieve significant efficiency gains. Fig. 4.1 presents a novel cross-layer approximation methodology for DNNs that combines effective software-level and hardware-level techniques in a synergistic manner. The methodology is composed of the following steps:

- **Pruning:** At the software level, the most effective technique used for optimizing DNNs is pruning. Therefore, Step 1 of the methodology employs *iterative* pruning to eliminate ineffectual weights from the DNNs. In iterative pruning algorithms, first the saliency of each weight/filter in the given DNN is computed using its L1 or L2-norm, and then, a small set of least significant weights/filters are pruned from the network. The pruned network is then (optionally) fine-tuned for a limited number of epochs to regain a fraction of the lost accuracy. This process is repeated

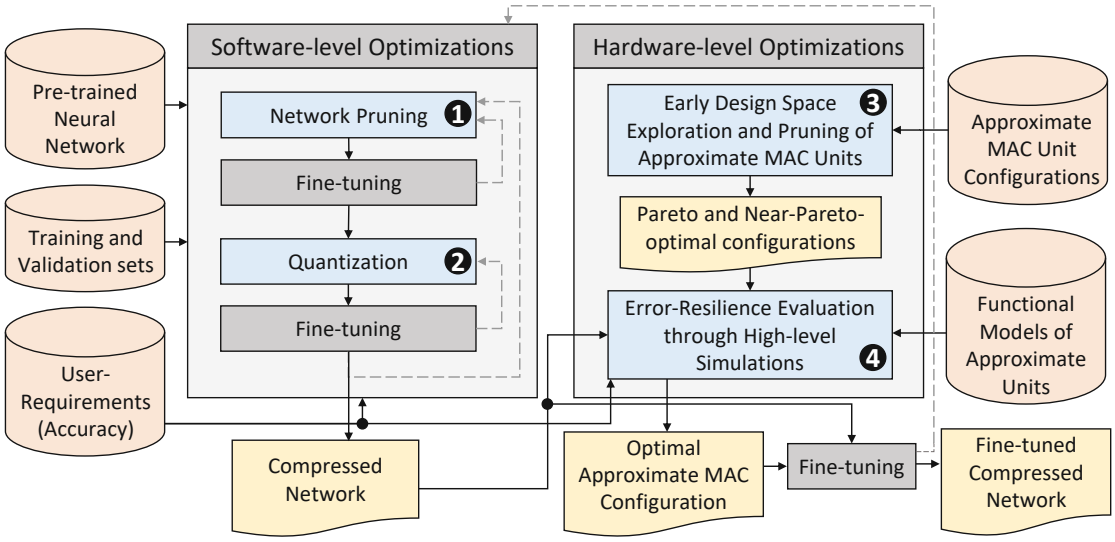


Figure 4.1: Our Cross-Layer Optimization Flow for DNNs.

until a point is reached where any further pruning results in accuracy drop below the user-defined accuracy constraint. Note, iterative pruning is preferred over one-shot pruning because it reevaluates the significance of the remaining weights/filters after every pruning step and, thereby, results in more reduction in the network size and computational complexity compared to one-shot approaches (e.g., [LAT18]).

- **Quantization:** Once the pruning is complete, Step 2 performs a search for an effective quantization policy to reduce the bitwidth of the weights and the activations of each layer of the DNN. Alongside reducing the size of DNNs, quantization also enables the use of simplified logic units at the hardware level. The quantization process can also (optionally) be followed by fine-tuning to adapt to the quantization error introduced due to the limited precision of weights and activations. Note, both pruning and quantization can be combined in a single unified framework to achieve optimal results. However, such a framework requires sophisticated optimization algorithms to explore the combined optimization space (for example, see CLIP-Q [TM18]) and, therefore, result in high complexity of search.

- **Hardware-Level Approximations:** At the hardware level, Step 3 explores the design space of approximate units, such as approximate adders and multipliers, to select the configurations that offer the best quality-efficiency trade-offs. The considered design space also includes configurations/modules that exhibit internal self-healing characteristics [GHV+19]. Once the design space of the approximate units has been reduced, Step 4 employs behavioral simulations using functional implementation of the approximate units to accurately quantify the impact of approximations on the functionality of the input DNN. Based on the results, the

configuration that offers the optimal savings while meeting the accuracy constraint is selected. This step can also (optionally) be followed by fine-tuning to partially compensate for the lost accuracy due to hardware-level approximations.

4.2 Case Studies for Improving Energy and Performance Efficiency of DNN Implementation

4.2.1 Structured Pruning

This section presents a novel methodology for pruning filters/neurons from a pre-trained DNN. Fig. 4.2 illustrates the overall flow of the methodology. The methodology receives a pre-trained DNN, a user-defined accuracy constraint, and a user-defined cost function as inputs. The cost function is used to compute the relative importance of filters/neurons within the network, which is then used to identify the least significant filters/neurons that have minimal impact on the model's accuracy while offering significant model size reduction (or efficiency gain). Some possible cost functions are listed in Table 4.1. The main steps of the methodology are as follows:

1. First, the methodology computes the significance of all the filters/neurons in each layer of the given DNN using a suitable saliency measure, e.g., L1-norm, as adopted in this study.
2. Then it creates as many copies of the DNN as the *number of layers* - 1. Then, for each copy l, where $l \in \{$*the set of all the layers except the last layer of the DNN*$\}$, it keeps all except the l^{th} layer intact and removes $x\%$ of the least significant filters/neurons from the l^{th} layer.
3. The accuracy for each pruned network is computed using the validation dataset and registered in θ alongside the total number of parameters of the corresponding layer.
4. Once the accuracy for all the created networks have been registered, the cost of each layer is computed using the user-defined cost function (C) and the values registered in θ.
5. The networks are then sorted based on their costs and the DNN which incurs the least cost is kept while all the rest are discarded.
6. The selected network is then fine-tuned for y number of epochs and its accuracy is estimated using the validation dataset.
7. The estimated accuracy is then compared with the user-defined accuracy constraint (A_c). If the network meets the constraint, the pre-trained network is replaced with the pruned network and the complete process is repeated. Otherwise, if the pruned DNN does not meet the constraint, the pruned DNN is discarded and the (updated) pre-trained model is forwarded as the output of the algorithm.

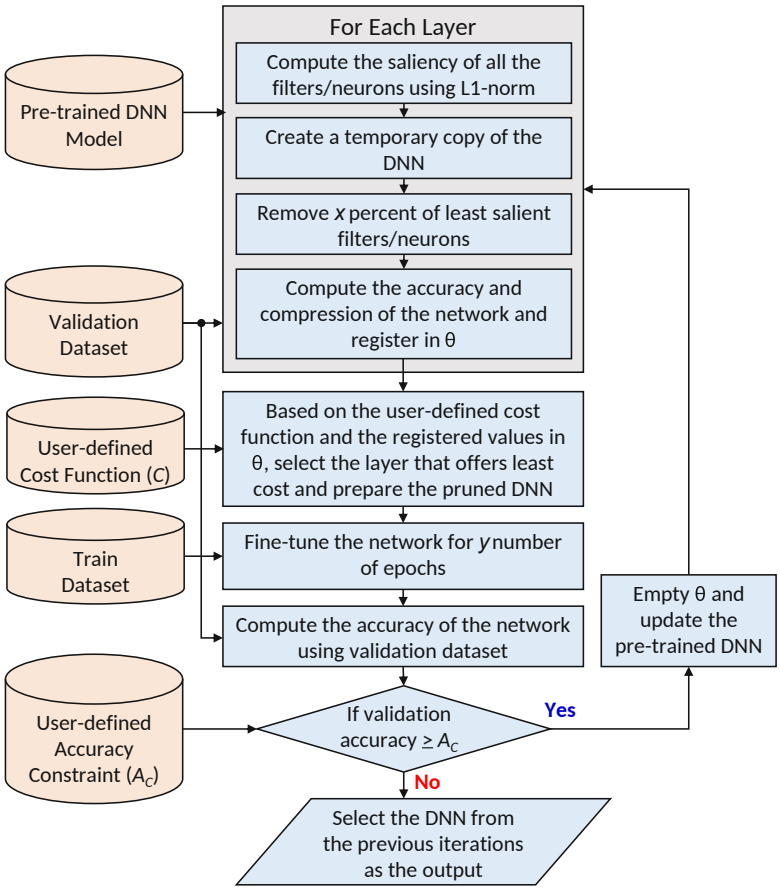


Figure 4.2: Proposed structured pruning methodology

Experimental setup: To illustrate the effectiveness of the proposed method, it is evaluated for two different DNNs, LeNet5 and VGG11, using two different datasets, MNIST and Cifar10. The details of the settings used for experiments are summarized in Table 4.1.

Impact of structured pruning on DNN accuracy and network size: Fig. 4.3 presents the results obtained by applying the proposed methodology on the LeNet5 network trained on the MNIST dataset. Figs. 4.3a, 4.3b and 4.3c are generated using cost functions C_A, C_B and C_c (listed in Table 4.1), respectively. Different cost functions are employed to analyze the impact of varying relative weights of the number of parameters in a layer and the DNN accuracy after pruning (before any fine-tuning). As can be seen from the figures, the network maintains its baseline accuracy until it is significantly pruned, i.e., until around 90% reduction in the network size. After the 90% mark, the DNN becomes increasingly sensitive to pruning, and this trend is consistent across all the considered cost functions. However, employing intermediate fine-tuning (i.e., setting $y > 0$) can help in maintaining the baseline accuracy till even higher percentage reduction.

Table 4.1: Settings used in the experiments

	Network and Dataset	Network Architecture	%age of filters pruned per iteration (x)	Cost function (C)	Number of epochs (y)
1	LeNet-5 for MNIST	CONV(6, 1, 5); CONV(16, 6, 5); FC(120, 400); FC(84, 120); FC(10, 84)	20	$C_A = 100 - Accuracy$	0, 1, 2
				$C_B = 100 - (Accuracy + P_i/\sum_{i\in\{alllayers\}} P_i)$	0, 1, 2
				$C_C = 100 - (Accuracy + 4 * P_i/\sum_{i\in\{alllayers\}} P_i)$	0, 1, 2, 4
2	LeNet-5 for Cifar10	CONV(6, 3, 5); CONV(16, 6, 5); FC(120, 400); FC(84, 120); FC(10, 84)	20	$C_A = 100 - Accuracy$	0, 1, 2
				$C_B = 100 - (Accuracy + P_i/\sum_{i\in\{alllayers\}} P_i)$	0, 1, 2
				$C_C = 100 - (Accuracy + 4 * P_i/\sum_{i\in\{alllayers\}} P_i)$	0, 1, 2, 4
3	VGG11 for Cifar10	CONV(64, 3, 3); CONV(128, 64, 3); CONV(256, 128, 3); CONV(256, 256, 3); CONV(512, 256, 3); CONV(512, 512, 3); CONV(512, 512, 3); CONV(512, 512, 3); FC(10, 512)	20	$C_A = 100 - Accuracy$	0, 1, 2
				$C_B = 100 - (Accuracy + P_i/\sum_{i\in\{alllayers\}} P_i)$	0, 1, 2
				$C_C = 100 - (Accuracy + 4 * P_i/\sum_{i\in\{alllayers\}} P_i)$	0, 1, 2, 4

This can be observed by comparing the $y = 0$ cases with the corresponding $y > 0$ cases.

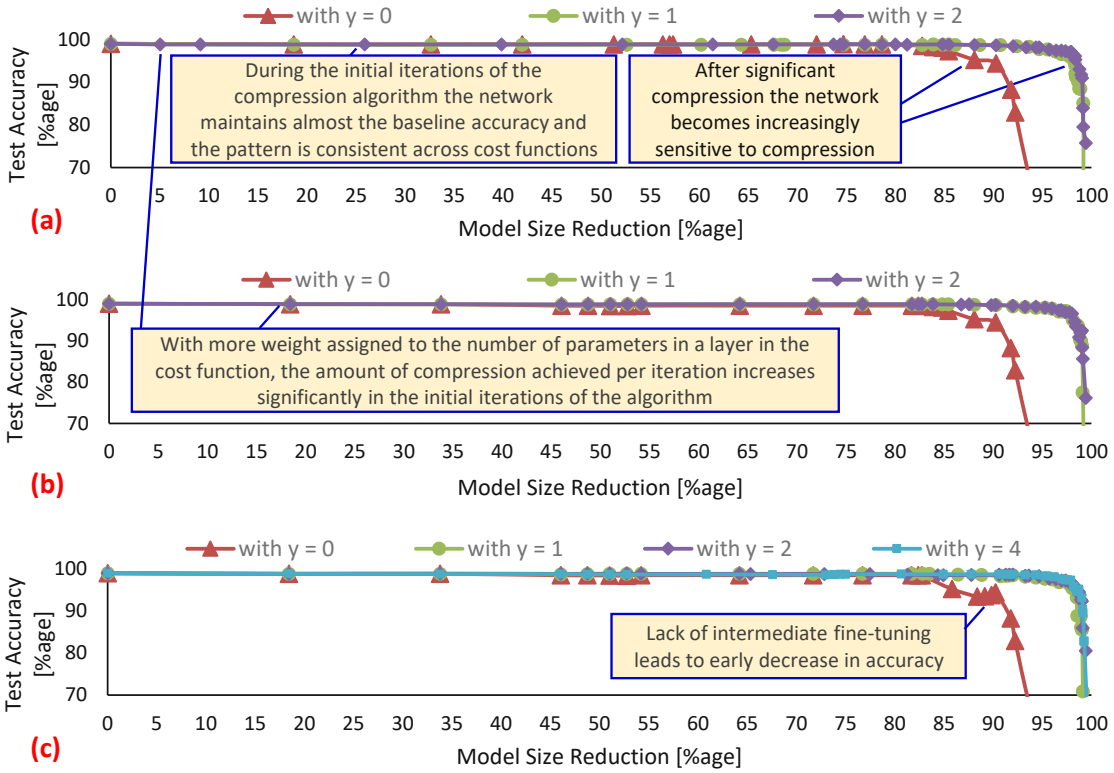


Figure 4.3: Results of structured pruning when applied to the LeNet5 network trained on the MNIST dataset. The sub-figures are generated using different cost functions, i.e., (a) C_A, (b) C_B, and (c) C_C, and different number of intermediate fine-tuning epochs (i.e., y) mentioned in Table 4.1.

Impact of using different cost functions: By comparing Fig. 4.3a with 4.3b, it can be observed that having a (non-zero) *factor for the number of parameters in a layer* in the cost functions (see equations of C_A, C_B and C_C in Table 4.1) results in relatively higher

compression in the initial iterations of the methodology. Therefore, if the methodology can be executed only for a limited number of iterations due to limited computational resources, a cost function with a higher *factor for the number of parameters* is preferable. However, it should be noted that assigning a larger weight value to *the number of parameters* in the cost function may lead to undesirable accuracy degradation, specifically during the final few iterations, or when used for a simple network trained on a relatively complex dataset, as can be observed by comparing Fig. 4.4a with Figs. 4.4b and 4.4c.

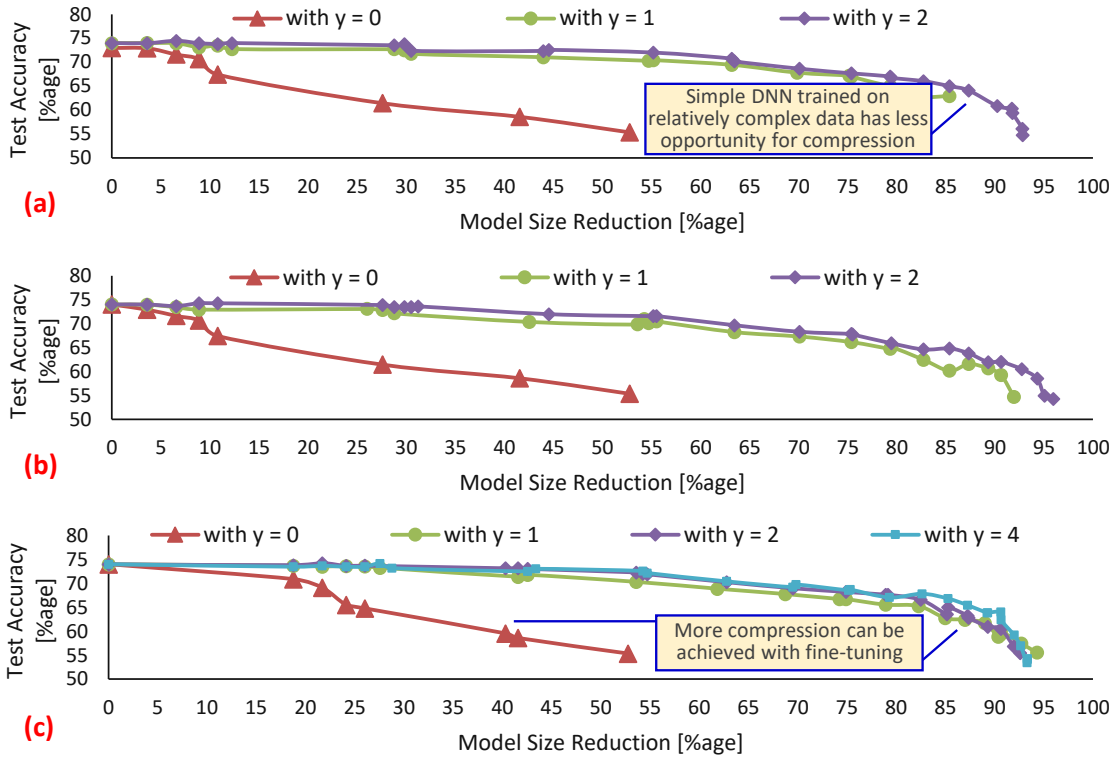


Figure 4.4: Results of structured pruning when applied to the LeNet5 network trained for the Cifar10 dataset. The sub-figures are generated using different cost functions, i.e., (a) C_A, (b) C_B, and (c) C_C, and different number of intermediate fine-tuning epochs (i.e., y) mentioned in Table 4.1.

Impact of dataset complexity on the reduction in DNN size through pruning: By comparing the results presented in Fig. 4.3 and Fig. 4.4, it can be observed that the pruning methodology is not as effective in case of the LeNet5 trained on the Cifar10 dataset as it is for the LeNet5 trained on the MNIST dataset. This is mainly due to the difference in the complexity of the two datasets. As can be seen in Fig. 4.5, the MNIST dataset is composed of black-and-white images of handwritten digits while the Cifar10 dataset is composed of colored images of objects. In general, more parameters are required to learn complex patterns than to learn simple patterns. This is why the DNN accuracy drops abruptly for the LeNet5 network trained on the Cifar10 with the

reduction in network parameters.

Figure 4.5: A few example images from: (a) the MNIST dataset; and (b) the Cifar10 dataset.

Impact of network complexity on the reduction in DNN size through pruning:

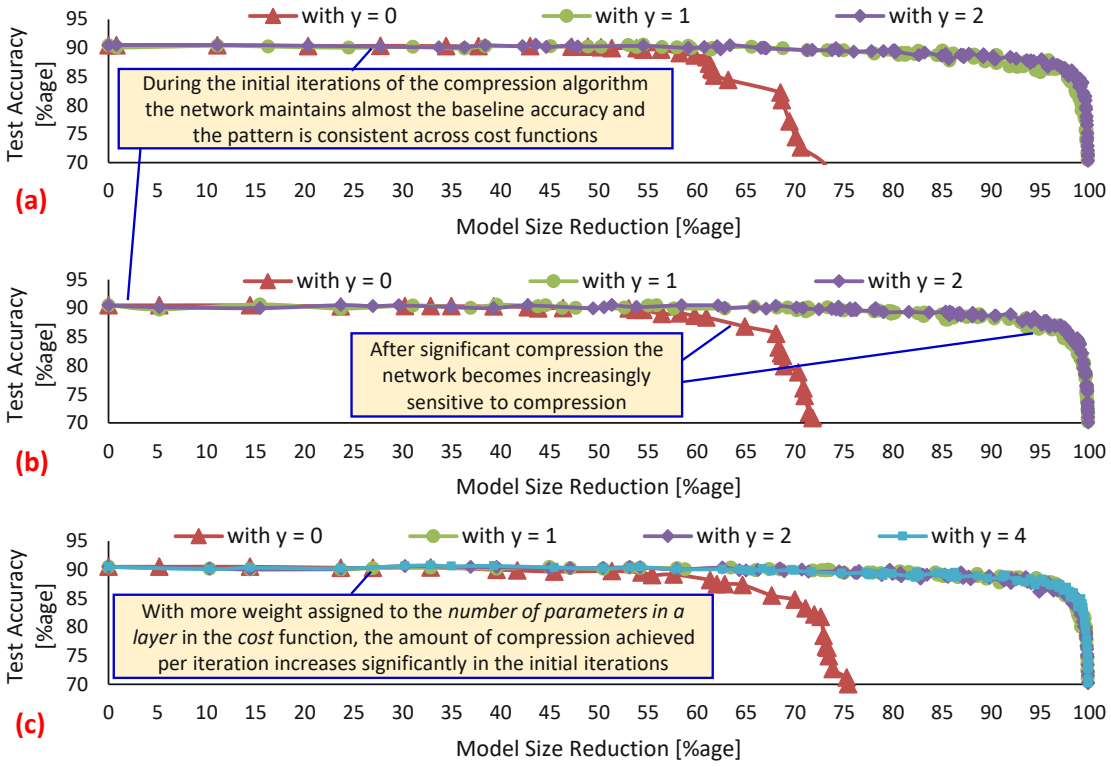


Figure 4.6: Results of structured pruning when applied to the VGG11 network trained on the Cifar10 dataset. The sub-figures are generated using different cost functions, i.e., (a) C_A, (b) C_B, and (c) C_C, and different number of intermediate fine-tuning epochs (i.e., y) mentioned in Table 4.1.

To analyze the effectiveness of the proposed technique for larger networks, two different DNNs, LeNet5 and VGG11, trained on the Cifar10 dataset are considered. Fig. 4.4 shows the results of the proposed technique when applied to the LeNet5 network, and Fig. 4.6 shows the results of the technique when applied to the VGG11 network. By comparing Fig. 4.4 and Fig. 4.6, it can be observed that the proposed technique (with $y > 0$) achieved higher compression for the VGG11 network (i.e., around 95%) than for the LeNet5 network (i.e., around 30%) while maintaining their respective baseline accuracy. From this, it can be concluded that the proposed technique helps in reducing the non-essential parameters, and it is more effective for larger networks that have higher number of non-essential parameters.

4.2.2 Quantization

Quantization is another technique besides pruning that is highly effective for reducing the size of DNNs. Apart from DNN size reduction, it also enables the use of simplified hardware modules and thereby offers the opportunity to further increase the energy efficiency of DNN implementations. As most embedded systems support 8-bit precision or higher, post-training quantization techniques that do not incur additional training/fine-tuning costs are sufficient to convert the models to low-precision variants. Therefore, in this study, only the post-training quantization approaches are considered, with the restriction that all the layers have the same bitwidth. To quantize the weights of a layer, the following equations is employed:

$$\hat{W_i^{<l>}} = round(W_i^{<l>} \times W_{scale}^{<l>}) \tag{4.1}$$

$$W_{scale}^{<l>} = 2^{floor(log_2(\frac{2^{n-1}-1}{max(abs(W^{<l>}))}))}$$

where $W^{<l>}$ is the set of all the weights in the l^{th} layer of the DNN, $W_i^{<l>}$ is the i^{th} element in $W^{<l>}$, $\hat{W^{<l>}}$ defines quantized weights, $W_{scale}^{<l>}$ is the scale for converting the weights and n represents the bitwidth.

To quantize the activations, first, the activation values generated using a small subset of sample images from the training set are profiled. Then, the activation scale is defined using the following equation:

$$A_{scale}^{<l>} = 2^{floor(log_2(\frac{2^{n-1}-1}{max(abs(A^{<l>}))}))}$$

where $A^{<l>}$ is the set of all the logged activations from the input of the l^{th} layer and $A_{scale}^{<l>}$ defines the activation scale. During the run time, the activations are scaled using the following equation:

$$\hat{A_i^{<l>}} = floor(A_i^{<l>} \times A_{scale}^{<l>}) \tag{4.2}$$

where $\hat{A^{<l>}}$ defines the quantized activations. Note that $W_{scale}^{<l>}$ and $A_{scale}^{<l>}$ are intentionally kept in the power-of-2 format in order to simplify the intermediate conversion of activation values between the layers. The power-of-2 format transforms the multiplication operations to shift operations and thereby enables cost-efficient implementation of dynamic fixed-point representation.

Fig. 4.7b shows the accuracies of five different variants of the LeNet5 network, having different pruning ratios, when exposed to different levels of quantization. The variants are marked with the help of labels in Fig. 4.7a. By analyzing Fig. 4.7b, it can be observed that the variants that are more pruned are more sensitive to quantization. It can also be observed that the accuracy of all the networks drops sharply after a specific quantization level, which is the same for all the networks. The same trend is observed for the LeNet5 and the VGG11 networks trained with the Cifar10 dataset, as can be seen in Figs. 4.8 and 4.9, respectively. From this analysis, it can be deduced that (1) quantization indeed can offer benefits without significant loss in accuracy, (2) aggressive quantization leads undesirable accuracy degradation, and (3) higher pruning levels are more beneficial than quantization.

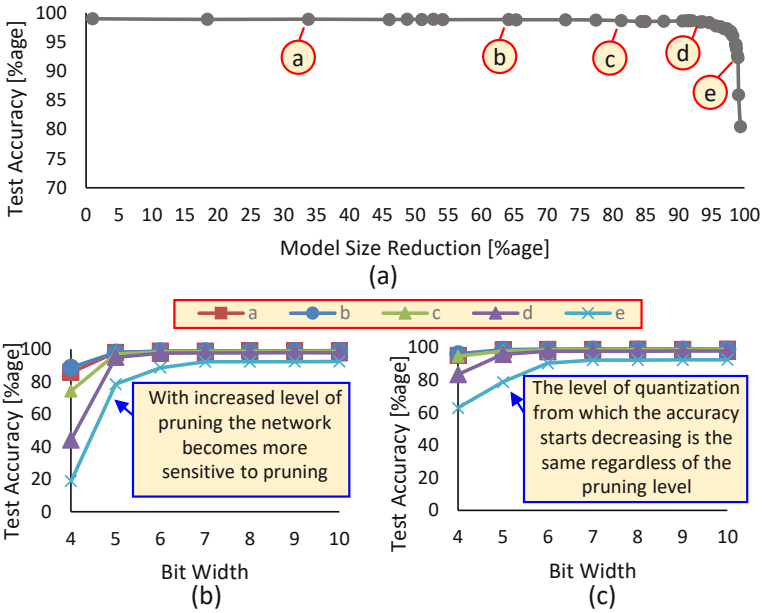


Figure 4.7: Network compression (structured pruning followed by quantization) results for the LeNet5 trained on the MNIST dataset. (a) Network compression through structured pruning. (b) Quantization of different pruned DNNs marked in (a) using Eqs. 4.1 and 4.2. (c) Quantization of the DNNs marked in (a) using Eqs. 4.1 and 4.3.

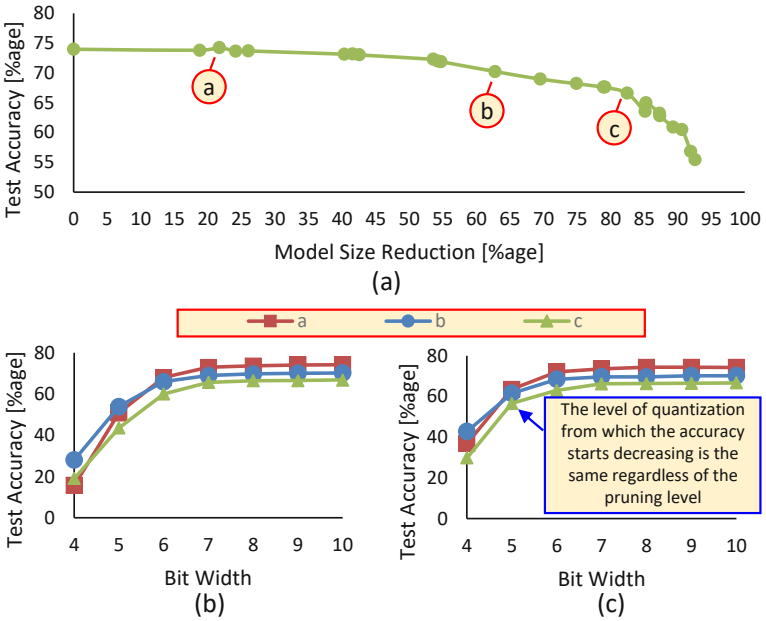


Figure 4.8: Network compression (structured pruning followed by quantization) results for the LeNet5 trained on the Cifar10 dataset. (a) Network compression through structured pruning. (b) Quantization of different pruned DNNs marked in (a) using Eqs. 4.1 and 4.2. (c) Quantization of the DNNs marked in (a) using Eqs. 4.1 and 4.3.

Besides Eq. 4.2 for evaluation, the impact of using Eq. 4.3 for quantizing the activation values on the accuracy of DNNs is also evaluated.

$$\hat{A_i^{<l>}} = round(A_i^{<l>} \times A_{scale}^{<l>}) \tag{4.3}$$

The results generated using Eq. 4.3 are shown in Figs. 4.7c, 4.8c, and 4.9c. By comparing Figs. 4.7b, 4.8b, and 4.9b with Figs. 4.7c, 4.8c, and 4.9c, it can be observed that using Eq. 4.3 leads to better accuracy results, mainly due to the use of *rounding* operation instead of *flooring* operation, as it leads to less quantization error.

4.2.3 Hardware-level Approximations: Impact of self-healing and non-self-healing designs on DNN Accuracy

This section analyzes the impact of using different types of approximate arithmetic modules on the accuracy of pruned and quantized networks. Mainly, two different types of approximate modules are considered: (1) modules designed using conventional approaches, and (2) modules designed using self-healing-based approaches. The main difference between these approaches is highlighted in Fig. 4.10. Fig. 4.10a shows the conventional method of deploying approximation in a system, where each individual

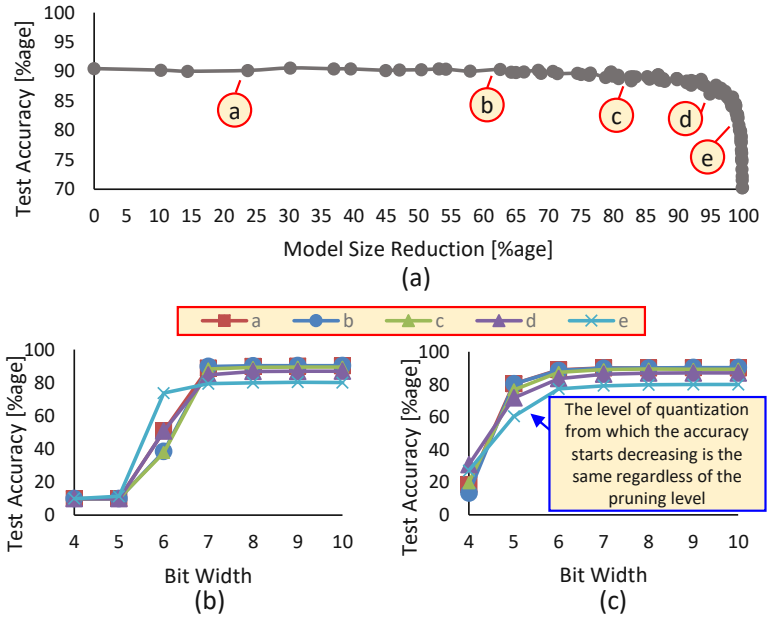


Figure 4.9: Network compression (structured pruning followed by quantization) results for the VGG11 trained on the Cifar10 dataset. (a) Network compression through structured pruning. (b) Quantization of different pruned DNNs marked in (a) using Eqs. 4.1 and 4.2. (c) Quantization of the DNNs marked in (a) using Eqs. 4.1 and 4.3.

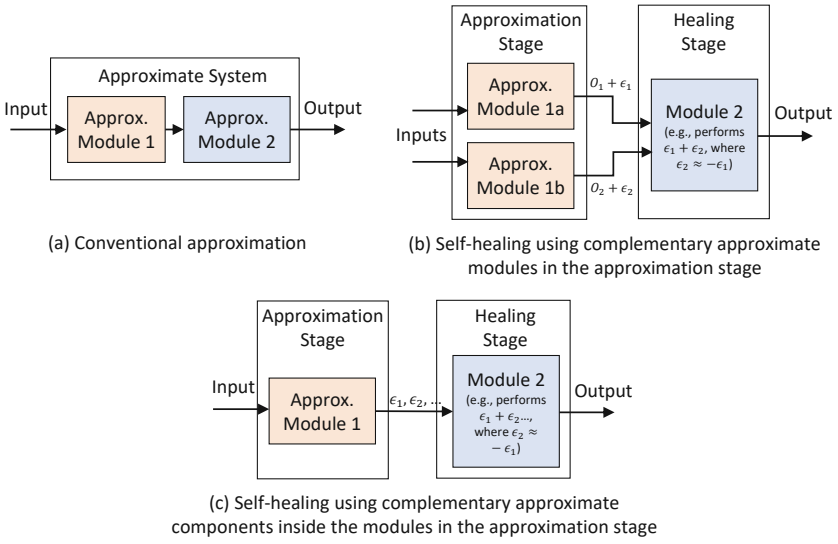


Figure 4.10: A comparison between conventional and self-healing approaches

module is approximated (without considering the overall computational flow) to trade-off accuracy for efficiency. The key drawback of this approach is that, in order to meet a user-defined quality constraint, the level of approximation that can be introduced in a system is limited. Therefore, such techniques cannot offer significant power/performance gains. To overcome this limitation, approaches for approximating systems using complementary approximate modules [GHK+18, GHV+19] have been introduced. The complementary modules are selected/designed such that they negate each other's adverse effects. This

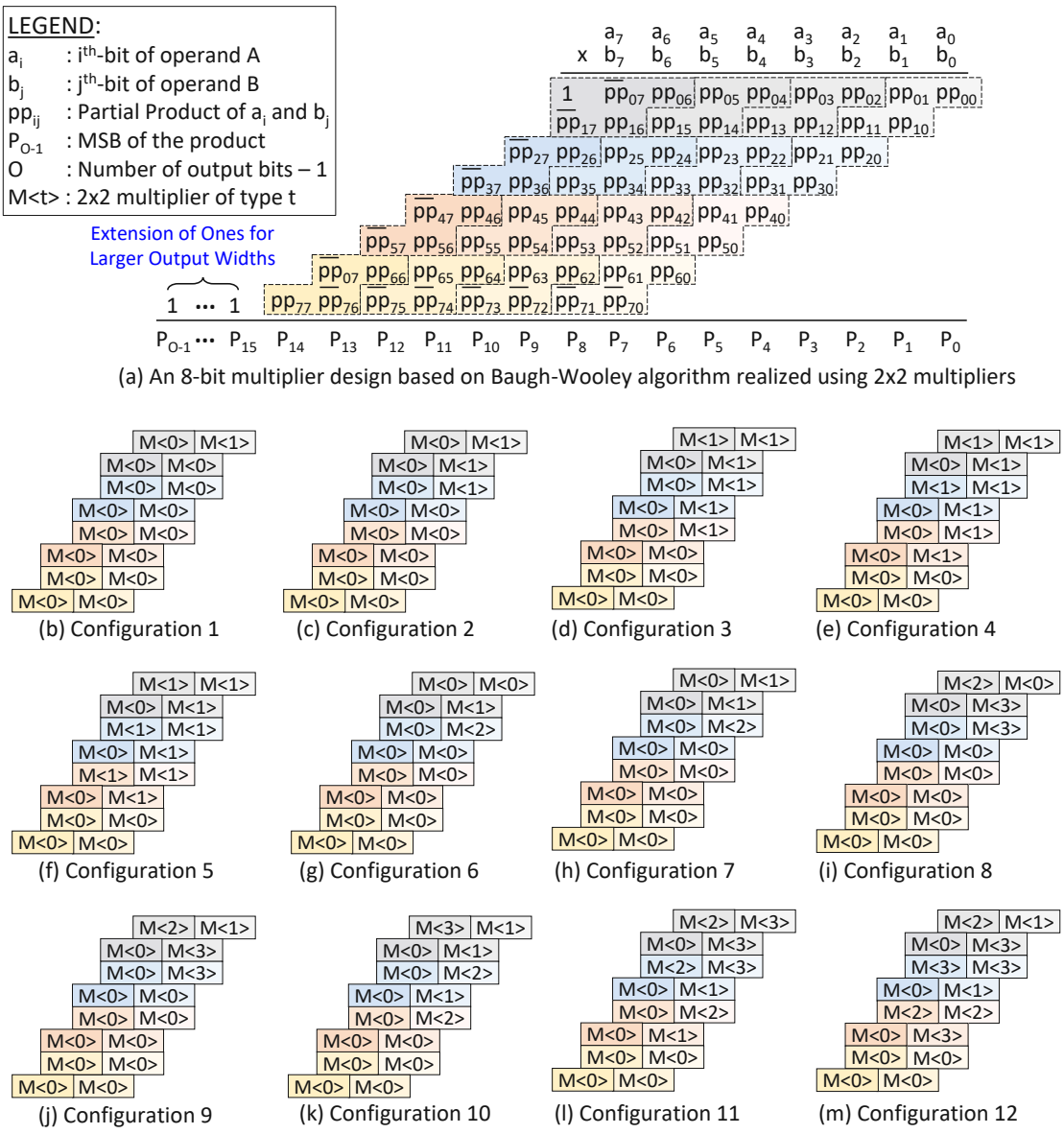
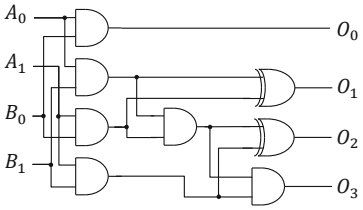


Figure 4.11: Types of 8x8 approximate multipliers considered for simulations

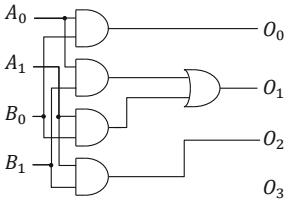
approach enables better quality-efficiency trade-off compared to conventional approaches. Figs. 4.10b and c show two different methods of how this concept can be realized. Fig. 4.10b shows two complementary approximate modules in the first stage and treats the second stage as the healing stage. Fig. 4.10c shows a single approximate module with internal complementary modules and treats the second stage as the healing stage. Note that both of these systems require a healing stage to recover from approximation errors, therefore, this concept can only be applied to cases where it is possible to have a healing stage at the end, e.g., in dot-product engines.



(a) Accurate 2x2 multiplier: M<0>

A \ B	00	01	10	11
00	0000	0000	0000	0000
01	0000	0001	0010	0011
10	0000	0010	0100	0110
11	0000	0011	0110	1001

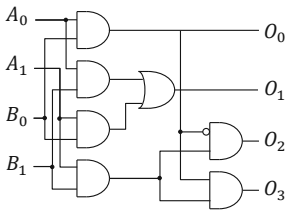
(e) Truth table of M<0>



(b) Approximate 2x2 multiplier having 3x3→7: M<1>

A \ B	00	01	10	11
00	0000	0000	0000	0000
01	0000	0001	0010	0011
10	0000	0010	0100	0110
11	0000	0011	0110	0111

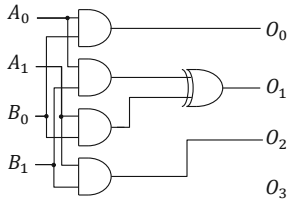
(f) Truth table of M<1>



(c) Approximate 2x2 multiplier having 3x3→11: M<2>

A \ B	00	01	10	11
00	0000	0000	0000	0000
01	0000	0001	0010	0011
10	0000	0010	0100	0110
11	0000	0011	0110	1011

(g) Truth table of M<2>



(d) Approximate 2x2 multiplier having 3x3→5: M<3>

A \ B	00	01	10	11
00	0000	0000	0000	0000
01	0000	0001	0010	0011
10	0000	0010	0100	0110
11	0000	0011	0110	0101

(h) Truth table of M<3>

Figure 4.12: The foremost 2x2 multiplier designs used for building conventionally approximate and self-healing approximation-based multipliers

Table 4.2: Error characteristics of the multiplier configurations presented in Fig. 4.11

	Multiplier Configurations											
	1	2	3	4	5	6	7	8	9	10	11	12
MSE	0.25	9.75	266.25	3102.30	24806.00	7.50	7.75	78.00	78.25	404.25	2128.00	18065.00
MED	0.13	1.13	7.13	23.13	55.13	0.94	1.02	3.38	3.44	8.29	19.94	53.67
Mean Error	-0.13	-1.13	-7.13	-23.13	-55.13	0.00	-0.13	0.00	-0.13	-4.13	-0.25	-0.13

Table 4.3: Hardware characteristics of the multiplier configurations presented in Fig. 4.11

	Multiplier Configurations												
	Accu.	Ax.1	Ax.2	Ax.3	Ax.4	Ax.5	Ax.6	Ax.7	Ax.8	Ax.9	Ax.10	Ax.11	Ax.12
Area [Cell Area]	753	716	696	616	609	571	726	711	727	701	665	672	654
Power [μW]	46.0	45.0	44.9	40.8	41.0	39.0	45.5	44.8	45.1	43.8	43.2	43.5	42.9
Delay [ns]	1.92	1.86	1.73	1.73	1.73	1.73	1.95	1.77	1.87	1.73	1.74	1.73	1.73
PDP [fJ]	88.4	83.7	77.7	70.6	70.9	67.4	88.7	79.3	84.2	75.8	75.2	75.2	74.3

As the dot-product operation is one of the fundamental operations in DNNs, the self-healing approximation concept can be applied by approximating the multiplication operations while keeping the addition operation accurate. Fig. 4.11a shows the baseline 8x8 multiplier design considered in this work. The design is based on Baugh-Wooley algorithm [BW73a]. The multiplier is implemented mainly using 2x2 multipliers. Fig. 4.12a shows the accurate 2x2 multiplier design while Figs. 4.12b, 4.12c, and 4.12d show the approximate variants that are used to implement approximate 8x8 multiplier configurations shown in Fig. 4.11. The 2x2 multiplier designs shown in Fig. 4.12b and 4.12d approximate 3×3 to 7 and 5 (i.e., less than 9), respectively, while the design in Fig. 4.12c approximates 3×3 to 11 (i.e., greater than 9). Figs. 4.11b-4.11m shows all the different approximate 8x8 multiplier configurations considered for evaluation in this work. The error characteristics of these configurations are presented in Table 4.2. Note, for this work we assume that the same type of multiplier is used for all the multiplication operations in a DNN inference. Figs. 4.11b-4.11f illustrate the configurations that generate uni-directional errors, i.e., only negative errors. These configurations correspond to designs generated using conventional approaches. Figs. 4.11g-4.11m illustrate configurations that generate bi-directional errors, as these configurations are composed of both types of approximate modules, i.e., the ones that generate positive errors and the ones that generate negative errors. These configurations correspond to designs generated using self-healing approaches. The hardware characteristics of all the configurations are presented in Table 4.3. These results are obtained by implementing the multipliers in Verilog and synthesizing using Cadence Genus for 65 nm technology node using TSMC 65 nm library.

Fig. 4.13 shows the results when the approximate multiplier configurations shown in Fig. 4.11 are used for different variants of the LeNet5 network having different pruning ratios. The pruning and accuracy characteristics of the considered networks are marked with the same labels in Fig. 4.7. As can be seen in Fig. 4.13, with the increase in the number of pruning ratio, the model becomes increasingly sensitive to approximations. Similar results are observed for other cases in Figs. 4.14 and 4.15. From Fig. 4.15, it

can also be observed that even moderate level of approximations in multipliers can significantly degrade the accuracy of a DNN. This is mainly due to the fact that the impact on the accuracy depends also on the data distribution.

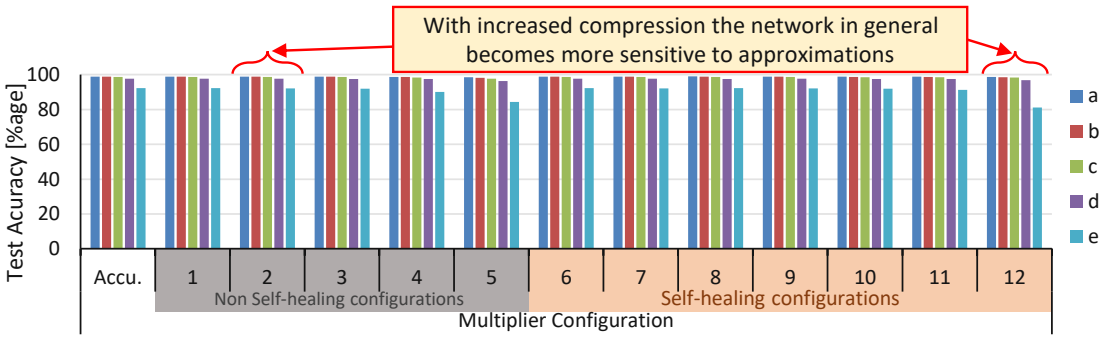


Figure 4.13: Effects of using approximate multipliers for inference of MNIST test images using different compressed LeNet5 variants marked in Fig. 4.7.

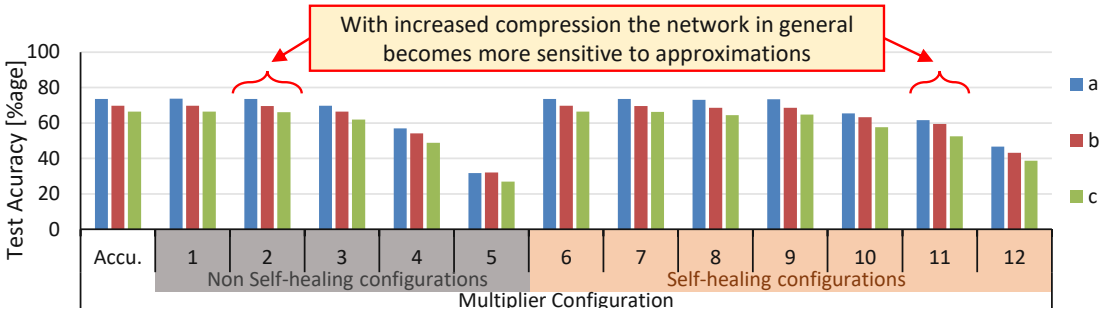


Figure 4.14: Effects of using approximate multipliers for inference of Cifar10 test images using different compressed LeNet5 variants marked in Fig. 4.8.

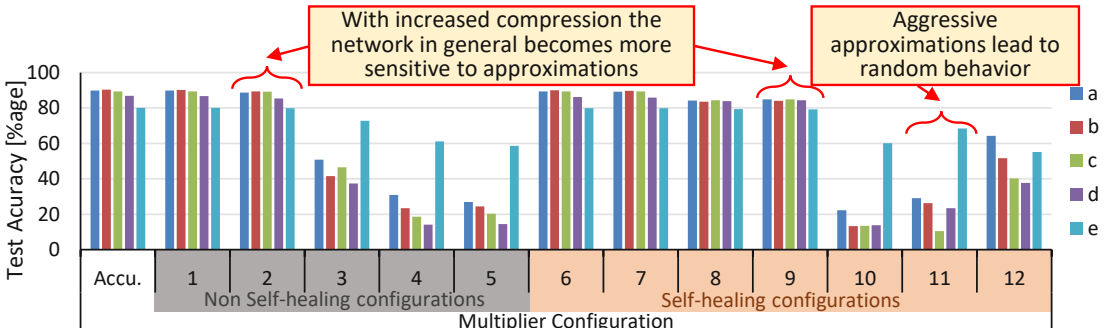


Figure 4.15: Effects of using approximate multipliers for inference of Cifar10 test images using different compressed VGG11 variants marked in Fig. 4.9.

4.2.4 Execution Time Analysis of the Proposed Cross-layer Approach

The overall execution time of the proposed cross-layer approach depends on various factors, e.g., the size of the network, the size of the input samples and the number of samples used for estimating the accuracy at intermediate steps. Table 4.4 presents the average execution time per iteration of the structured pruning algorithm. It clearly highlights that an increase in the number of samples for intermediate evaluation or the number of epochs for intermediate fine-tuning results in an increase in the overall execution time. Note that the table presents the average time per iteration, as the number of iterations depends on the cost function and the user-defined accuracy constraint (see from Figs. 4.3, 4.4, and 4.6). Table 4.5 presents the time required for evaluating the impact of quantization and approximation on the accuracy of the DNNs. Similar to the structured pruning case, the table presents the average time for evaluating the impact of quantizing a DNN to a specific bitwidth, as the overall time depends on the number of bitwidths considered in the search. However, for approximation, the overall execution time is presented, as, based on our implementation, the execution time can vary across configurations.

Table 4.4: Execution time of structured pruning algorithm for different cases

Network and dataset	No. of samples for intermediate evaluation	y	Average simulation time per iteration
LeNet with MNIST	512	0	1.67
		1	4.21
		2	7.44
		4	12.46
	10000	0	3.39
		1	6.38
		2	9.23
		4	14.68
LeNet with Cifar10	512	0	2.86
		1	9.95
		2	15.62
		4	28.29
	5000	0	5.49
		1	12.37
		2	18.86
		4	32.98
VGG11 with Cifar10	512	0	3.78
		1	15.19
		2	26.35
		4	49.24
	5000	0	10.5
		1	21.61
		2	33.06
		4	54.91

Table 4.5: Simulation time for evaluating the performance of quantization and approximation

Network and dataset	No. of samples for evaluation	Average simulation time per bitwidth for evaluating the impact of post-training quantization	Simulation time for evaluating the impact of accurate and 13 approximate configurations
LeNet with MNIST	10000	0.54	7.77
LeNet with Cifar10	5000	1.23	17.42
VGG11 with Cifar10	5000	4.72	69.68

4.3 Chapter Summary

State-of-the-art DNNs are highly overparameterized and resource-hungry. To enable the use of DNNs in resource constraint devices, several optimization techniques have been proposed at different abstraction layers of the computing stack. This chapter presented a novel cross-layer methodology to optimize DNNs. At the software level, it employs structured pruning along with quantization of inputs and network parameters to reduce the computational complexity and memory requirements of DNNs. At the hardware level, it deploys functional approximations in the arithmetic modules to further improve the efficiency by exploiting the error resilience characteristics of DNNs. The chapter also presented different case studies to investigate the effectiveness of each individual optimization technique employed in the cross-layer methodology in order to justify the proposed optimization flow. Note that although the case studies presented in the chapter considered only LeNet5 and VGG11 architectures, based on the results it can be concluded that the proposed technique can offer reasonable amount of compression and energy savings for other types of DNNs as well including ResNet and MobileNet architectures. However, based on the compactness of the given network and the complexity of the dataset/problem, the effectiveness of the proposed methodology can vary. The results in the chapter highlighted that software-level approximation/optimization techniques (such as pruning and quantization) offer far more potential for reducing the computational complexity and improving the efficiency of DNN inference compared to hardware-level functional approximation of modules. However, hardware-level approximations can further improve the efficiency when coupled with pruning and quantization. The results also highlighted that functional approximations of the arithmetic modules can lead to undesirable accuracy degradation. *This is mainly due to the fact that the impact on the DNN accuracy also depends on the data distribution of inputs, which highlights the need for data-driven approximations.*

CHAPTER 5

Neural Processing Arrays for Efficient Deep Neural Network Inference

This chapter presents novel techniques for improving the efficiency of Neural Processing Units (NPUs). The outline of the chapter is illustrated in Fig 5.1. First, Section 5.1 highlights different real-world settings for optimizing DNNs and discusses that under some scenarios, retraining is not possible to compensate for the accuracy lost due to approximations. Therefore, techniques that offer efficiency gains without affecting the accuracy under such scenarios are required to be explored. Afterward, Section 5.2 presents a study on the impact of approximations in neural arrays on the accuracy of DNNs. The study shows that moderate and high-intensity approximations lead to significant degradation in the application-level accuracy, and therefore, mitigation of approximation errors is essential. Towards this, Section 5.3 presents the concept of curable approximations and a neural array design based on the concept. The potential of data-driven approximations for DNNs is discussed in Section 5.4. The section presents a novel number representation format that enables simplification of MAC units in the

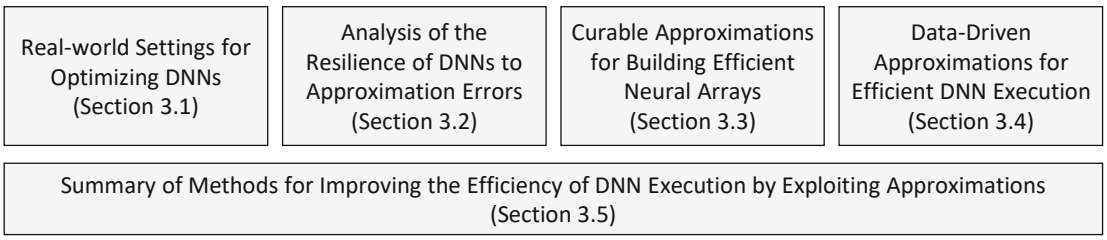


Figure 5.1: Chapter Overview.

neural arrays. At the end, Section 5.5 presents the summary of the chapter.

5.1 Real-World Settings for DNN Optimization

Training DNN models for complex applications is computationally intensive and moreover requires extensive data to achieve high accuracy. Most of the DNN optimization techniques are either embedded in the training process to train and compress the models simultaneously [HPN+16] or utilize retraining to counter the effects of approximations [ZGBG18]. However, extensive training data and computational resources are not always available. For example, consider a case where a vendor has released a DNN IP but has not made the training dataset available, for example, models trained on JFT-300M dataset [SSSG17], which is an internal Google dataset used for training image classification models. In such a case, as extensive training data is not available, it would not be possible for the user to retrain the model properly after applying approximations. Any significant modification to the network without proper retraining can affect the model's generalizability or some other characteristics.

Based on the availability of the training data and computational resources, there can be four different scenarios. The scenarios are listed in Fig. 5.2. As can be seen in the figure, retraining is only possible when both training data and sufficient computational resources are available. In all rest of the three cases, retraining is not possible. For the case where retraining is possible, aggressive pruning and quantization can be performed by exploiting retraining, and networks can be pruned by more than 90% [HMD15] and quantized to as low as 1-bit precision [RORF16][CHS+16]. However, for the cases where retraining is not possible, approximations can lead to significant accuracy loss. Therefore, it is essential to adopt approximations that have the least possible impact on the intermediate outputs while offering maximum efficiency gains.

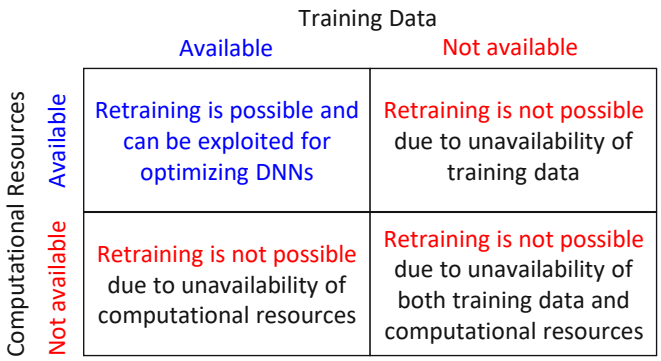


Figure 5.2: Different Settings for Optimizing DNNs.

5.2 Impact of Using Approximate Modules in Neural Processing Arrays

This section presents an analysis of the impact of using hardware-level approximations in neural arrays on the accuracy of DNNs. For this analysis, an 8-bit quantized version of the LeNet-5 network [VL15] trained on the CIFAR-10 dataset [KH+09] is considered. Moreover, it is assumed that approximations are deployed only in the multipliers. The baseline 8x8 multiplier design used in this analysis is based on Baugh-Wooley algorithm [BW73a] and is constructed using smaller 2x2 multipliers (similar to the design presented in Section 4.2.3). The approximate 8x8 multiplier variants are generated by replacing 2x2 accurate multipliers with 2x2 approximate multipliers. For this analysis, mainly, three different 8x8 approximate multipliers (Type 1, Type 2, and Type 3) are constructed by using the 2x2 approximate multiplier design proposed in [KGE11] for the least significant one, three, and four 2x2 multipliers, respectively. The error and hardware characteristics of the considered 8x8 multipliers are shown in Table 5.1. The approximation error of each multiplier is represented in terms of Mean Error Distance (MED) and is computed assuming uniform input distribution.

Table 5.1: Error and hardware characteristics of different multipliers used for implementing the LeNet network for classifying the cifar-10 images. The hardware results are generated for 65 nm technology node using Cadence Genus tool with TSMC 65 nm library.

	Latency [ps]	Area [cell area]	Power [μW]	MED
Accurate	1966.3	746	46.81	0
Approx_Mul_1	1915.9	710	45.64	0.125
Approx_Mul_2	1738.1	689	45.4	1.125
Approx_Mul_3	1728.5	682	44.87	3.125

Fig. 5.3 shows the impact of using the approximate multipliers on the classification accuracy of the network. Note that, for this analysis, we assume homogeneous array architecture, i.e., all the processing elements have exactly the same configuration and experience the same level of approximation. It can be observed from the figure that the classification accuracy of the network decreases with the increase in the approximation level. Moreover, the classification accuracy falls below the baseline even for the least approximate variant of the hardware, which can significantly degrade the performance and dependability of a safety-critical system. These results are in line with the results presented in Section 4.2.3, where it is highlighted that hardware-level approximations result in notable accuracy degradation, even in the case of pruned networks. *Therefore, there is a need to design approximate hardware such that the effects of approximations can be compensated, enabling significant performance and energy-efficiency gains while ensuring accurate/near-accurate results that have no impact on the application-level accuracy of safety-critical applications.*

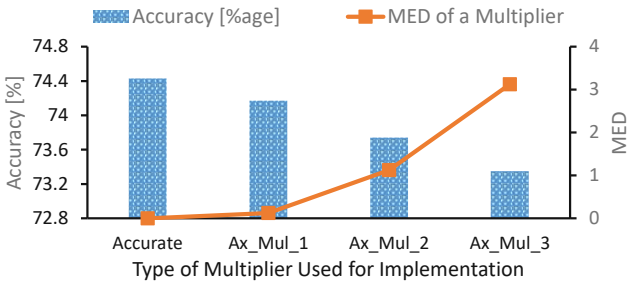


Figure 5.3: Effects of deploying approximations in multipliers on the accuracy of the LeNet-5 network trained on the Cifar-10 dataset.

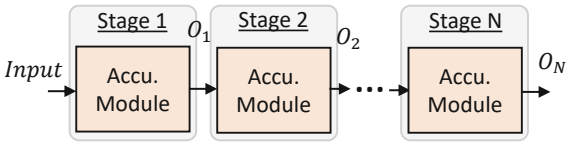
5.3 Curable Approximations for Building Efficient Neural Processing Arrays

This section presents a novel methodology for building approximate datapaths. The methodology utilizes modules with curative properties to compensate (or completely rectify) the errors that occurred in the previous stages/modules of a cascaded system. Each module accepts an error signal in a compressed form from its previous stage along with the inputs, compensates for the error, and generates an approximate output with a compressed error signal containing the information of the error in the current stage, which should be compensated in the subsequent stage.

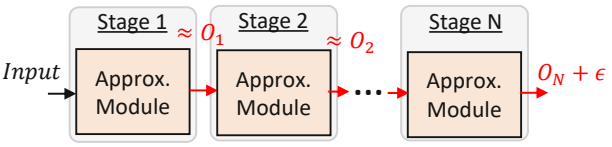
5.3.1 Methodology for Designing Datapaths with Curable Approximations

To understand the concept consider a reference system composed of N cascaded stages/ modules, as shown in Fig. 5.4a. An approximated version of the system is illustrated in Fig. 5.4b, where all the modules are approximated to achieve efficiency gains at the cost of some accuracy loss. As illustrated in the figure, each module generates output with some level of inaccuracy and thereby adds some amount of uncertainty in the overall output. The final output of the system is not accurate and can deviate significantly from the desired output based on the number of stages and the level of approximation deployed in each stage. Therefore, such design methods are unusable for many safety-critical applications and other high precision computing scenarios.

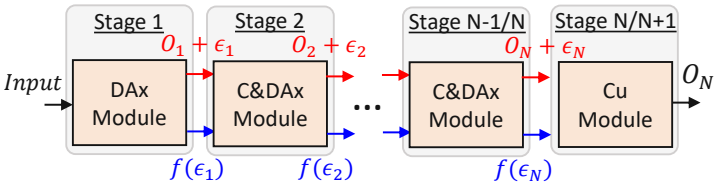
Fig. 5.4c illustrates the proposed variant of the reference system. The system is composed of three types of modules: 1) Deterministic Approximate (DAx) module (see Fig. 5.5a); 2) Cure & Deterministic Approximate (C&DAx) module (see Fig. 5.5b); and 3) Cure (Cu) module (see Fig. 5.5c). The DAx module generates approximate output along with a compressed yet deterministic error signal that can be used by the subsequent stage to decipher the exact amount of error occurred in the previous stage and compensate for it. The C&DAx module compensates for the error that occurred in the previous stage and generates an approximate output and a compressed error signal for the subsequent stage.



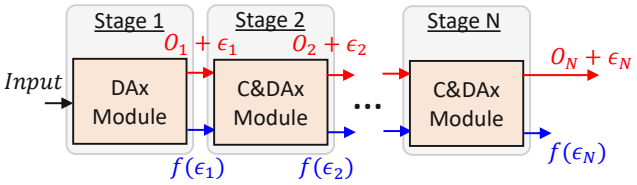
(a) An example system composed of N cascaded modules



(b) Equivalent approximate system composed of approximate modules



(c) Proposed equivalent accurate system composed of deterministic approximate and cure modules



(d) Proposed near-accurate system composed of deterministic approximate modules

Figure 5.4: Methods for building systems with cascaded modules. Here, $f(\epsilon_i)$ represents a reversible function of the error from the i^{th} stage, i.e., ϵ_i, which represents the error in a compressed form.

The last stage of the system is required to be a Cu stage to ensure completely accurate output. The stage is mainly responsible for compensating the error that occurred in the second to the last stage. Note that in some cases the Cu stage can be the N^{th} stage while in others, where it is not possible to design a cure stage while meeting the required functionality and efficiency gains, an additional stage, i.e., $N + 1^{th}$, is introduced to generate the accurate output. However, an alternative to this can be not using the cure stage, which introduces a small error equivalent to the approximation error in the last stage (see Fig. 5.4d). *Using the proposed methodology, unlike the system in Fig. 5.4b, the system in Fig. 5.4c (Fig. 5.4d) produces accurate (near-accurate) output while benefiting from the approximations in the modules.*

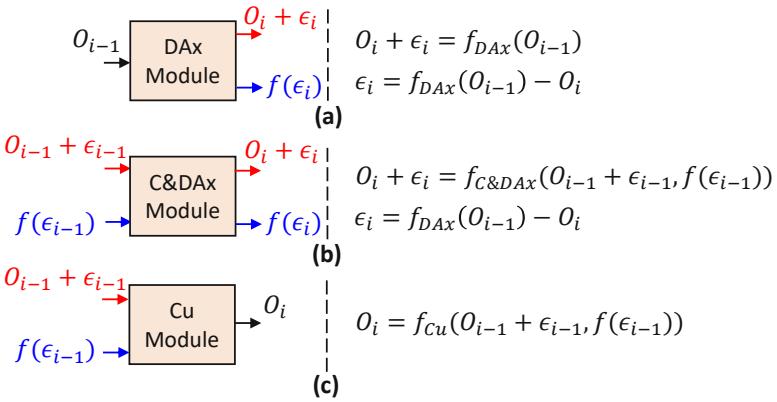


Figure 5.5: Functionality of different modules used in Fig. 5.4. O_i represents the accurate expected output and ϵ_i represents the approximation error generated by the i^{th} stage. The functions $f_{DAx}(.)$ and $f_{C\&DAx}(.)$ are approximate variants of the corresponding accurate module and $f_{Cu}(.)$ can also be a variant of the corresponding accurate module or just an additional correction module. $f(\epsilon_i)$ represents a reversible function of the error from the i^{th} stage.

5.3.2 Designing Efficient Neural Array using Curable Approximations

5.3.2.1 Designs of Required MAC units

To employ the proposed methodology for designing efficient neural processing arrays, first, the required MAC units are designed and then integrated in PEs for building an array. In this work, an array structurally similar to the array in Tensor Processing Unit (TPU) architecture [JYP+17] is designed. Moreover, 8-bit fixed-point precision is assumed for multiplications and the maximum possible partial sum size (within the array) is assumed to be 19-bit, which is sufficient for an 8x8 array. Note that, for DNN inference, 8-bit quantized DNNs offer almost the same accuracy as their full-precision counterparts, specifically for classification tasks [JKC+18][Inc]. An example computational array is shown in Fig. 5.8b and will be discussed in the next subsection.

Fig. 5.6 shows the Baugh-Wooley algorithm [BW73b] for multiplying two 8-bit signed operands. The sign extension can be performed by either extending the P_{15} bit directly or by adding $1s$ at the most significant location of the last partial product. This work employed the latter along with the Wallace tree architecture to design different required types of high-performance MAC units.

Fig. 5.7 shows the designs of accurate, DAx, and C&DAx MAC units by illustrating the accumulation steps for accumulating the partial-products and the partial sum. The partial-products are generated from the multiplication of a weight and an activation based on the Baugh-Wooley algorithm (as shown in Fig. 5.6) and the partial sum (19-bit number) is the output from another MAC unit. Note that here a merged MAC design is considered in which the partial-products (from the multiplication) and the partial sum are

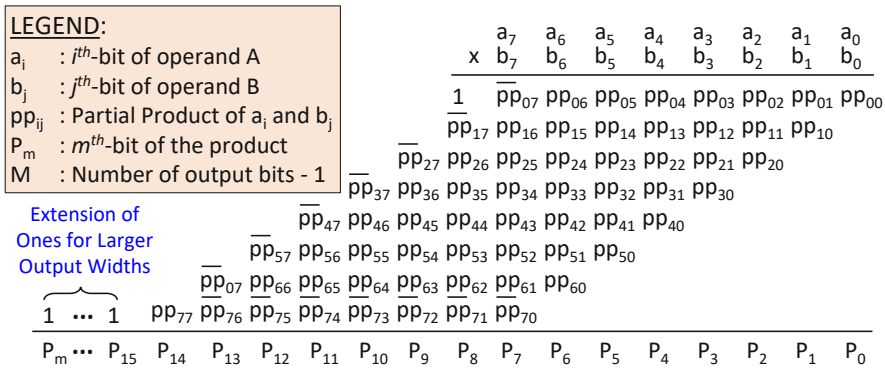


Figure 5.6: An 8x8 signed multiplication based on Baugh-Wooley algorithm.

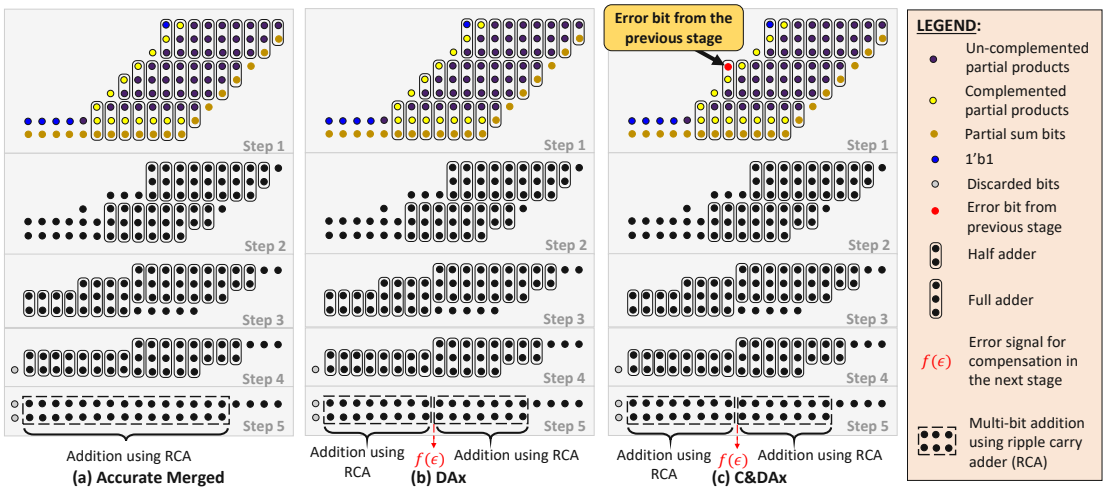


Figure 5.7: Different MAC unit designs based on Bough-Wooley algorithm and Wallace tree architecture. The multiplicand and the multiplier are assumed to be 8-bit wide, and the partial sums are assumed to be 19-bit wide. (a) Accurate Merged MAC. (b) Deterministic Approximate (DAx) MAC. (c) Cure amd Deterministic Approximate (C&DAx) MAC.

added simultaneously rather than performing the complete multiplication first and then adding the partial sum. Each MAC design has five accumulation steps where the first four steps are the compression steps followed by the final addition step. The compression steps use full and half adders as compressors to compress the partial products and the final step uses a multi-bit Ripple Carry Adder (RCA) for adding the final two arrays of bits. Fig. 5.7a shows the accumulation steps of the *Accurate Merged* MAC. For building DAx MAC (Fig. 5.7b), the final addition (in the 5^{th} step) of the accurate merged MAC is divided into two parts to improve the performance of the MAC unit, as the final addition is the most delay-incurring step. The compressed error signal $f(\epsilon)$, which is the carry out from the least-significant addition, is generated along with the approximate output

for the subsequent module. For building C&DAx MAC (Fig. 5.7c) from accurate merged MAC, the error bit ($f(\epsilon)$) is added in the 1^{st} compression step at the corresponding significance location by replacing a half adder with a full adder. Through this the error generated in the previous stage is compensated. However, to improve the performance, the final addition (in the 5^{th} step) is truncated, similar to DAx MAC shown in Fig. 5.7b. For this scenario, the cure (Cu) module is composed of an RCA equivalent to the length of the most significant RCA in the 5^{th} step of the DAx and the C&DAx MAC units, i.e., almost half the length of the RCA used in the accurate merged MAC.

5.3.2.2 Neural Array Design

Fig. 5.8b shows the neural array design similar to the one used in the TPU architecture [JYP+17]. The array is composed of multiple processing elements (PEs). The PE architecture is shown in fig. 5.8a. In the array, each PE is connected to its neighboring PE in a manner that it receives activations from its left neighboring PE (or input) and partial sums from the above neighboring PE. The weights are communicated to the respective PEs through vertical channels from top to bottom, and are stored inside the PEs during the computation.

Fig. 5.8d shows the modified neural array architecture designed based on the proposed methodology. The architecture of the PEs used in the array is shown in Fig. 5.8c. As the partial sums are communicated from top to bottom in the array, the first row comprises the approximate PEs containing DAx MAC (shown in Fig. 5.7b) at their core. Rest of all the PEs are designed to have C&DAx MAC (shown in Fig. 5.7c) at their core. An additional row is added at the bottom of the processing array to compensate for the error occurred in the last row of the array composed of approximate PEs containing C&DAx MAC units. The additional row is composed of Cu modules, which are adders of size equivalent to the size of the most-significant adder in Step 5 of Fig. 5.7c.

Note that the proposed curable approximations method is orthogonal to most of the circuit and architecture-level approximations. Therefore, it can be used in conjunction with other approximations for significantly improving the performance and power/energy efficiency of error-resilient systems, however, at the cost of some accuracy loss.

5.3.3 Results and Evaluation of the Proposed Neural Array Design

This subsection presents the performance, area and power results of the proposed neural array. To compare the methodology with the state-of-the-art, the conventional systolic array design shown in Fig. 5.8b is considered. Moreover, a systolic array composed of PEs containing accurate merged MAC units (from henceforth referred to as *Merged Accurate* systolic array) and an array composed of approximate PEs (from henceforth referred to as *Approximate* systolic array) are also considered. The approximate systolic array is constructed by using Type 3 approximate multiplier design (*Approx_Mul_3* from Section 5.2) in the conventional PEs, as shown in Fig. 5.9.

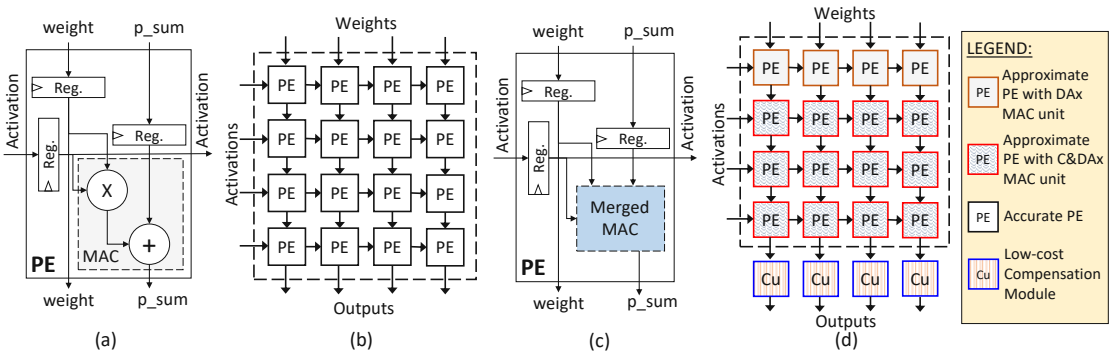


Figure 5.8: (a) Processing Element (PE) design with conventional MAC. (b) Conventional systolic array design similar to the systolic array of the TPU [JYP+17]. (c) Processing Element (PE) design with merged MAC. (d) Modified systolic array design based on the proposed methodology.

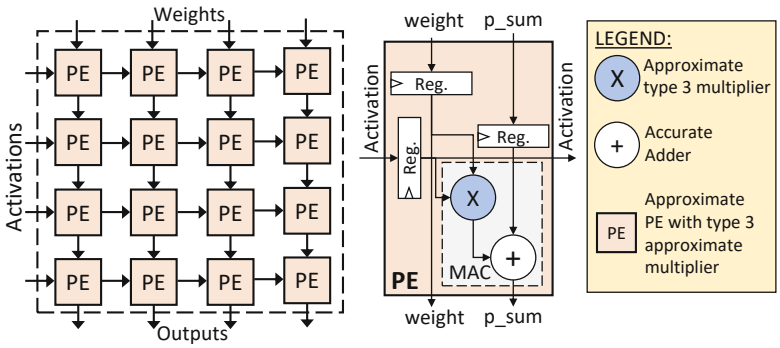


Figure 5.9: An approximate systolic array design based on Type 3 approximate multiplier from Section 5.2.

5.3.3.1 Accuracy Comparison

As highlighted earlier, the proposed methodology results in a system that offers accuracy equivalent to the reference system. To validate the results, a functional model of the proposed neural array was implemented in MATLAB and used to simulate the LeNet-5 network trained on the Cifar-10 dataset. The test accuracy of the network came out to be 74.43%, which is exactly the same reported with accurate simulations (see Fig. 5.3).

5.3.3.2 Performance, Area, and Power Evaluation of MACs

Table 5.2 presents the hardware results of the proposed MAC designs shown in Fig. 5.7, the conventional MAC shown in Fig. 5.8a and the approximate MAC shown in Fig. 5.9. It can be seen from the table that all the designs have almost the same power consumption, with the approximate and the conventional accurate MACs having slightly less consumption than others. The area numbers of the proposed MAC designs are close to each other; however, the conventional MAC and the approximate MAC consume approximately

19% and 10% more area compared to the proposed MACs respectively. The delay of the proposed DAx and C&DAx MAC units is the same and is slightly less than 50% of the delay offered by the conventional MAC and around 65% of the delay offered by the accurate merged MAC. The approximate MAC has higher delay than that of the accurate merged MAC due to the fact that its design is based on the conventional MAC design. Note that all the hardware results are generated for 65 nm technology node using Cadence Genus (Encounter) tool with TSMC 65 nm library.

Table 5.2: Hardware characteristics of different types of MAC units.

	Latency [ps]	Area [Cell Area]	Power [μW]
Accurate MAC (Merged)	1871.1	746	66.56
DAx MAC	1214.2	744	66.3
C&DAx MAC	1214.2	746	68.13
Accurate MAC (Conventional)	2470.9	889	62.73
Approx MAC with Approx_Mul_3	2274.2	822	61.14

5.3.3.3 Performance, Area, and Power Evaluation of Neural Arrays

Fig. 5.10 shows the overall hardware characteristics of four different neural array designs (i.e., *Conventional*, *Approximate*, *Merged Accurate* and *Proposed*) for two different systolic array sizes (i.e., 4x4 and 8x8). As can be seen in Fig. 5.10a, the *Proposed* design has less critical path delay compared to all other designs, which allows it to operate at 1.91x the frequency of the *Conventional*, 1.72x the frequency of the *Approximate*, and 1.47x the frequency of the *Merged Accurate* design. The area (in *Cell Area* unit) and power numbers are shown in Fig. 5.10c and b, respectively. It can be observed from the figures that the overall power and area of all the designs are approximately the same, with *Approximate* offering a bit better power consumption and *Accurate Merged* offering a bit better area. However, from Fig. 5.10d, it can be observed that the *Proposed* design offers approximately 46% reduction in PDP (Power Delay Product) compared to the *Conventional*, 38% compared to the *Approximate*, and 30% compared to the *Merged Accurate* design. From these results, it can be concluded that the proposed curable approximations method is highly effective for designing neural arrays.

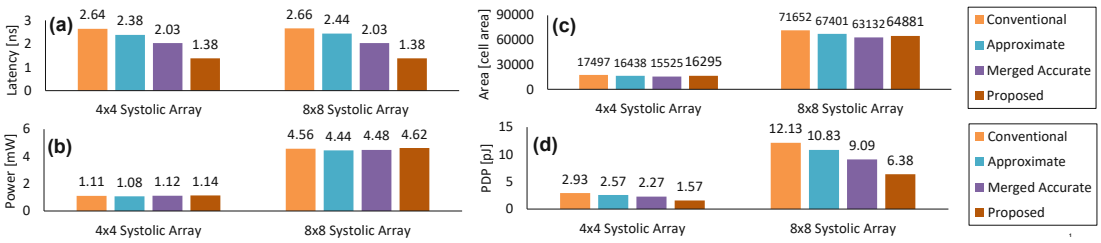


Figure 5.10: Comparison of the hardware characteristics of four different neural array designs for 4x4 and 8x8 sizes.

5.4 A Data-driven HW/SW Co-design Approach for Energy-Efficient DNN Inference

As highlighted in Sec. 5.1, retraining DNNs is not possible under some scenarios due to a lack of computational resources or training dataset. Therefore, the scope of the conventional retraining-based DNN optimization techniques, e.g., aggressive pruning [HMD15] and quantization [RORF16], is usually limited. Moreover, conventional approximations also offer sub-optimal quality-efficiency trade-offs (see Sec. 5.2). Therefore, to significantly improve the efficiency of DNNs without involving retraining, alternate techniques are required to be explored. Post-training quantization techniques are proposed to improve the DNN inference efficiency without involving retraining. These techniques achieve efficiency gains mainly by converting DNN data structures to low-precision Fixed-Point (FP) format. However, conversion to low-precision format introduces quantization errors, leading to potentially noticeable accuracy loss, specifically when used to reduce the precision below 8-bits. Techniques such as [JVS+18] have been proposed to reduce the quantization error through dynamic compensation and enable less than 8-bit precision. However, dynamic compensation requires additional hardware resources, which results in diminishing returns. To overcome the above-mentioned limitations, this section presents a novel data-driven algorithm and architecture co-design framework, CoNLoCNN, for efficiently approximating DNNs to improve their power/energy efficiency without affecting the application-level accuracy. Towards this, this section first highlights that a low-precision data representation format that is aligned to the long-tailed data distribution of DNN parameters (see Fig. 5.12) can result in low overall quantization error. Moreover, the application-specific data distribution can also be exploited to simplify hardware components, e.g., MAC units. Then, it presents a correlation-based low-cost error compensation strategy that is required to be applied only once at the conversion-time to partially compensate for the quantization errors. The section then finally presents a novel quantization scheme, a supporting neural array design and a systematic methodology for quantizing DNNs while exploiting error compensation strategies. A summary of the novel contributions is shown in Fig. 5.11.

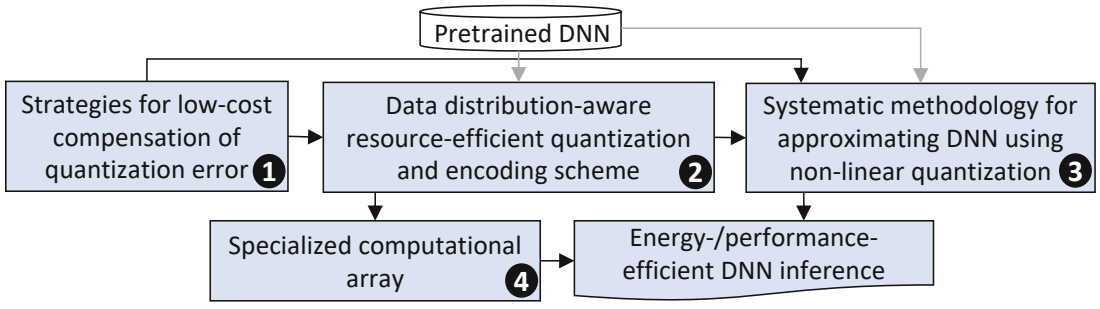


Figure 5.11: Novel contributions

5.4.1 Strategies for Enabling Low-Precision and Energy-Efficient DNN Inference

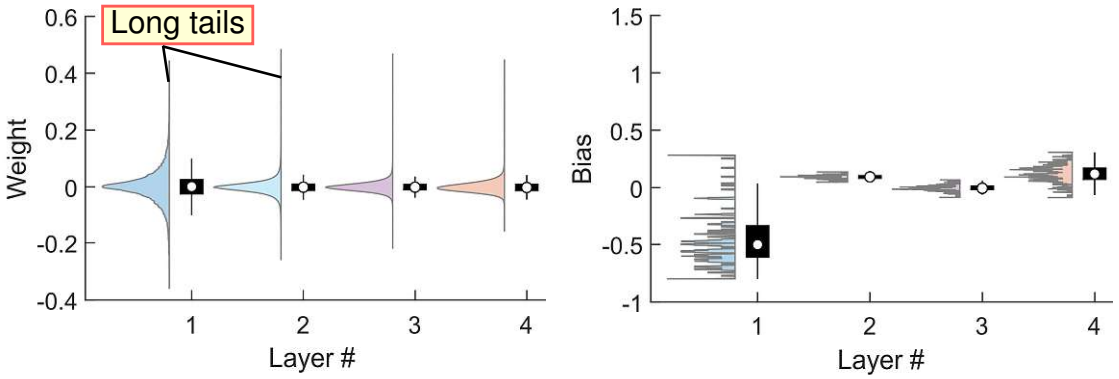
5.4.1.1 Strategy 1: Select quantization scheme based on data distribution of the corresponding DNN data structures

Fig. 5.12 shows the distributions of weights, biases and activations of the layers of a pre-trained AlexNet. It can be observed from the figure that the distributions of weights and activations have long tails, i.e., majority of the values are concentrated in small region and only a small set of values have large magnitude. Moreover, for each layer, the distribution of weights is close to a Gaussian distribution with mean at zero. Considering the long-tailed data distributions, a low-precision uniform quantization (for example, see Fig. 5.13(a)) would result in high average/overall quantization error compared to a non-uniform quantization (for example, see Fig. 5.13(b)) in which same (or less) number of quantization levels are distributed based on the probability density, i.e., more number of narrowly-spaced quantization-levels in dense regions and less number of widely-spaced quatization-levels in light regions. Therefore, aligning quantization scheme with the data distribution can reduce the average quantization error. However, a potential limitation of this is that low-precision non-uniform quantization can result in a significant increase in the maximum quantization error, which can increase the error variance. *Hence, the ideal quantization scheme should find a perfect balance between the average and the maximum quantization error to minimize accuracy loss.*

5.4.1.2 Strategy 2: Exploit correlation between neighboring feature map values to reduce the effective mean and variance of quantization errors

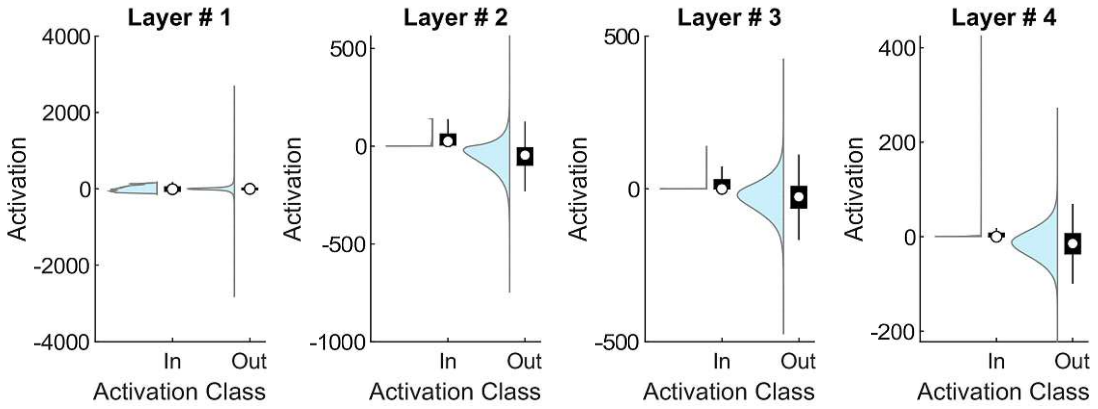
This section first presents an analysis of the impact of variations in bias values of a CNN on its classification accuracy. Then, it presents a study of the correlation of data within and across input feature maps of a layer. Afterwards, it presents a mathematical analysis to highlight how the gained insights can be exploited to reduce the impact of quantization error on intermediate outputs.

Impact of modifying the bias values in DNNs: Fig. 5.14 shows the impact of modifying the bias values of different number of filters of a layer of a pre-trained AlexNet on its classification accuracy. From Fig. 5.14(a) and Fig. 5.14(c) it can be observed that when a small value, i.e., a value close to the range of original bias values (see Fig. 5.12(b)), is added to the bias values of a number of filters, the accuracy of the DNN stays close to its baseline accuracy. However, when the magnitude of the value added is large, it significantly degrades the DNN accuracy. The difference between the impact of positive and negative noise (added value) is mainly due to the ReLU activation functions in the AlexNet, as a large negative bias leads to a large negative output which is then clipped to zero by the ReLU function and thereby limits the impact on the final output. Fig. 5.14(b) and Fig. 5.14(d) show that when the bias values of half of the selected filters are injected with positive noise and half with negative noise (all having the same magnitude), the



(a) Distribution of weights of the layers

(b) Distribution of biases of the layers



(c) Distribution of input and output activations of the layers

Figure 5.12: Distribution of weights, biases and activations of the first four convolutional and layers of the AlexNet (in the form of half-violin plots and box plots). Note that the output activations here represent the output of the layer before passing through activation functions.

behavior of classification accuracy is dominated by the behavior of filters that are injected with positive noise.

Given that large variations in bias values significantly affect the accuracy of DNNs, it can be said that approximations that lead to mean shifts in the output activations can degrade the application-level accuracy significantly. To highlight this further, Fig. 5.15 shows the results of an experiment where noise generated using a Gaussian distribution was added to the bias values of the filters/neurons of different layers of a pre-trained AlexNet. The results clearly show that when the noise is generated using smaller standard deviation values and is injected to intermediate layers of the network, it does not affect the accuracy much. However, when the noise is injected to the last layer of the network or is generated using large standard deviation values, it leads to a significant drop in the accuracy.

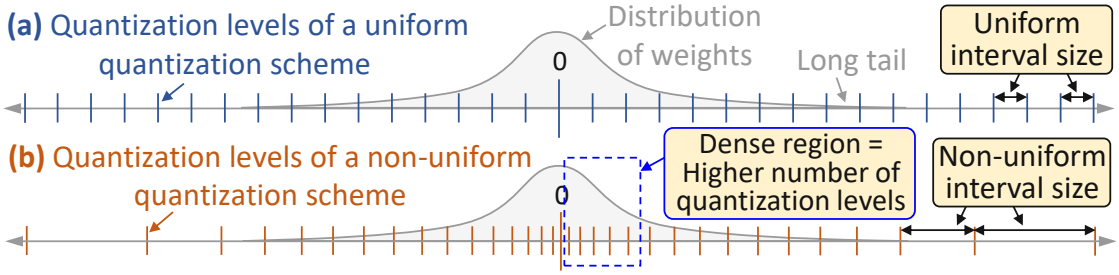


Figure 5.13: Comparison between uniform and non-uniform quantization.

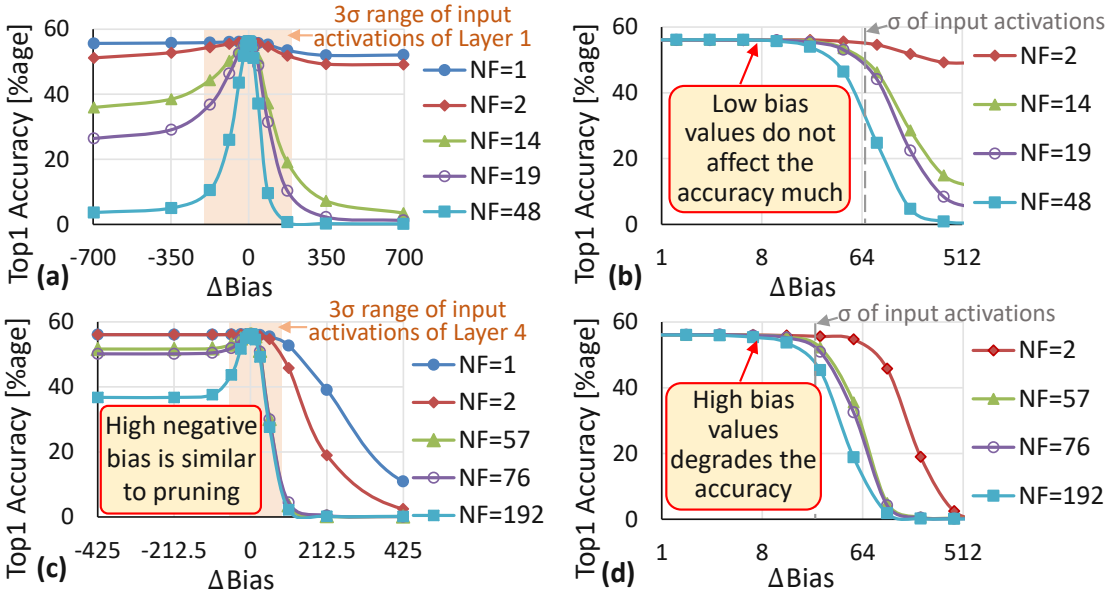


Figure 5.14: Impact of altering the bias values of different number of randomly selected filters/neurons (NF) of different layers of a trained AlexNet on its classification accuracy. (a) and (c) show the impact when same amount of positive (or negative) value is added to the bias values of the selected filters of layer 1 and 4, respectively. (b) shows the impact when the bias values of half of the selected filters of layer 1 are injected with positive noise and half with negative noise having the same magnitude. Similar to (b), (d) shows the results for layer 4.

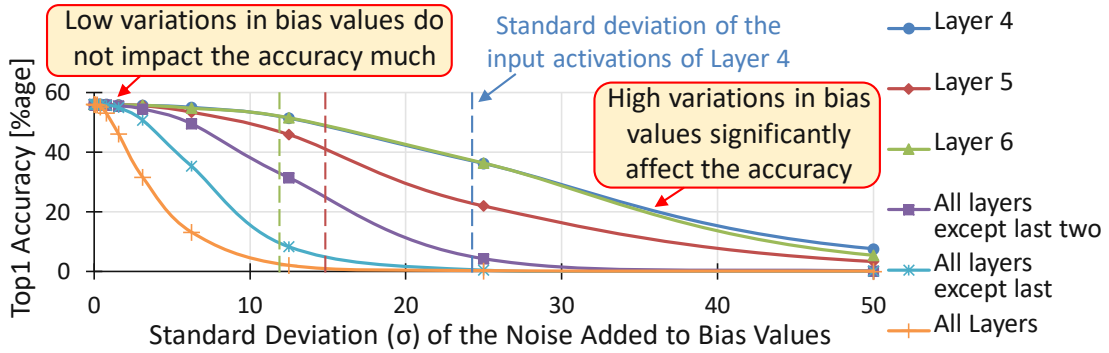


Figure 5.15: Impact of adding noise generated using a Gaussian distribution to the bias values of filters/neurons of different layers and different number of layers of a trained AlexNet on its classification accuracy.

From the above analysis, the following three key observations/conclusions can be extracted. (1) Small to moderate level of errors in the bias values of all the layers except the last layer do not impact the accuracy much. (2) Mean shift in the output degrades the accuracy only if it is large or is in the output of last layer of a DNN. (3) Resilience of DNNs to small-to-moderate errors in bias values highlight the significance of large activation values.

Intra-Feature Map Correlation: Fig. 5.16 shows the correlation between neighboring activation values located at a constant shift from each other (represented using i and j in the figure) in input feature maps of a convolutional layer. Based on the values reported in the figure, it can be said that, there is a significant correlation between the neighboring activation values in activation maps.

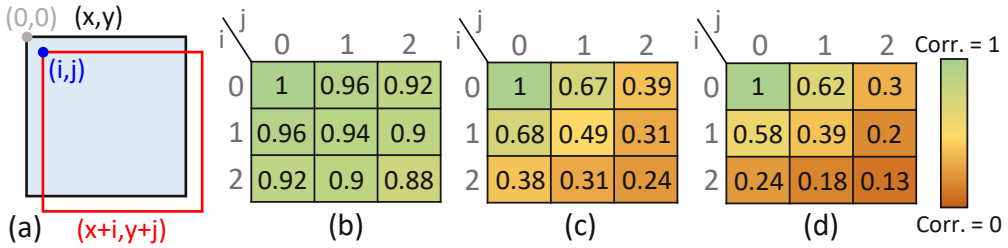


Figure 5.16: Intra-feature map correlation of input activations of different layers of the AlexNet and the VGG16. (a) Illustration of an input feature map (shown in blue) and its shifted variant (shown with red border). i and j define the shifts in x and y directions, respectively. (b) and (c) show the correlation between the input feature maps and their shifted variants of layer 1 and layer 3 of the AlexNet, respectively. (d) shows correlation between neighboring input activations of layer 12 of the VGG16.

Inter-Feature Map Correlation: Fig. 5.17 shows the distribution of correlation between different input feature maps of a layer of the AlexNet. Fig. 5.17(a) shows that

there is a significant correlation between the input feature maps of layer 1 of the DNN. However, the distributions in Figs. 5.17(b), 5.17(c) and 5.17(d) show that, for layer away from the input, the across-feature map correlation is almost centered around zero.

Based on the above analysis, it can be concluded that only intra-feature map correlation can be exploited for designing an effective error compensation strategy.

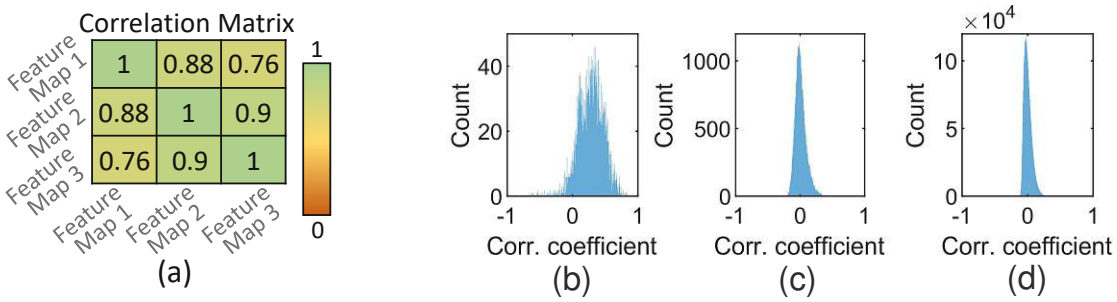


Figure 5.17: Correlation between input feature maps of different layers of the AlexNet. (a) Correlation matrix of input feature maps of layer 1. (b) Distribution of the correlation between input feature maps of layer 2. Similar to (b), (c) and (d) show distribtions of layer 4 and layer 7, respectively.

Analysis of quantization error: To analyze the effects of quantization on the output quality, consider a scenario in which the weights of a layer of a DNN are quantized while the activations are kept in full-precision format. A quantized weight can be mathematically represented as:

$$W_q = W + \Delta W \tag{5.1}$$

where W represents unquantized weight and ΔW represents quantization error. Now, if it is assumed that $W \sim \mathcal{N}(\mu_W, \sigma_W^2)$ and $\Delta W \sim \mathcal{N}(\mu_{\Delta W}, \sigma_{\Delta W}^2)$, and that W and ΔW are independent of each other, then

$$W_q \sim \mathcal{N}(\mu_W + \mu_{\Delta W}, \sigma_W^2 + \sigma_{\Delta W}^2) \tag{5.2}$$

Similar to the distribution of W, for activations, it is assumed that

$$A \sim \mathcal{N}(\mu_A, \sigma_A^2) \tag{5.3}$$

Using the above equations and assuming the weights and activations to be independent, $O_q = \sum_{i=1}^{n} W_{qi} * A_i$ can be transformed to

$$O_q \sim \mathcal{N}(n * \mu_A * (\mu_W + \mu_{\Delta W}), n * ((\sigma_W^2 + \sigma_{\Delta W}^2 + (\mu_W + \mu_{\Delta W})^2) * (\sigma_A^2 + \mu_A^2) - \mu_A^2 * (\mu_W + \mu_{\Delta W})^2)) \tag{5.4}$$

As highlighted in the earlier analysis, small deviations in the mean of output activations do not impact the accuracy much; therefore, the main focus should be on comparing the variance of O_q with the variance of unquantizaed output, i.e., $O = \sum_{i=1}^{n} W_i * A_i$. Subtraction of the variance of O from the variance of O_q leads to

$$n * (\sigma_{\Delta W}^2 * \sigma_A^2 + \sigma_{\Delta W}^2 * \mu_A^2 + \sigma_A^2 * \mu_{\Delta W}^2 + 2 * \sigma_A^2 * \mu_W * \mu_{\Delta W}) \quad (5.5)$$

Now, to reduce the intensity of this additional term, the intensities of $\sigma_{\Delta W}$ and $\mu_{\Delta W}$ are required to be reduced, which can be achieved by exploiting the high correlation among the neighboring activation values in feature maps. Fig. 5.18(a) shows a possible way of decomposing activation values of a block of feature map based on correlation among neighboring values. In case of high correlation (for example, see Fig. 5.18(b)), the variables x, y and z (shown in Fig. 5.18(a)) all have high values (i.e., close to 1). As $\bar{a_2}$, $\bar{a_3}$ and $\bar{a_4}$ represent the elements of a_2, a_3 and a_4 (respectively) that are orthogonal to a_1, the variables $\bar{a_2}$, $\bar{a_3}$ and $\bar{a_4}$ exhibit low variance compared to a_2, a_3 and a_4, specifically in cases where a_2, a_3 and a_4 have a strong correlation with a_1. This presence of xa_1, ya_1 and za_1 terms in the neighboring activations can be exploited to partially compensate/balance the error introduced in the dot-product of weights and activations due to quantization of weights, and it can be achieved by modifying the quantization scheme in such a way that it balances the mean quantization error of the neighboring weights.

Example: To understand this, consider the activation block A shown in Fig. 5.18(c) and 2D filter W shown in Fig. 5.18(d). The output of dot-product of A and W comes out to be 50.32. Now, quantize the weights of the filter are quantized to the nearest integer values and then the dot-product operation is performed, the result comes out to be 57.7 (see Fig. 5.18(e)). However, if the second weight is mapped to its other nearest integer value, i.e., 2 instead of 3 (see Figs. 5.18(e) and 5.18(f)), the mean absolute error (MAE) in weights can be reduced from 0.225 to 0.025 and the absolute error in the output of dot-product from 7.38 to 1.12. This shows that high correlation among the neighboring values can be exploited to reduce the effective mean and variance of ΔW by minimizing the local mean quantization error inside filter channels.

Impact of Adjusting Intra-channel Mean Quantization Error in Weights:

Fig. 5.19 highlights the impact of adjusting the mean error in filters on the output accuracy of a DNN trained for image classification application. It mainly covers three different cases: (1) *No adjustment* in the mean error of the weights; (2) *Mean error adjustment* where the mean error of each filter is subtracted from the corresponding weights; and (3) *Mean error adjustment* where the mean error of each filter channel is subtracted from the corresponding weights. As can be seen in the figure, among the three cases, intra-channel error adjustment leads to highest accuracy/compensation.

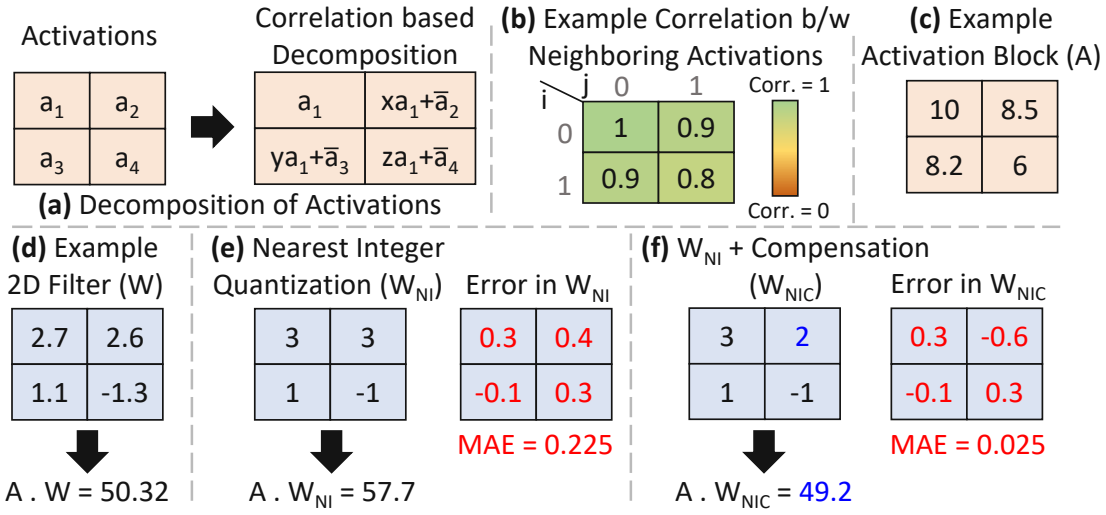


Figure 5.18: Decomposition of activations, and an example illustrating the impact of exploiting correlation for error compensation on the output of dot-product operation.

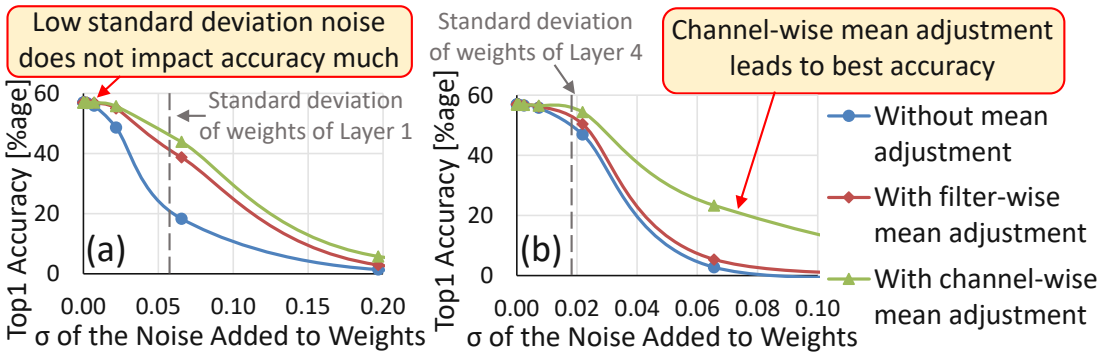


Figure 5.19: Impact of adjusting the inter- and intra-channel mean quantization error in the weights of AlexNet. (a) Layer 1; (b) Layer 4.

5.4.2 Proposed Type of Non-Uniform Quantization and the Design of Supporting DNN Hardware

Strategy 1 in Section 5.4.1 highlights that aligning quantization scheme with the data distribution can help in reducing the adverse effects of quantization on DNN accuracy. However, the key challenge is *how to exploit this observation for improving the efficiency of DNN systems.* To address this, in this work, an Encoded Low-Precision Binary Signed Digit (ELP_BSD) representation format is proposed. The format is mainly inspired by the power-of-two representation, which enables the use of resource-friendly shift operations instead of multiplications. The following subsections present details of the proposed concept and data representation format.

5.4.2.1 Initial Proposition

The main focus of this work is on exploiting non-uniform quantization to simplify the MAC unit design and to reduce the bit-width of weights (wherever possible), as both can significantly contribute towards reducing the overall inference cost. Power-of-two quantization is one potential candidate, as it enables simplification of a costly multiplication operation to a shift operation; moreover, it also offers the opportunity to reduce the bit-width of weights by storing only the exponents. The power-of-two quantization levels are aligned to the probability density function of weights, as can be observed from Fig. 5.20(b). However, use of power-of-two quantization results in a significant drop in application-level accuracy due to the reduction in the number of quantization levels compared to a traditional FP quantization scheme; this can be observed by comparing Fig. 5.20(b) with Fig. 5.20(a). *Therefore, almost all the previous works that use power-of-two quantization employ quantization-aware retraining to tune the given DNN for the underlying quantization scheme and enable efficiency gains with minimal accuracy loss.* To avoid retraining, this work proposes the use of sum of signed power-of-two quantization, where *more than one signed power-of-two digits* are combined to offer additional quantization levels than simple power-of-two quantization, which help in significantly reducing the overall quantization error and its impact on the DNN accuracy while still benefiting from the key advantages of power-of-two quantization scheme. Fig. 5.20(c) shows that addition of even a single low-range low-precision signed power-of-two term can significantly increase the number of unique quantization levels.

5.4.2.2 Limitations of Initial Proposition and Proposed Improvements

One of the key issues with sum of signed power-of-two quantization is redundant representations of values, which can result in diminishing returns. For example, $2^{x_1} - 2^{x_2} = 0 \; \forall \; x_1 = x_2$. To reduce such redundancy, this work proposes to reduce the number of possible power-of-two values per term. For example, if a number representation is given as $2^{x_1} - 2^{x_2}$ and $x_1, x_2 \in \{0, 1, 2, 3\}$, to reduce redundancy, we can reduce the set of possible values of x_1 to $\{1, 3\}$. Fig. 5.20(d) shows that reducing the number of possible power-of-two values for the first digit leads to a reduction in the amount of redundancy; this can be observed from the reduced number of yellow semi-circles (representing the range of the second term) at the center of the overall range. Note that this reduction in redundancy can be exploited to further simplify the MAC unit design as well as to reduce the bit-width of DNN weights. However, a drawback of this is that such optimizations can result in a small decrease in the number of quantization levels, as highlighted in Fig. 5.20(d). The redundancy can also be reduced by restricting some of the terms in sum of power-of-two format to only positive (i.e., unsigned) values; for example, see Fig. 5.20(e) where the second digit is restricted to only positive values.

5.4.2.3 ELP_BSD data representation

To efficiently represent low-precision sum of signed power-of-two numbers, this work defines a novel data representation format, Encoded Low-Precision Binary Signed Digit

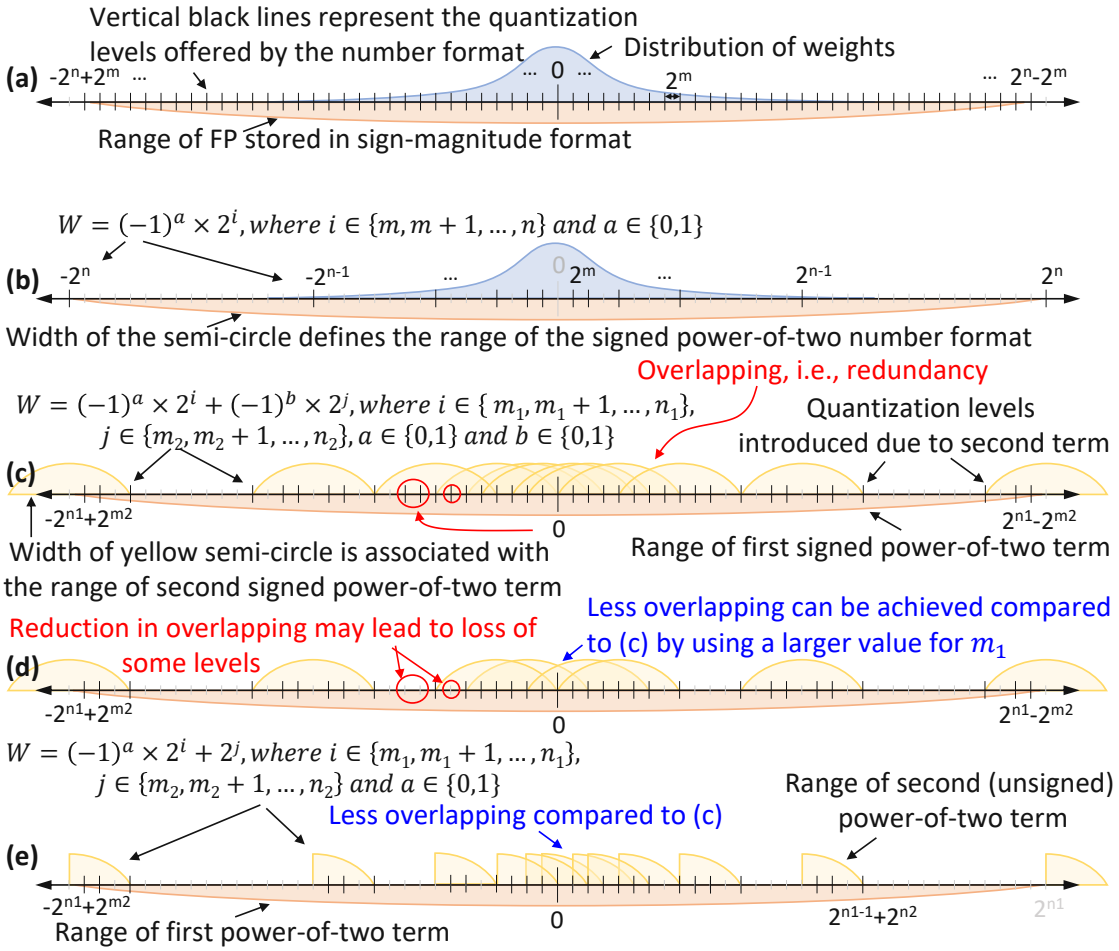


Figure 5.20: Illustration of different data representation formats that show step-by-step evolution of traditional quantization scheme to our ELP_BSD representation.

(ELP_BSD) representation. Fig. 5.21(a) shows how the specifications of an ELP_BSD representation are defined, and Fig. 5.21(b) shows the corresponding binary representation format. As shown in Fig. 5.21(b), the bits are divided into m groups, where each group is responsible for representing a single signed power-of-two digit/term. Each group consists of a sign bit and $ceil(log_2(d_i))$ bits to represent the index of $shift\ count$ mentioned in the digit specification. Here d_i is the total number of different $shift\ counts$ mentioned for i^{th} digit in the specification. Note that the sign bit is optional and only used when the corresponding digit is signed. Fig. 5.21 also presents two examples to explain the conversion of ELP_BSD numbers to values.

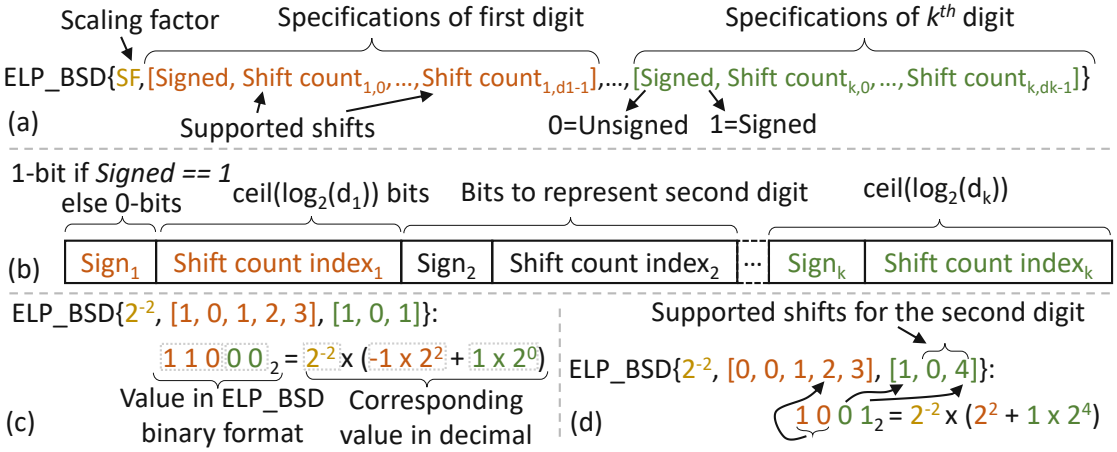


Figure 5.21: (a) Specifications of an ELP_BSD format. (b) ELP_BSD format. (c) and (d) Examples to explain conversion between ELP_BSD format and values.

5.4.2.4 Complementary Hardware Design

As shown in Fig. 5.21, ELP_BSD format mainly stores power-of-two digits in an encoded format. To efficiently implement multiplication with a power-of-two digit at the hardware level, shifters (e.g., barrel shifters) can be used. To realize a MAC unit for the processing array, the shifter is followed by an adder that adds previously computed partial sum to the newly computed product. Fig. 5.22(a) shows the MAC unit design for the case where weights are represented using a single signed power-of-two digit (see Fig. 5.23 for a detailed view of the same). If the same ELP_BSD format is used for all the weights, the shifter can be designed such that index term/s from the encoded weights are used directly to perform the corresponding amount of shifts/multiplication. Note that the set of possible *shift counts* can be optimized to get a less complex shifter design.

For the case where weights are represented using an ELP_BSD format that contains multiple digits, multiple of these units can be employed in parallel to perform the required multiplications (for each individual digit) and generate partial sums, which then have to be added together to generate the output. To achieve this addition, a compressor tree is first employed to efficiently compress the operands, and then, the final output is achieved using a multi-bit adder. Fig. 5.22(b) shows how multiple single digit MAC units can be integrated in a Processing Element (PE) of a Neural Processing Array (NPU), e.g., a Tensor Processing Unit (TPU) [JYP+17] like array. Fig. 5.22(c) shows the processing array design of a TPU like architecture. The processing array follows a weight stationary dataflow, i.e., where weights are kept stationary inside the PEs during execution. The input activations are fed from the left and moved towards the right cycle-by-cycle. Similarly, the partial sums flow from top to bottom. Note that this work assumes that activations are represented in FP 2's complement format, as changing the format of activations won't have any significant impact on the length of the adders in PEs but using 2's complement format simplifies the inversion block. Also note that in

most of the case one to three single digit MAC units per PE are sufficient to meet the user-defined accuracy constraints.

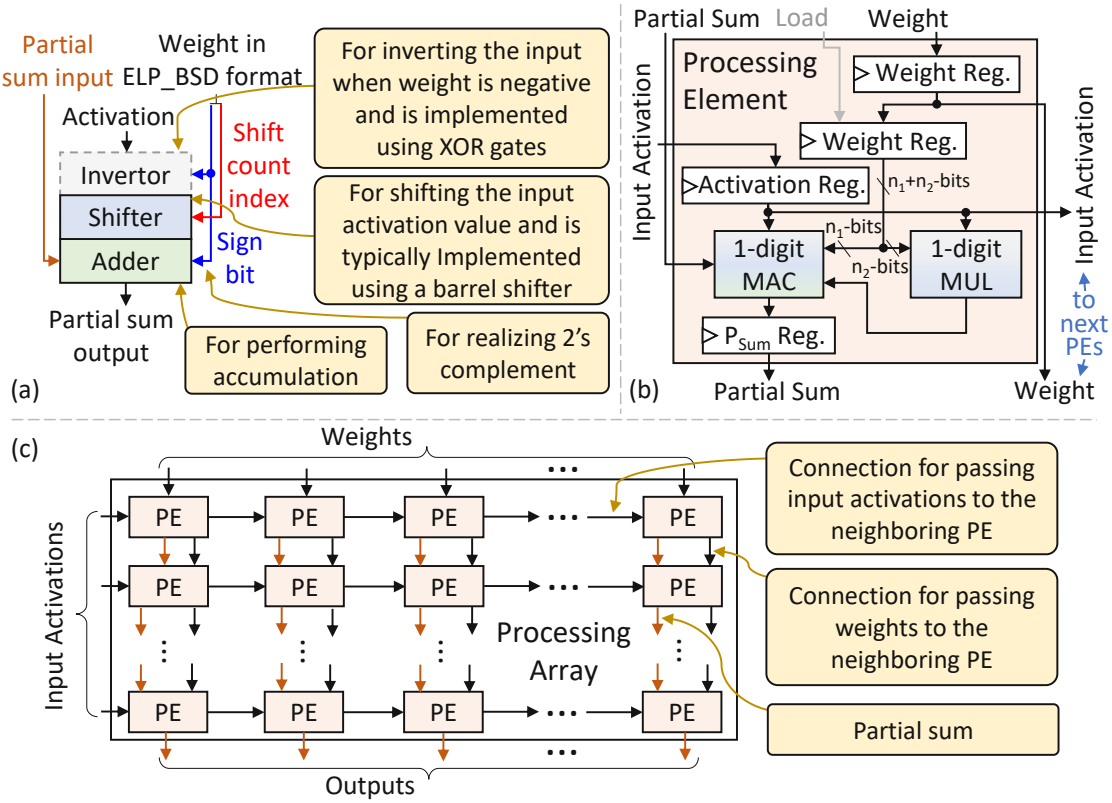


Figure 5.22: (a) Single digit MAC design. (b) Modified processing element for an NPU. (c) A Neural Processing Array architecture.

5.4.3 Novel Methodology for Efficient Approximation of CNNs through Non-Uniform Quantization

Fig. 5.24 presents a novel methodology for approximating CNNs through non-uniform quantization while exploiting the strategies mentioned in Section 5.4.1. The following steps explain the working of the overall methodology.

1. **Determine critical FP bit-width of activations:** Starting from maximum allowed FP bit-width (defined by the user) for activations, gradually reduce the bit-width of activations to find the critical point after which the accuracy of the given DNN falls below the user-defined Accuracy Constraint (AC). For this work, the bit-width of all the activations is assumed to be the same. The key intuition behind this is that a decrease in the activations' bit-width results in a linear decrease in the width of MAC units, which can significantly contribute towards improving the energy-efficiency of the system. This step outputs critical bit-width for activations, denoted as CBW_A.

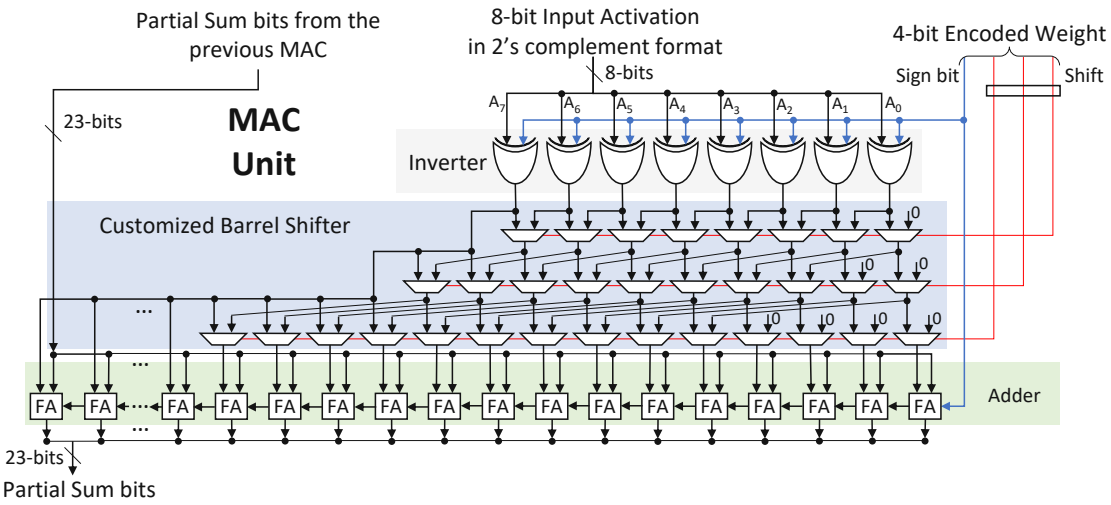


Figure 5.23: Detailed view of the single digit MAC unit shown in Fig. 5.22(a), assuming activation bit-width to be 8-bits, encoded weight bit-width to be 4-bits and the array size to be 256x256

2. **Determine scaling factor for weights:** Given the data representation format for weights, the scaling factor is computed for the weights of each layer of the given DNN individually. In this work, for ELP_BSD format, we compute the scaling factor using: $SF = max(weights)/2^{max(shift\_counts)}$.

3. **Apply nearest neighbor quantization:** Using the data representation format and the scaling factors, generate a table of possible quantization levels (TQL) for each layer of the given DNN. Then, to quantize the weights, replace the weights with their corresponding nearest values in the table.

4. **Apply error compensation algorithm:** For each convolutional layer, pass the weights (complete set of filters) to Algo. 5.1, which (partially) compensates for the quatization errors by exploiting Strategy 2 mentioned in Section 5.4.1. Note, as most of state-of-the-art CNNs are based on 3x3 or 5x5 filters, Algo. 5.1 focuses on reducing the mean quantization error within filter channels. Given a filter channel, it mainly computes the mean quantization error of the channel, finds the weights that can be mapped to their other neighboring quantization level to reduce the error mean, sorts all the located weights based on a cost function and starts altering the values starting from the weights having the least cost and keeps altering till the absolute mean error is decreasing.

5. **Estimate the overall accuracy loss:** This step computes the DNN accuracy to check if the user-defined accuracy constraint is met. If the constraint is not satisfied, the algorithm increases CBW_A by 1 and performs accuracy evaluation again. If the constraint is still not met and CBW_A becomes equal to BW_{max}, it outputs the latest quantized DNN.

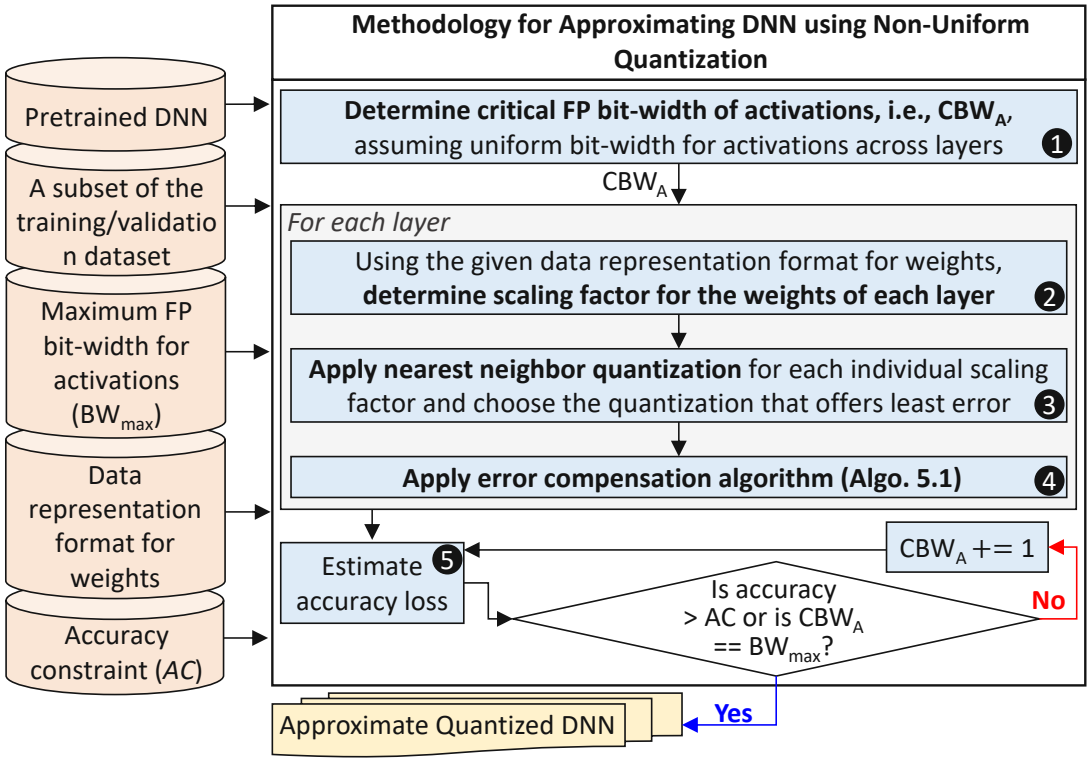


Figure 5.24: Proposed DNN quantization methodology

5.4.4 Results

5.4.4.1 Experimental Setup

To highlight the effectiveness of CoNLoCNN, in this work, two popular DNNs, i.e., AlexNet and VGG16, are considered, as these networks are commonly used for benchmarking FP DNN implementations. Both the networks are trained on ImageNet dataset. For accuracy evaluation, the CoNLoCNN methodology is implemented using MatConvNet [VL15] framework. For hardware results, the PE designs composed of different MAC units were implemented in VHDL and synthesized for the TSMC 65nm technology using Cadence Genus.

5.4.4.2 Effectiveness of the Proposed Error Compensation Strategy

To demonstrate the effectiveness of error compensation algorithm, i.e., Algorithm 5.1, it is first tested over low-precision FP implementation of DNNs. Note, for this experiment, it is assumed that the bit-width of weights and activations are the same and uniform across all the layers of the given DNN. Fig. 5.25(a) shows the results for AlexNet with ImageNet dataset. As can be observed from the figure, the proposed error compensation strategy helps in reducing the adverse effects of quantization, specifically at lower bit-widths. For

Algorithm 5.1: Pseudo-code for low-cost error compensation

Input: Un-quantized weights of a CONV layer ($UnQunat\_W$); Table of possible quantization levels (TQL)
Result: Quantized weights ($Quant\_W$)
$Quant\_W \leftarrow$ NNQuat($UnQunat\_W$, TQL); % Nearest neighbor quantization
$Error\_W \leftarrow UnQunat\_W - Quant\_W$; % Error in weights
for ($i = 1$; $i \leq$ *No of filters* ; $i+ = 1$) **do**
  for ($j = 1$; $i \leq$ *No of channels* ; $j+ = 1$) **do**
    $Mean\_Err$ = mean($Error\_W(:, :, i, j)$); % Mean error of the channel
    $S \leftarrow$ Subset of $UnQunat\_W(:, :, i, j)$ having corresponding error opposite in sign to $Mean\_Err$
    $SO \leftarrow$ Values of S quantized to closet levels in the opposite direction to the nearest neighbor
    $Cost_S = abs(S - SO)$
    $Sorted\_S \leftarrow$ Set of sorted values of S in order of increasing cost
    for ($k = 1$; $i \leq$ *No of values in Sorted_S* ; $k+ = 1$) **do**
      $New_Mean_Err \leftarrow$ Mean quantization error if the quantized value of $Sorted\_S(k)$ in $Quant\_W$ is replaced with the corresponding value from SO
      if *abs(Mean_Err) > abs(New_Mean_Err)* **then**
        Accept the change in quantization level of value corresponding to $Sorted\_S(k)$ in $Quant\_W$
        $Mean\_Err = New\_Mean\_Err$
      else
        break
      end
    end
  end
end

example, consider the case where the weights and activations are quantized to 5-bits, the implementation without proposed error compensation offers lower accuracy compared to the implementation that is passed through the the error compensation algorithm.

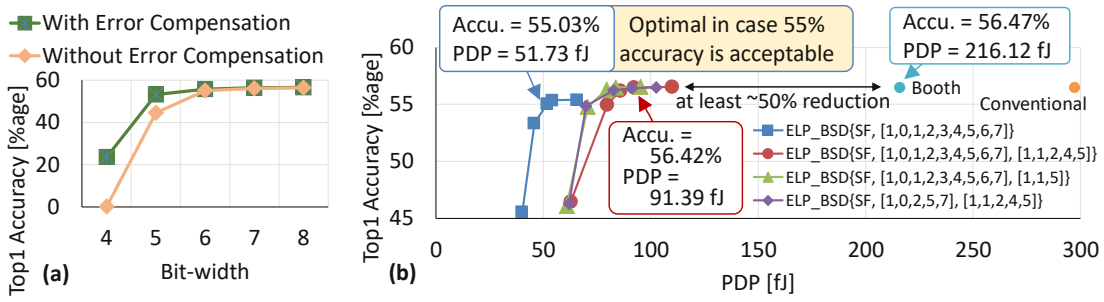


Figure 5.25: (a) Effectiveness of our error compensation strategy when used with traditional FP quatization for the AlexNet. (b) Accuracy of the AlexNet vs. PDP for different ELP_BSD data representations.

5.4.4.3 Effectiveness of CoNLoCNN for State-of-the-Art DNNs

To demonstrate the effectiveness of the proposed methodology, one single digit and three two digit ELP_BSD data representation formats are considered in this work. The complete details of the formats are listed in Table 5.3 along with the hardware

characteristics of the PEs implemented using the corresponding MAC designs for a 32x32 processing array (array architecture is shown in Fig. 5.22(c)). Note, the scaling factor (represented by x in the table) of the representations is not considered to be the same across layers, and it is selected for each layer individually based on its parameters. For each data representation format, five different activation bit-widths, i.e., 8-bit till 4-bit, are considered to study the impact of change in activation bit-width on the accuracy of the given DNN and the inference cost (in terms of hardware efficiency). Fig. 5.25(b) shows the accuracy vs. PDP results achieved with CoNLoCNN for AlexNet considering the ELP_BSD configurations mentioned in Table 5.3. Note, for each ELP_BSD configuration, five points are plotted in the figure, which correspond to five different activation bit-widths, i.e., 8-bit till 4-bit. The right most point belonging to the same configuration corresponds to 8-bit activations while the left most corresponds to 4-bit activations. Fig. 5.25(b) also includes the results achieved with the conventional and the booth multiplier-based designs for comparison purposes. The figure clearly shows that all the considered ELP_BSD formats result in significant reduction in the PDP compared to the conventional and the booth multiplier-based designs. Moreover, the figure also shows that a decrease in the bit-width of the activations from 8-bit to 6-bit offers further reduction in PDP without significantly affecting the accuracy of AlexNet. However, further decrease in the bit-width results in higher accuracy degradation. Further, it can be observed from the figure that different ELP_BSD formats offer different accuracy-efficiency trade-offs. Quantitatively, Fig. 5.25(b) shows that even the most power consuming considered CoNLoCNN design offers (at least) around 50% reduction in PDP compared to the conventional and booth multiplier-based designs, and if the accuracy constraint is 55%, around 75% reduction in PDP can be achieved through CoNLoCNN.

5.4.4.4 Comparison with State of the Art

For comparison with state of the art, CoNLoCNN is compared with CAxCNN [RHK+20]. First, CAxCNN [RHK+20] is implemented using the developed framework. Then, Canonical Approximate (CA) representation with one non-zero digit is selected to generate results. The weights of AlexNet are converted to CA format using their most expensive exhaustive search algorithm to have a fair accuracy comparison. For AlexNet, CAxCNN with one non-zero digit CA representation achieves 50.9 % top1 accuracy while CoNLoCNN with ELP_BSD{SF,[1,0,1,2,3,4,5,6,7]} (i.e., the simplest format closest to one non-zero digit CA representation) can offer 55.4% accuracy. This improvement is mainly due to the proposed error compensation strategy, as the quantization levels offered by one non-zero digit CA are almost the same as offered by ELP_BSD{SF,[1,0,1,2,3,4,5,6,7]} format with the only difference being of '0', which is not present in ELP_BSD{SF,[1,0,1,2,3,4,5,6,7]}. However, absence of '0' does not affect the accuracy as ELP_BSD{SF,[1,0,1,2,3,4,5,6,7]} has '1' and '-1' quantization levels and error compensation. Moreover, even in the best possible scenario, CA representation can reduce the weight bit-width to only 5 bits per weight while ELP_BSD{SF,[1,0,1,2,3,4,5,6,7]} reduces the weight bit-width to 4 bits per weight. Similar to AlexNet, for VGG-16, it is observed that CoNLoCNN offers 3% higher accuracy compared to CAxCNN.

Table 5.3: Hardware characteristics of the PEs designed using our methodology for some of ELP_BSD representations and their comparison with booth multiplier-based and conventional multiplier-based PEs.

	Bit-width		With 8-bit Activations			
Configuration of MAC for PEs	Weights	Activations	Area [cell area]	Power [uW]	Delay [ns]	PDP [fJ]
ELP_BSD{x, [1,0,1,2,3,4,5,6,7]}	4	8	556	28.55	2.30	65.68
		5	450	23.06	1.99	45.79
ELP_BSD{x, [1,0,1,2,3,4,5,6,7], [1,1,2,4,5]}	7	8	838	59.60	1.85	109.96
		5	694	46.53	1.71	79.71
ELP_BSD{x, [1,0,1,2,3,4,5,6,7], [1,1,5]}	6	8	814	51.65	1.85	95.29
		5	676	41.22	1.71	70.65
ELP_BSD{x, [1,0,2,5,7], [1,1,2,4,5]}	6	8	835	56.57	1.81	102.61
		5	680	43.07	1.62	69.86
Booth Multiplier-based MAC	8	8	1195	86.73	2.49	216.12
Conventional FP	8	8	1179	83.56	3.56	297.47

5.5 Summary of Efficient Neural Array Design

Retraining DNNs is not feasible in some cases due to the lack of computational resources and/or extensive training data. Therefore, alternate techniques are required that can offer high-efficiency gains without affecting the accuracy of DNNs. Various approximation techniques have been proposed in the literature to reduce the DNN inference cost. However, they offer only limited gains when used with strict accuracy constraints. Towards this, this chapter presented a curable approximation technique, which enables high-efficiency gains at no accuracy loss by employing approximate modules having error curing characteristics. Based on the concept, the chapter presented a novel neural array design that offers over 30% reduction in PDP while offering the same DNN accuracy as an accurate (non-approximate) array design. The chapter also highlighted the significance of application and distribution-aware approximations. By exploiting the concept and the advantages of power-of-two quantization, a novel non-linear quantization approach is presented, which significantly reduces the complexity of MAC units used in the neural arrays. Alongside the non-linear quantization method, a novel data representation format, i.e., Efficient Low-Precision Binary Signed Digit (ELP_BSD) format, is also presented. Using the proposed data representation format, DNN weights can be represented using lower bit widths to reduce the overall memory requirements during inference. The chapter also presented an error compensation strategy based on the correlation between activation values, to reduce the impact of errors induced due to the non-linear quantization of DNN parameters. The proposed CoNLoCNN methodology, which combines the novel non-linear quantization, ELP_BSD data representation format and the compensation strategy, offers about 75% reduction in PDP of the neural array at only 1.44% accuracy

loss for AlexNet trained on the ImageNet dataset. Note that partial retraining can be combined with the proposed techniques to overcome the accuracy loss, whenever possible.

CHAPTER 6

A Low-cost Technique for Mitigating the Effects of Permanent Faults in Deep Neural Network Accelerators

As the semiconductor industry is moving towards smaller technology nodes, technology-induced reliability threats are drastically increasing. These threats include *permanent faults* that impact the yield of the manufacturing process, *soft errors* that affect the functionality of the system at run-time, and *aging* that results is gradual degradation of the fabricated hardware over time due to different phenomena like Bias Temperature Instability (BTI) and Hot Career Injection (HCI).

This chapter presents a novel technique for mitigating permanent faults in the processing array of a systolic array-based DNN accelerator. First, Section 6.1 presents the motivation behind this work. Then, Section 6.2 presents a novel saliency-driven fault-aware mapping technique for salvaging DNN accelerators having permanent faults. The section also presents different hardware modifications for supporting the proposed mitigation technique under different scenarios. It also explains how the filters/neurons in a DNN layer can be permuted for realizing fault-aware mapping without any run-time overheads. Towards the end, Section 6.3 highlights the effectiveness of the proposed technique by evaluating it for different DNNs and fault rates.

6.1 Motivation and Problem Identification

DNN accelerators, such as TPU [JYP+17] and Eyeriss [CYES19], are fabricated using nanoscale technologies. The imperfections in the nanoscale manufacturing processes

result in defects in the fabricated chips. These defects can take a wide variety of forms, from permanent faults (e.g., stuck-at faults) that affect the functionality of the chips to variations that affect only the operating characteristics (e.g., through variability-driven voltage and frequency guardbands to avoid timing errors) [Con03][HBD+13]. Moreover, technology scaling that has played a vital role in improving the performance and efficiency of digital systems offers these benefits at the cost of increased fault rates (related to both permanent and transient faults). Prior works have highlighted that permanent faults in DNN accelerators can significantly degrade the accuracy of DNNs [ZGBG18]. These faults can be detected using post-fabrication testing. However, discarding every defected chip affects the yield and increases the average per-unit cost of the chips [ZGBG18].

Improving the yield is one of the foremost challenges towards reducing the per-unit cost of DNN accelerators. This is specifically important for devices manufactured using smaller process nodes [KK98]. The significance of this challenge, in general, increases with an increase in the size of the chip, aggressive technology scaling, and the integration density [KK98]. Therefore, in the current nanometer regime, where the number of Processing Elements (PEs)/cores per chip has drastically increased to meet the performance requirements of modern applications, many integrated circuit (IC) manufacturing companies sell their faulty chips under low performance categories after making necessary modifications and without revealing this information to the end-users. Two of the most prominent techniques, that are widely studied (and employed) for yield enhancement, are:

- **Product binning:** This approach is mainly used to categorize the manufactured products based on their characteristics observed during post-fabrication testing [SKMB03][pro]. The binning process is used to reduce the large variances in the characteristics by dividing the products into categories with smaller variances. This process is commonly used by chip manufactures such as Intel and AMD to improve their manufacturing yield [Hru] [Wit].
- **Redundancy:** In this approach, spare components are added in the design to replace the defected ones after post-fabrication testing [SKMB03][ZHXL08][MW07].

Apart from the aforementioned techniques, several other techniques have been studied that offer yield enhancement at the cost of some performance loss [KR89][KL83]. Recently, Zhang et al. in [ZGBG18] proposed Fault-Aware Pruning (FAP) specifically for mitigating the effects of permanent faults in the processing array of systolic array-based DNN accelerators. The technique offers fault mitigation without affecting the performance of the system. The method works on the principle of dropping the computations associated with parameters that are mapped on faulty MAC units. Note that faults in the fabricated hardware can be detected using an initial BIST (built-in-self-test) like method [AKS93][FFR16]. Zhang et al. in [ZGBG18] also proposed hardware modifications that enable *bypassing* of faulty computational units at low cost to realize the FAP method. Further, they proposed a Fault-Aware Pruning + Training (FAP+T) approach

that offers better accuracy due to fault-aware retraining of the model. However, these methods "tend to either be ineffective or lack efficiency" in situations where any of the following conditions hold.

1. *The additional circuitry used for bypassing the MAC units is faulty.* In this case, the faults can propagate to the output of the hardware and result in accuracy degradation.
2. *The number of chips manufactured is large and the number of computational units in the DNN accelerator design is also significant.* In this case, the number of faulty chips can be large with each having a distinct fault map. For example, Fig. 6.1a shows the possible number of distinct fault maps that can exist for different systolic array sizes, and Fig. 6.1b shows the possible number of distinct fault maps for different array sizes and having the total number of faulty PEs less than or equal to five. As can be seen from the figure, with an increase in the size of the systolic array, the number of possible fault maps increases exponentially. Moreover, the number of possible fault maps for larger array sizes is greater than the number of chips usually manufactured of the same type/configuration. In such a case, chances of each faulty chip having a distinct fault map are significant, and fine-tuning/training a neural network for each faulty chip is impractical due to their gigantic computational overheads, specifically for deeper and complex DNNs (see Fig. 6.2).
3. *The training dataset is not available*, e.g., in a case where a vendor has released a trained DNN but has not made the training data available. This limits the applicability of fault-aware retraining, as dataset generation is a costly and time-consuming process. Moreover, training/fine-tuning with a smaller dataset can lead to sub-optimal performance.
4. *The fault map is changing with time*, e.g., due to aging.

In all the above-mentioned scenarios, the state-of-the-art techniques result in undesirable accuracy degradation or significant training/fine-tuning overheads. *Therefore, there is a need for a technique that can mitigate the effects of permanent faults without any significant design-time or run-time costs.*

6.2 SalvageDNNs: A Methodology for Salvaging DNN Accelerators using Fault-Aware Mapping

This section presents a novel methodology for salvaging DNN accelerators having permanent faults in the computational array of a systolic array-based DNN accelerator using saliency-driven fault-aware mapping, without requiring extensive retraining as typically required by state-of-the-art. Figure 6.3 illustrates the overview of the methodology. In Step ❶, the DNN accelerator design is modified such that the components having permanent faults can be disconnected from the main datapath to mitigate the effects

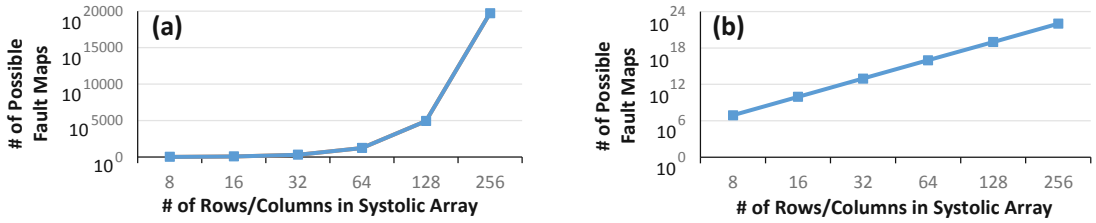


Figure 6.1: Trends illustrating the relation between the possible number of fault maps vs. the number of rows/columns in a systolic array: (a) when *number of faulty PEs* $\leq$ *total PEs in the array*, and (b) when *number of faulty PEs* ≤ 5

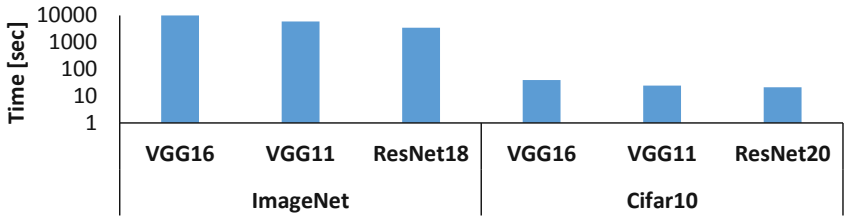


Figure 6.2: Time required for training different DNNs (i.e., VGG16, VGG11, ResNet18, and ResNet20) with different datasets (i.e., ImageNet and Cifar-10) for one epoch using a Core i7 machine with one GTX1080Ti.

of the faults. In Step ❷, the saliency of the weights of the input DNN is computed. Based on the saliency, dataflow, architectural characteristics of the modified accelerator design and the fault map of the fabricated hardware, Step ❸ then defines mapping of neurons/filters of a layer of the input DNN on different segments of the hardware. The mapping is defined such that the sum of saliency of the weights that have to be pruned (mapped on the faulty/disconnected parts of the hardware) is minimized. Step ❹ makes the required modifications in the input DNN by permuting the filters/neurons based on the mapping defined by Step ❸ to avoid any run-time overheads due to highly memory intensive data rearrangement operations. Steps ❸ and ❹ are repeated for each layer (starting from the first layer) and the resultant network is forwarded to Step ❺ after setting all the weights that correspond to faulty/disconnected computational units based on the given dataflow scheme to zero. In Step ❺, required adjustments are made to DNN parameters to (partially) compensate for the accuracy loss incurred due to pruning. The details of these steps are presented in the following subsections.

6.2.1 Hardware Design Optimization/Enhancements for Permanent Fault Mitigation

Fig. 6.4c highlights the hardware modifications proposed in [ZGBG18] for realizing FAP approach.

The baseline systolic array architecture and the Processing Element (PE) design are

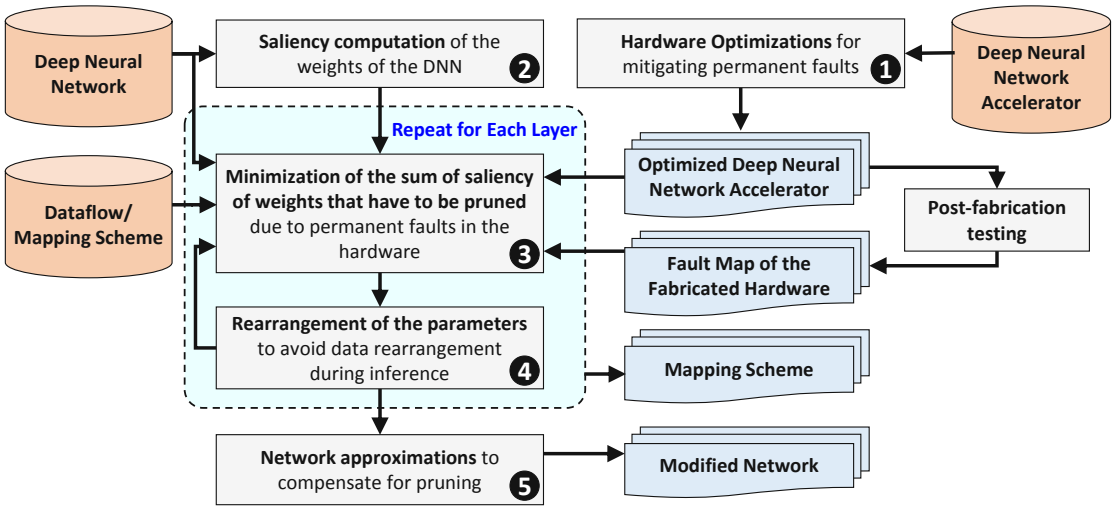


Figure 6.3: Overview of the proposed SalvageDNN methodology for saliency-driven fault-aware mapping of DNNs on a hardware with permanent faults.

shown in Figure 6.4a and 6.4b, respectively. The additional multiplexer in Fig. 6.4c is inserted to bypass the MAC unit in case it is faulty. The proposed modification can help mitigate the effects of permanent faults in the MAC units, but faults can occur in other hardware component as well, e.g., the added multiplexer circuitry. In case a fault occurs in the multiplexer, the design in Fig. 6.4c is incapable of mitigating it. To overcome this limitation, different hardware enhancements are studied in this work. Note that the PE design in Fig. 6.4c, from here onward, is also referred as SOA_PE.

Figs. 6.4d, 6.4e and 6.4f show three different architectural modifications studied in this work. These designs offer different trade-offs between the hardware overhead vs. the average amount of MAC units dropped/disconnected from the datapath in case of a fault. Note that dropping MAC units impacts the DNN inference accuracy directly, as it corresponds to dropping corresponding weights mapped onto those MAC units. Fig. 6.4d shows the design where, in case of a fault in the MAC unit inside a PE, the complete set of MAC units above and including the MAC in the faulty PE are *cutoff* from the datapath. In case of a fault in the multiplexer of a PE, additionally the MAC unit in the subsequent PE is also disconnected by using its multiplexer. This design is, henceforth, referred as C_PE in the thesis. The main advantage of this design is that it incurs low hardware overhead. However, the DNN accuracy can drop significantly, depending on the locations and number of faults. Fig. 6.4e illustrates a hybrid design that combines the best of both worlds, i.e., in case of a fault in the MAC unit of a PE, the MAC unit can be *bypassed* and, in case of a fault in the multiplexer, the complete set of MAC units including the MAC in the subsequent PE can be *cutoff* from the datapath. This design, from here onward, is referred as BNC_PE. The main advantage of this design is that it can handle faults in the MAC units as well as the multiplexers without much overhead cost compared to that of the design shown in Fig. 6.4c.

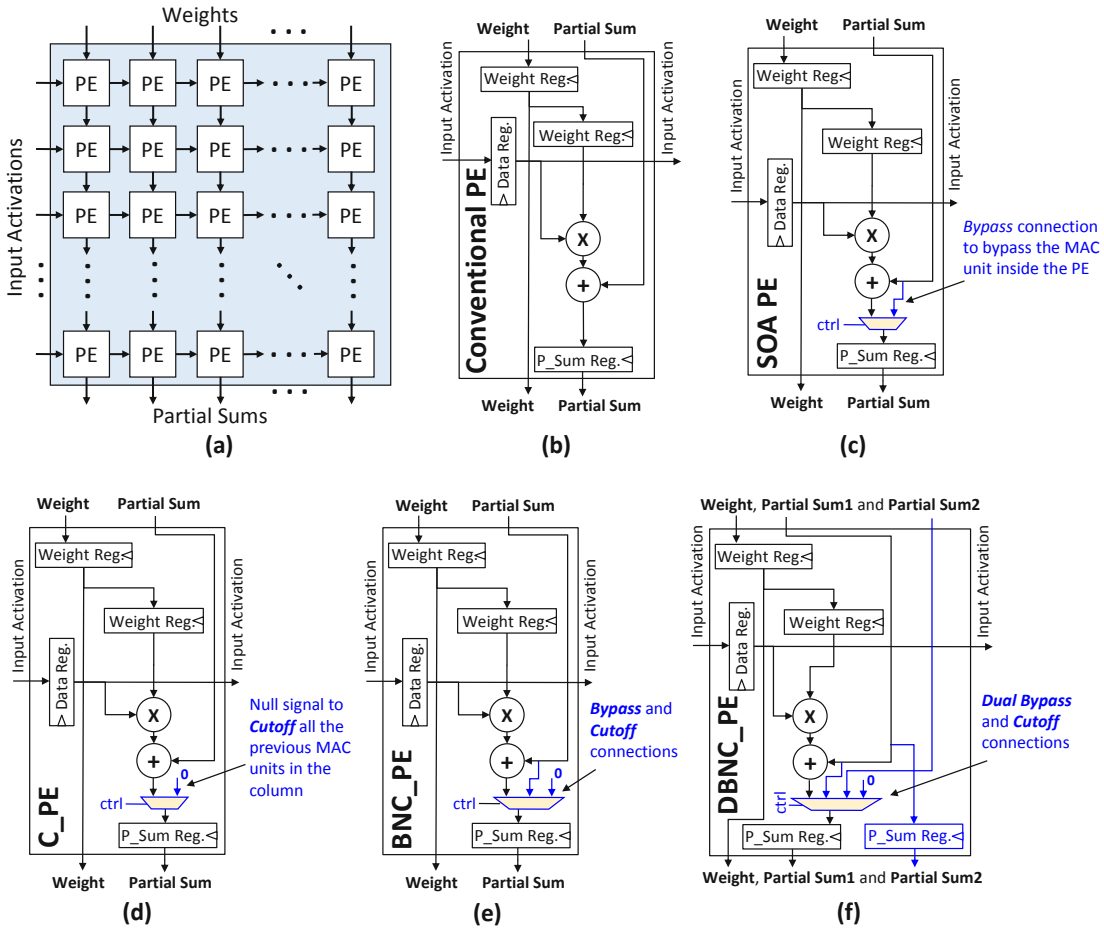


Figure 6.4: (a) The baseline systolic array design similar to the TPU. (b) The conventional PE design. (c) The modified PE proposed in [ZGBG18] for permanent fault mitigation using the FAP and the FAP+T techniques. (d), (e) and (f) show our novel additional PE designs for handing permanent faults. Note, the changes in the PEs with respect to the conventional PE design, i.e., (b), are shown in color.

Fig. 6.4f shows the third design, which is a modified version of BNC_PE. It contains an additional connection compared to BNC_PE for *bypassing two adjacent MAC units* in a column, which enables us to bypass the intermediate multiplexer in case it is faulty. This design is referred as $DBNC\_PE$ in the following sections. Similar to $DBNC\_PE$, a design can be made where n adjacent MAC units can be bypassed. However, there are two main shortcomings of this approach: (1) it leads to high overheads due to the need of a larger multiplexer and additional registers required for partial sums to maintain the computational flow; and (2) if n consecutive PEs have faults in their multiplexers, error can still propagate to the output. Due to these limitations and considering the fact that fault rates are usually very low, in this work, we limit n to 2, as shown in Fig. 6.4f.

6.2.2 Saliency Computation of DNN Parameters

This work studies the impact of two different saliency computation methods on the effectiveness of the proposed methodology. The methods are as follows.

1. **Norm-based method:** In this approach, L1-norm (or L2-norm) of the weights, neurons, and/or filters of the given DNN are computed. The L1/L2 scores define the relative importance of the parameters.

2. **Importance score propagation-based method:** In this approach, the importance of each weight/neuron/filter is computed using its L1-norm and the saliency of the connections between the corresponding neuron/filter and the output of the DNN. Let $s_i^{<l>}$ defines the saliency of the connections between the i^{th} neuron/filter in the l^{th} layer of the given DNN and the output of the DNN. The saliency of a weight in a neuron/filter can be computed by multiplying the L1-norm of the weight with the saliency of the connections between the neuron/filter and the output of the DNN, i.e., in l^{th} fully-connected layer it is computed by $|W_{(i,j)}^{<l>}| \times s_i^{<l>}$ and in l^{th} convolutional layer it is computed by $|W_{\{(r,c),(C,i)\}}^{<l>}| \times s_i^{<l>}$. The saliency of the connections between a neurons/filters in l^{th} layer of the given DNN and the output of the DNN can be computed by:

$$s_i^{<l>} = \sum_k |W_{(k,i)}^{<l+1>}| \times s_k^{<l+1>} \tag{6.1}$$

in case $l + 1^{th}$ layer is a fully-connected layer, and it is computed by

$$s_i^{<l>} = \sum_k \sum_r \sum_c |W_{(r,c),(i,k)}^{<l+1>}| \times s_k^{<l+1>} \tag{6.2}$$

in case $l + 1^{th}$ layer is a convolutional layer. In this work, $s_i^{<last\,layer>} = 1\ \forall\ i \in \{1, 2, ...N^{<last\,layer>}\}$.

Note, the aforementioned saliency computation methods are highly effective for comparing the saliencies of weights within the same layer, i.e., not across layers. As the layers are processed sequentially in most of the DNN accelerators, only the relative importance of weights within a layer is required. Hence, the above the mentioned approaches are sufficient for evaluating the importance of weights of a DNN.

6.2.3 Minimization of the Sum of Saliency of Pruned Weights

The objective of SalvageDNN is to map the least important weights on the components disconnected from the datapath. To achieve this goal, the following steps are followed for each layer of the given DNN.

Algorithm 6.1: A Fast Method to Reduce the Sum of Saliency of the Weights of a Layer that have to be Pruned due to Permanent Faults.

Inputs: A matrix ***Costs*** containing the costs of mapping neurons/filters to faulty columns of the array and a vector ***Idx*** containing the indexes of faulty columns of the array
Outputs: A matrix ***Mapping*** which defines which neuron/filter has to be mapped to which faulty column and a variable *TCost* which defines the cost of this mapping scheme
Initialize: *TCost* = 0, ***Mapping*** = Zeros(Number of Columns in ***Costs***, 2) and ***NF_Idx*** = [1, 2, 3, ... Number of neurons/filters in the layer]
for *i = 1 to Number of columns in* ***Costs*** *at the start of the loop* **do**
    [*val*, *row_id*, *col_id*] = min(***Costs***)
    Mapping(*i*,1) = ***NF_Idx***(*row_id*)
    Mapping(*i*,2) = ***Idx***(*col_id*)
    TCost = *TCost* + *val*
    Costs ⇐ ***Costs*** after removing *row_id* row and *col_id* column
    Idx ⇐ ***Idx*** after removing *col_id* column
    NF_Idx ⇐ ***NF_Idx*** after removing *row_id* column
end
return *TCost* and ***Mapping***

- **Disconnection Map (DM) Generation:** DM is a matrix that defines which MAC units are disconnected from the datapath in the processing array due to permanent faults. The map is generated by setting the values corresponding to the MACs that are disconnected from the array to 1. For example, if the MAC unit in the first row and first column of the array is disconnected, the $DM(1,1)$ is set to 1; otherwise, it is set to 0. The DM is constructed using the fault map of the array. The fault map is represented using two matrices FM_{MAC} and FM_{MUX}, where FM_{MAC} keeps track of the faulty MAC units and FM_{MUX} keeps track of the faulty multiplexers in the array. Note that the sizes of DM, FM_{MAC} and FM_{MUX} are the same as the size of the array, and there is one-to-one mapping between the elements of the matrix and respective hardware components in the systolic array.

- **Unrolling the Saliencies of Weights of a Layer into a Matrix S:** The saliency values after computation are organized in the same format as the weights of the DNN. For each layer individually, the multidimensional saliency tensor is required to be unrolled and stored in a 2D matrix, **S**. After unrolling, the matrix **S** contains the saliencies of the weights of a layer in flattened form, i.e., each column in the matrix contains saliency values of the weights of a neuron/filter. Note that the unrolling is performed based on a pre-defined dataflow that is provided as input by the user.

- **Pruning Matrix (PM) Generation:** The DM matrix is replicated in x and y dimensions such that the number of columns and rows in the final matrix is at least equivalent to the number of columns and rows in **S**. The additional columns

(from the right) and rows (from the bottom) are then removed, and the resultant matrix is stored in the Pruning Matrix (PM). The columns containing all zeros are removed from the PM while keeping track of the original indexes of the non-zero columns in a vector ***Idx***.

- **Defining Mapping Policy that Minimizes the Sum of Saliency of the Weights to be Pruned:** The objective of this step is to generate a mapping policy that minimizes the sum of saliency of weights to be pruned. This can mathematically be represented as:

$$argmin_{mapping}||S^*.*PM|| \tag{6.3}$$

Here, $\mathbf{S}^*$ represents a transformed version of $\mathbf{S}$ generated after applying rearrangement of network parameters based on the mapping policy presented in Section 6.2.4. The '.*' operation represents element-wise multiplication of two matrices. In this step, Algorithm 6.1 or Algorithm **??** are used to find the mapping sequence of neurons/filters. Algorithm 6.1 is based on a greedy approach for fast generation of an acceptable solution. However, Algorithm **??** is based on a branch and bound approach to propose an optimal solution. The detailed descriptions of Algorithm 6.1 and Algorithm **??** are as follows.

Algorithm 6.1 presents a naive method to find a sub-optimal mapping policy. The algorithm takes the ***Costs*** matrix and the ***Idx*** vector as inputs. The ***Costs*** matrix contains the costs of mapping each neuron/filter to each faulty column of the systolic array, and the ***Idx*** vector contains the indexes of the faulty columns of the array. The ***Costs*** matrix is computed through matrix multiplication of *transpose* of ***S*** with ***PM***. Each row of the ***Costs*** matrix contains the cost of mapping a filter to each faulty column of the array individually. The algorithm, at each iteration, finds the minimum value in the matrix (line 5) and then associates the corresponding neuron/filter index from ***NF_Idx*** with the index of the column of the array from ***Idx*** (lines 6 and 7). The costs associated with the selected filter and the column of the array are then removed from the ***Costs*** matrix (lines 9). Similarly, the index vectors are also updated (lines 10 and 11). The resultant matrix and vectors are then used in the next iteration. The algorithm iterates for the number of filters to be mapped on the faulty columns (line 4), i.e., the number of columns in the ***Costs*** matrix, and outputs the mapping (***Mapping***) and the corresponding total cost ($TCost$).

Algorithm ?? presents a more robust approach for finding a suitable mapping policy that offers minimum sum of saliency of weights to be pruned. The algorithm is based on a branch and bound method, which recursively calls (line 19) the *Branch&Bound* function (line 6) with a smaller problem, i.e., smaller ***Costs*** matrix (lines 14 till 17). The algorithm keeps track of the cost of the already selected pairs in CS_Cost and restricts the algorithm from searching in the same branch if the overall cost is more than the cost of the reference mapping ***R_Mapping*** (lines 13 and 29). Note that at beginning of the algorithm, the reference cost is computed using Algorithm 6.1 to speed-up the process.

If, at some point, a mapping that has cost less than the cost of the reference mapping is found, the reference mapping and the reference cost are updated (lines 29 till 31). As the algorithm is meant to search in all the possible branches (combinations) and can take endless amount of time for larger ***Costs*** matrices, the search is limited by applying a limit on the number of miss hits (*MH_Count*), i.e., *Termination_Limit* (line 21), from the last reference mapping update. The algorithm also employs a *per branch search limit* (line 10) to decrease the number of searches in each branch and search only in a defined number of nodes that offer minimum costs (lines 9 and 10).

6.2.4 Rearrangement of the Network Parameters

In most of the DNNs, the neurons/filters within a layer can be permuted without affecting the functionality of the DNN. Fig. 6.5 illustrates this with the help of an example in which two neurons in an FC layer are swapped. Fig. 6.6 illustrates a similar example for a CONV layer in which two filters are swapped. To maintain the functionality, the corresponding connections in the subsequent layer of the DNN are also swapped. This property helps in maintaining the original structure and the dataflow inside the accelerator without requiring additional data arrangement operations. The only exceptions are the DNNs that have Local Response Normalization (LRN) layers, e.g., the AlexNet [KSH12]. In such networks the computing sequence in the accelerator has to be slightly modified and additional data arrangement operations might be required that can affect the efficiency of the system. However, almost all the modern DNNs do not employ LRN layers, and hence they can directly benefit from the proposed methodology.

6.2.5 Network Approximations (Optional)

After rearrangement and setting the corresponding weights that have to be mapped on the disconnected computational units to zero, we compute the mean of the output activations of the neurons and filters using a small subset of the validation dataset. The mean values are compared with the mean values acquired through the original DNN, and based on their difference, the biases of the neurons/filters are adjusted to partially compensate for the effects of pruning. Note that, in case, the training dataset and sufficient computational resources are available, fine-tuning can be employed instead for regaining a fraction of the lost accuracy.

6.2.6 SalvageDNN under Static and Dynamic Conditions

This section explains with the help of examples how the proposed methodology helps salvage a faulty DNN hardware. The practical cases can be broadly represented using two different scenarios: (1) where the fault maps are extracted only after fabrication during post-fabrication testing and each chip can have a distinct fault map; and (2) where the fault map of a chip is changing over time due to aging and can be extracted using BIST support available in the chip. Note that, for ease of understanding, in both the scenarios,

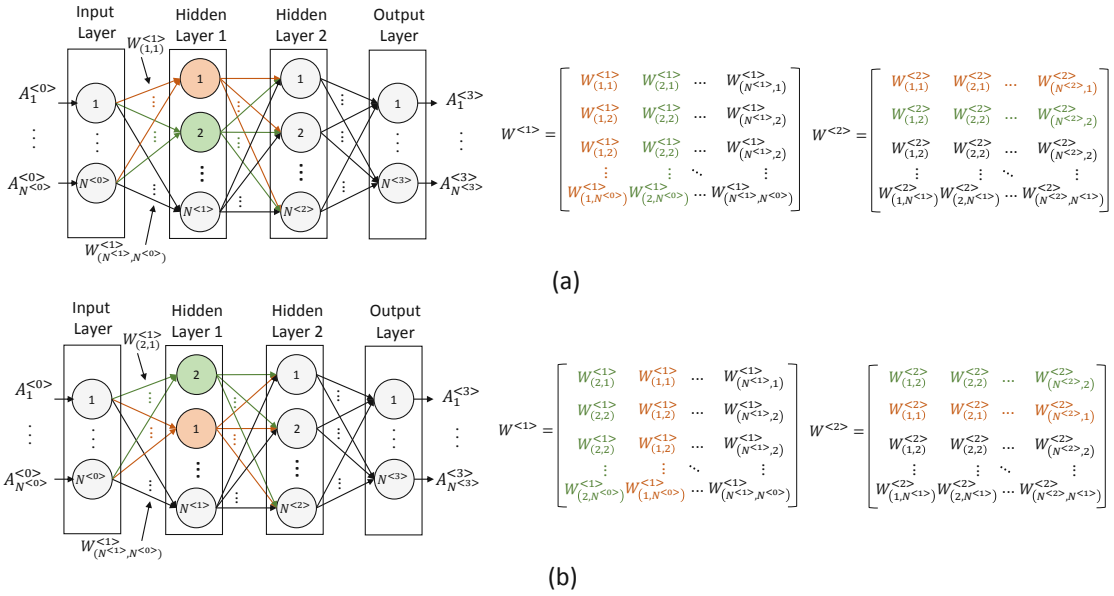


Figure 6.5: Impact of rearranging neurons in a layer of a fully-connected DNN on the arrangement of the weights to be mapped on the systolic array. (a) shows the arrangement before swapping neurons 1 and 2 in the first hidden layer of a fully-connected DNN, and (b) shows the arrangement after swapping the neurons. The left side of the figure illustrates the state of the neural network and the right side shows the weights of the first and second hidden layers in a manner in which they will be mapped on a systolic array. Different colors are used to show the association between the neurons and weights.

it is assumed that only the faulty PEs are bypassed to avoid fault propagation. Moreover, the saliency of the DNN parameters is computed using L1-norm.

Scenario 1: Different hardware chips can have different fault maps, i.e., the fault maps can vary across chips. To explain how SalvageDNN handles such a scenario, consider an example where four filters (see Fig. 6.7a), each having four values (i.e., weights), have to be mapped on a 4x4 systolic array to perform the dot-product operations. The filters are unrolled and mapped on the systolic array based on the baseline systolic array design shown in Fig. 6.4a. Fig. 6.7 highlights four different example cases, each representing a systolic array belonging to a different chip. The cases are explained as follows:

- Case 1: None of the PEs in the array is faulty (see Fig. 6.7b)
- Case 2: The array has two faulty PEs (see Fig. 6.7c)
- Case 3: The array has the same number of faulty PEs as in Case 2, but at different locations (see Fig. 6.7d)
- Case 4: The array has a higher number of faulty PEs than Case 2 and Case 3, and also at different locations (see Fig. 6.7e).

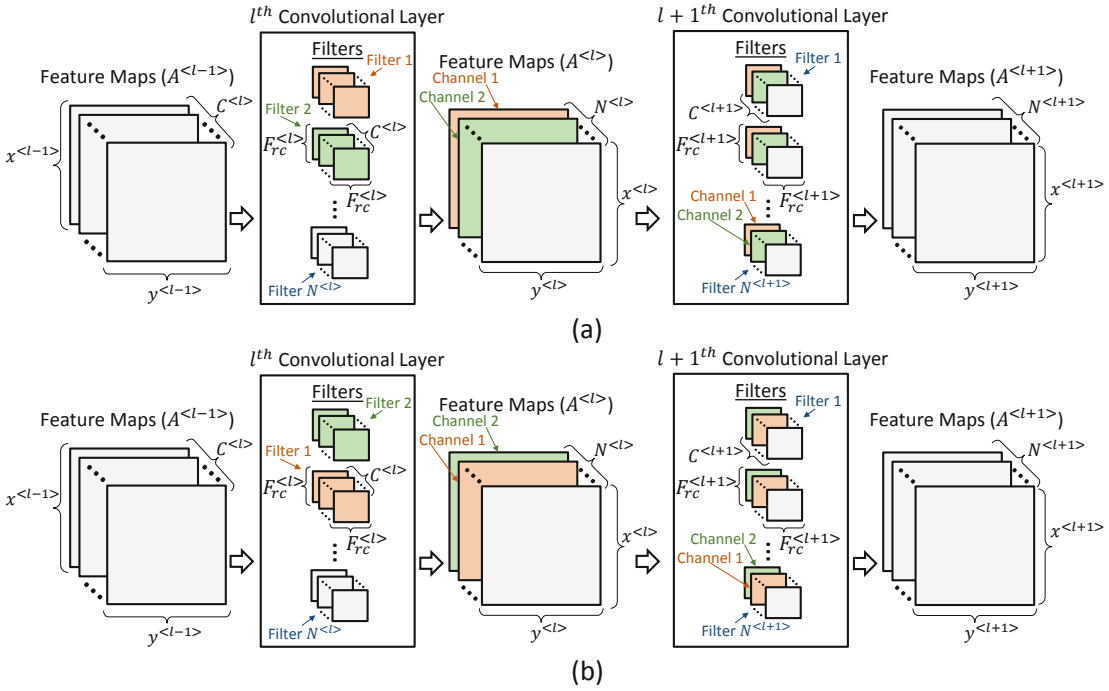


Figure 6.6: Impact of rearranging filters in a layer of a CNN on the arrangement of the weights in the neural network. (a) shows the arrangement of filters and their channels before swapping filters 1 and 2 in the l^{th} convolutional layer of a CNN, and (b) shows the arrangement after swapping the filters. Note that a swap of filters in the l^{th} layer of a CNN requires a swap of the respective channels in the $l+1^{th}$ layer of the CNN to maintain the functionality.

In Case 1 (Fig. 6.7b), the mapping does not affect the accuracy of the results, as there are no faulty PEs in the array, which have to be bypassed during execution. In Case 2 (Fig. 6.7c), the filters are mapped onto the columns such that the sum of absolute weights mapped on the faulty PEs is minimum, which in this case is achieved by mapping filters 1, 2, 3, and 4 on columns 2, 1, 4, and 3 respectively, of the systolic array. In Case 3 (Fig. 6.7d), the fault map of the array is different from the ones presented in earlier two cases. Therefore, the mapping can be different, depending on the fault map and the filter values. As can be seen in Fig. 6.7d, to minimize the sum of absolute weights mapped onto the faulty PEs, filters 1, 2, 3, and 4 are now mapped on columns 2, 3, 4 and 1 respectively. Similarly, in Case 4 (Fig. 6.7e) the mapping is defined based on its fault map. Here, it is important to clarify that, similar to the state-of-the-art FAP technique [ZGBG18], SalvageDNN also adjusts the DNN for each faulty chip individually based on its fault map. However, unlike state of the art, SalvageDNN does not employ any retraining, and therefore, requires less time and resources for making the adjustments compared to the retraining-based approaches, as shown later in Section 6.3.2.5. Moreover, SalvageDNN also does not require access to the training dataset, which in most of the practical cases is not available (as stated in Section 6.1). Details regarding how the proposed technique

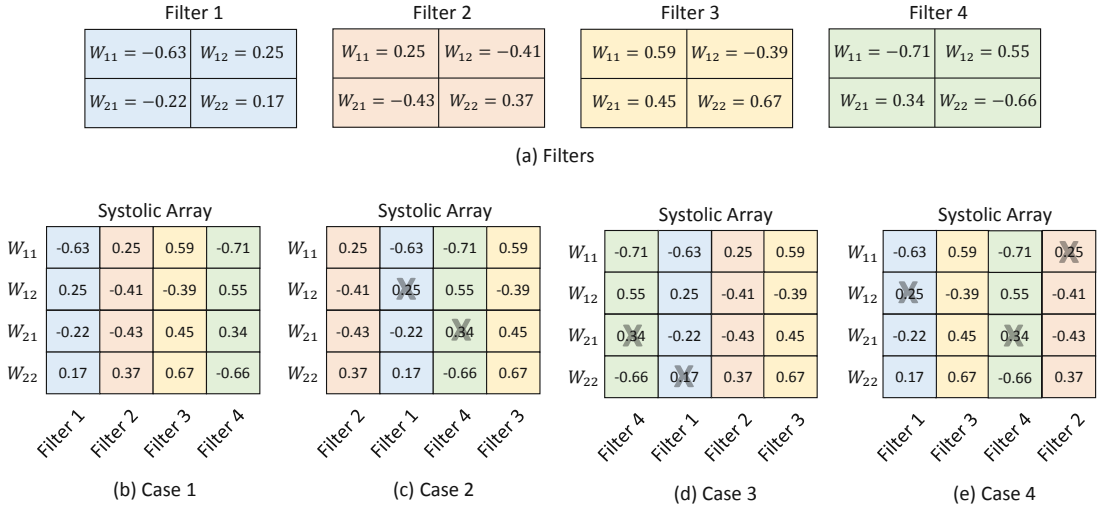


Figure 6.7: An example illustration of how the mapping would vary across chips having different fault maps. The grids shown in b, c, d, and e corresponds to 4x4 systolic arrays from 4 different chips. Each small box inside a grid represents a single Processing Element (PE). The PEs with black cross (x) over them in c, d, and e correspond to faulty PEs. The filters considered in this example are shown in a.

minimizes the cost are presented in Algorithm 6.1 and Algorithm **??**.

Scenario 2: The fault maps can change over time, e.g., due to wear-out. To explain how SalvageDNN can be used in this scenario without incurring high overheads, consider an example where four filters (see Fig. 6.8a), same as shown in Fig. 6.7a, have to be mapped on a 4x4 systolic array. Fig. 6.8 illustrates three different cases that are:

- Case 1: None of the PEs in the systolic array is faulty (see Fig. 6.8b)
- Case 2: The array has two faulty PEs (see Fig. 6.8c)
- Case 3: With time, the number of faulty PEs in the array presented in Case 2 has increased to three (see Fig. 6.8d)

Cases 1 and 2 are the same as presented in Fig. 6.7. In Case 1, the mapping policy does not affect the accuracy of the results as there are no faulty PEs in the array. In Case 2, filters 1, 2, 3, and 4 are mapped on columns 2, 1, 4, and 3 respectively (see Fig. 6.8c), to minimize the sum of saliency of weights that are mapped onto the faulty PEs. However, in Case 3, once the fault map is updated using BIST, the mapping of filters should also be updated to achieve a lower cost. As can be seen in Fig. 6.8d, the minimum sum is achieved by mapping filters 1, 2, 3, and 4 on columns 3, 1, 2, and 4 respectively. Note that, with the mapping policy of Case 2 and the fault map of Case 3, the sum of absolute weights mapped on the faulty PEs would have been 1.25, while with the updated mapping, the sum is 0.78, as can be observed in Fig. 6.8d.

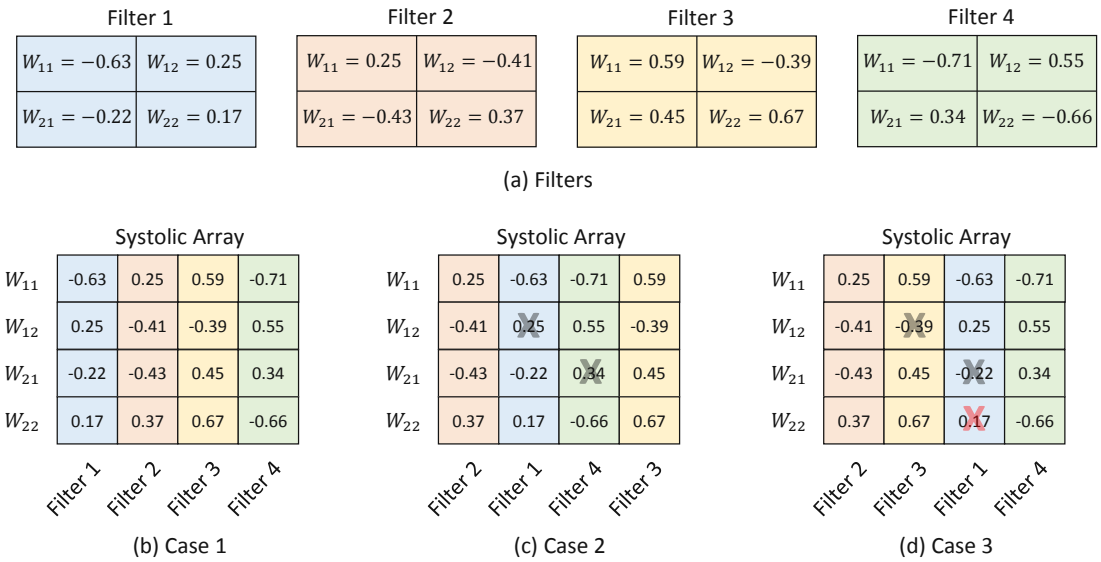


Figure 6.8: An example illustration of how the mapping would change if the fault map of the DNN hardware changes over time. The grid shown in b, c, and d corresponds to a systolic array, where each small box represents a single Processing Element (PE). The PEs with black cross (x) over them in c and d correspond to faulty PEs detected during post-fabrication testing and the PE with red cross (x) over it in d correspond to the PE which experienced fault over time due to wear-out.

Associated testing and re-configuring overheads: Post-fabrication testing is an essential part of the manufacturing process, which is required to verify the functional correctness of the fabricated hardware. It is also used for identifying faulty components in the hardware [PBGM09][NHK+99][SV05]. Therefore, no additional cost is usually required to extract the initial fault map. As Scenario 1 considered only the initial fault maps, no additional effort is required, apart from the usual cost necessary for post-fabrication testing. The extracted fault map are used, in our case, for re-configuring the fault tolerant systolic arrays, as well as for determining the DNN mapping using the proposed, SalvageDNN, methodology. Note that storing the fault map requires some memory. However, it is relatively small (e.g., one bit per PE) as it completely depends on the size and type of the systolic array. Unlike Scenario 1, Scenario 2 requires online BIST support to extract the updated fault map at regular intervals. The cost of such a support is usually high; however, it can be reduced by using algorithm-based fault detection and localization techniques [ABF+87]. The algorithm-based techniques have proven to be very effective for regular hardware structures composed of homogeneous components, such as systolic arrays.

6.3 Results and Discussion

6.3.1 Hardware Synthesis Results

Table 6.1 presents the hardware characteristics of the conventional PEs and *SOA PEs* designed for different systolic array sizes. The results are generated following an ASIC design flow using Cadence Genus tool with the TSMC 65nm technology library. As can be seen in the table, in all the listed systolic array sizes, the MAC units consume approximately 66% of the area of the respective PEs, and the multiplexer inside each *SOA PE* consumes approximately 6% of the area of the respective PE. This shows that mitigating permanent faults in the MAC and MUX units of the array can significantly improve the yield of DNN hardware manufacturing process.

Table 6.1: Hardware characteristics of the components used in different sizes of the baseline and the state-of-the-art fault-tolerant systolic arrays. The bit-widths of the partial sums in 8x8, 16x16, 32x32 and 256x256 arrays were set to 19, 20, 21 and 24, respectively, assuming the bit-width of weights and activations to be 8-bit.

		Area [cell area]	Delay [ns]	Power [μW]
8x8 Array	Baseline PE	931	2.17	76.77
	SOA PE	972	2.21	80.10
	MAC	620	1.97	62.11
	2-to-1 MUX	56	0.24	2.69
16x16 Array	Baseline PE	950	2.23	77.09
	SOA PE	993	2.26	82.09
	MAC	632	2.01	62.01
	2-to-1 MUX	59	0.25	2.88
32x32 Array	Baseline PE	965	2.32	77.62
	SOA PE	1010	2.35	81.10
	MAC	640	2.10	61.10
	2-to-1 MUX	62	0.25	2.90
256x256 Array	Baseline PE	1006	2.65	77.40
	SOA PE	1058	2.68	80.64
	MAC	661	2.44	61.66
	2-to-1 MUX	70	0.28	3.39

Fig. 6.9 presents a comparison of the hardware characteristics of the PEs designed for different sizes and types of the systolic arrays discussed in Section 6.2.1. The figure shows that the fault-tolerant PEs consume slightly more area as compared to the corresponding baseline PEs, depending on the complexity of the bypassing/cutoff circuitry in the design. For example, a PE with dual bypass and cutoff circuitry (i.e., *DBNC_PE*) consumes approximately 25% more area, and a PE with cutoff circuitry (i.e., *C_PE*) consumes approximately 1% additional area, when compared to the corresponding baseline PE used for the same size of the systolic array. The variations in PE area with array size are due to the fact that the bit-width of the partial sums increases with an increase in the number of rows in the array. Similar to the area characteristics, the critical path delay of a PE also depends on the design and the size of the systolic array. However,

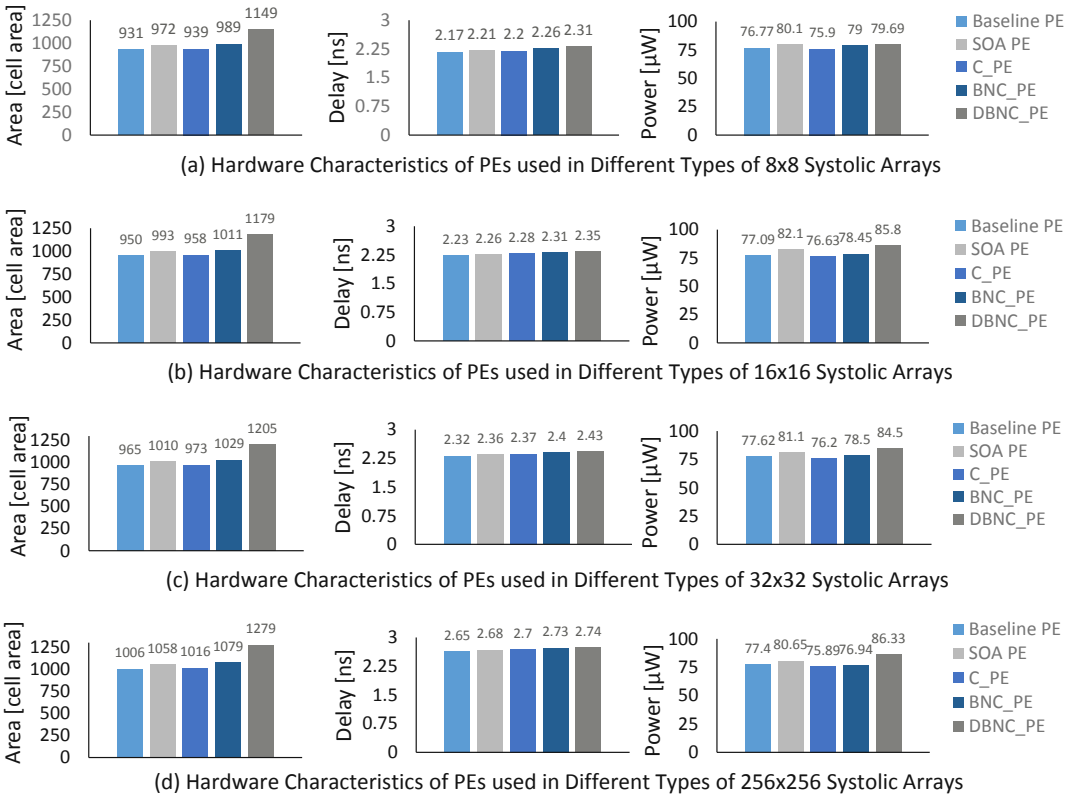


Figure 6.9: Hardware synthesis results of the PEs of different sizes and types of systolic array designs.

the variations in the delay are relatively small, i.e., at the maximum 6.5% for the 8x8 systolic array composed of *DBNC_PE*, when compared to the baseline PE belonging to the same sized systolic array. The trend in the power characteristics is approximately similar to the trend observed in the area characteristics of the PEs. Hence, the options illustrated in Fig. 6.4, Section 6.2.1 provide different design trade-offs between hardware characteristics (i.e., area, latency, power) and resilience.

6.3.2 Impact of Fault-Aware Mapping (SalvageDNN) on DNN Accuracy

6.3.2.1 Experimental Setup

For analyzing the effectiveness of SalvageDNN for mitigating the effects of permanent faults, different variants of VGG model (i.e., VGG11 trained with the Cifar-10 [KH+09], VGG11 trained with ImageNet [DDS+09] and VGG16 trained with ImageNet [DDS+09]) are considered in this work. The pre-trained DNNs are taken from the *Neural Network distiller* [ZJZ+18], which is an open-source tool for neural network compression. The details of the models are presented in Table 7.2, and a few key characteristics of the

datasets are highlighted in Table 6.3. To quantize the DNN weights and activations to 8-bits, the same *neural network distiller* tool is employed. For fault injection, similar to the well-established works in the reliability community, a random distribution of the faults across the array is considered. For each experiment, multiple iterations are performed using different seed values. To achieve a fair comparison, same seed values are used across experiments involving different types and sizes of arrays as well as fault rates. Moreover, the relative sizes of MAC and MUX units are also taken into consideration for distributing faults in the arrays.

Table 6.2: DNNs and the datasets used for evaluation.

Network and Dataset	Network Architecture	Baseline Accuracy (%)
VGG11 for Cifar-10	Layers: CONV(64, 3, 3, 3); CONV(128, 64, 3, 3); CONV(256, 128, 3, 3); CONV(256, 256, 3, 3); CONV(512, 256, 3, 3); CONV(512, 512, 3, 3); CONV(512, 512, 3, 3); CONV(512, 512, 3, 3); FC(10, 512)	85.38%
VGG11 for ImageNet	Layers: CONV(64, 3, 3, 3); CONV(128, 64, 3, 3); CONV(256, 128, 3, 3); CONV(256, 256, 3, 3); CONV(512, 256, 3, 3); CONV(512, 512, 3, 3); CONV(512, 512, 3, 3); CONV(512, 512, 3, 3); FC(4096, 25088); FC(4096, 4096); FC(1000, 4096)	68.04% (Top1) 88.07% (Top5)
VGG16 for ImageNet	Layers: CONV(64, 3, 3, 3); CONV(64, 64, 3, 3); CONV(128, 64, 3, 3); CONV(128, 128, 3, 3); CONV(256, 128, 3, 3); CONV(256, 256, 3, 3); CONV(256, 256, 3, 3); CONV(512, 256, 3, 3); CONV(512, 512, 3, 3); CONV(512, 512, 3, 3); CONV(512, 512, 3, 3); CONV(512, 512, 3, 3); CONV(512, 512, 3, 3); FC(4096, 25088); FC(4096, 4096); FC(1000, 4096)	70.85% (Top1) 90.0% (Top5)

6.3.2.2 Comparison with State-of-the-art FAP Approach

Fig. 6.10 illustrates the effectiveness of the FAP and the proposed techniques for mitigating the effects of permanent faults in a 256x256 systolic array composed of SOA_PEs used for executing the VGG11 network trained with the Cifar-10 dataset. To have a fair comparison of the proposed method with the state-of-the-art FAP approach, a similar evaluation methodology is adopted as reported in [ZGBG18]. This means, for this evaluation, it is assumed that faults can only occur in the MAC units of the array, and not in the MUXes or other components of the PEs. From this experiment, the following

Table 6.3: A summary of the key characteristics of the datasets used in the evaluation of SalvageDNN and the comparison with the state-of-the-art

Dataset	Characteristics	
Cifar-10	Number of Training Images	50000
	Number of Testing Images	10000
	Size of Images	32 x 32 x 3
	Number of Classes	10
ImageNet	Number of Training Images	~14 million
	Number of Testing Images	100000
	Size of Images	224x224x3
	Number of Classes	1000

key observations can be made:

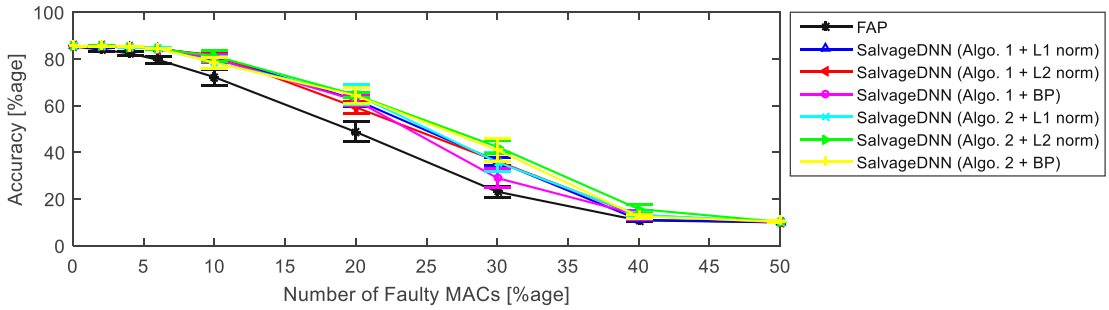


Figure 6.10: Impact of number of faulty MAC units in a 256x256 systolic array, composed of SOA_PEs, on the accuracy of the VGG-11 network trained on the Cifar-10 dataset.

- The FAP approach results in a rapid decrease in the DNN accuracy with increasing fault rate compared to the SalvageDNN methodology.
- At lower fault rates (i.e., below the 6% mark), SalvageDNN helps in maintaining the baseline accuracy of the DNN. However, the DNN accuracy when mapped only using the FAP approach starts decreasing instantly with the increase in the fault rate. This highlights the practical significance of the proposed SalvageDNN methodology, as, due to the advancements in the fabrication processes, typically lower fault rates are observed in practical settings.
- The error bars in the figure, which shows the standard deviation of the accuracy across multiple iterations of the same experiment when performed using a different seed value, highlights that the accuracy is highly dependent on the locations of the faulty PEs in the array. Therefore, the FAP approach, which uses fixed mapping, results in higher standard deviation values compared to the SalvageDNN methodology. This is mainly because SalvageDNN make faults-aware adjustments in the DNNs to map less significant weights on faulty/disconnected PEs.
- The type of the saliency computation method and the algorithm for minimizing the cost function do not impact the result much. Note that Algorithm **??** provides sightly better results at higher fault rates. However, based on the computational requirements of Algorithm **??**, Algorithm 6.1 with L1 norm should be preferred.

6.3.2.3 Comparison with State-of-the-art FAP+T Approach

Fig. 6.11 illustrates the impact of using SalvageDNN along with fault-aware retraining on the accuracy of the VGG11 trained on Cifar10 dataset. The subfigures present a comparison between the "FAP+T" [ZGBG18] and the "SalvageDNN + retraining" techniques. As can be seen from the figure, both the approaches perform equally well.

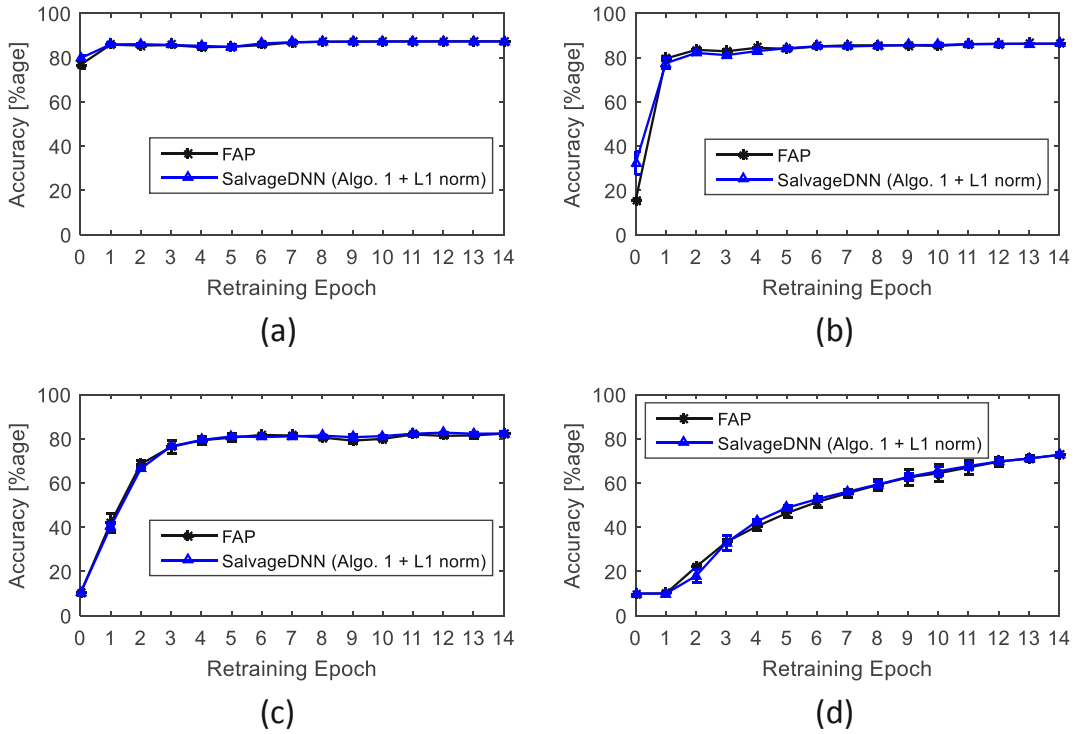


Figure 6.11: Impact of using SalvageDNN before applying fault-aware training on the accuracy of the VGG11 network trained for the Cifar10 classification. The subfigures show the comparison between FAP + retraining and SalvageDNN + retraining when the underlying hardware has: (a) 10% faulty PEs, (b) 30% faulty PEs, (c) 50% faulty PEs, and (d) 70% faulty PEs

Therefore, if the training dataset and sufficient computational resources are available for retraining, then retraining should be employed to regain a significant percentage of the lost accuracy. The number of epochs required to regain the accuracy seems to be dependent on the number of faulty PEs in the array. This can be observed from Fig. 6.11, where Figs. 6.11a, 6.11b, 6.11c, and 6.11d present the test accuracies achieved after every epoch of retraining for the cases when 10%, 30%, 50%, and 70% of the total PEs are faulty respectively. For example, in Fig. 6.11a, the baseline accuracy is regained after only one epoch of retraining; however, in Fig. 6.11c, it took five epochs to reach the saturation point.

6.3.2.4 Evaluation for DNNs Trained on Larger Datasets

Fig. 6.12 illustrates a comparison of the proposed approach with the state-of-the-art FAP approach for the case when both are used for the VGG11 network trained on the ImageNet dataset. The systolic array considered for this evaluation is a 256x256 sized array composed of SOA_PEs. As can be seen from the figure, for lower fault rates, i.e., until around 0.06 (6%), the SalvageDNN methodology helps maintain the baseline

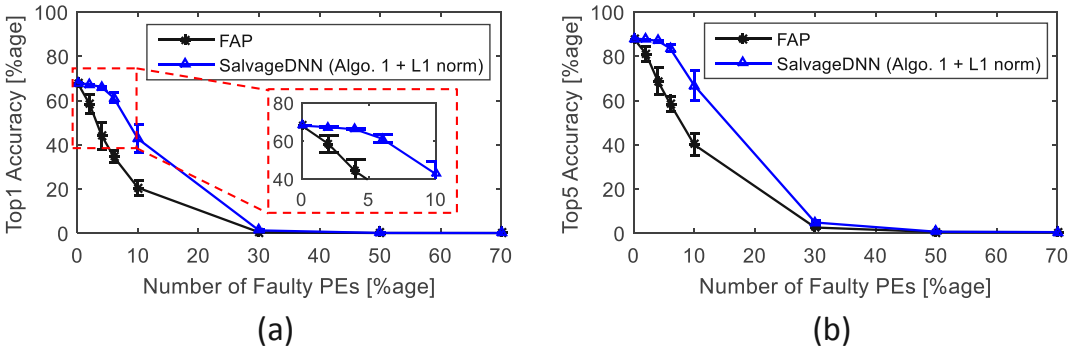


Figure 6.12: Comparison of SalvageDNN with the state-of-the-art FAP approach when used for the VGG11 network trained on the ImageNet dataset, which has to be mapped on a 256x256 sized systolic array. The two commonly used accuracy metrics, i.e., the Top1 and the Top5 accuracies, are shown for cases having different number of faulty PEs, in subfigures (a) and (b) respectively

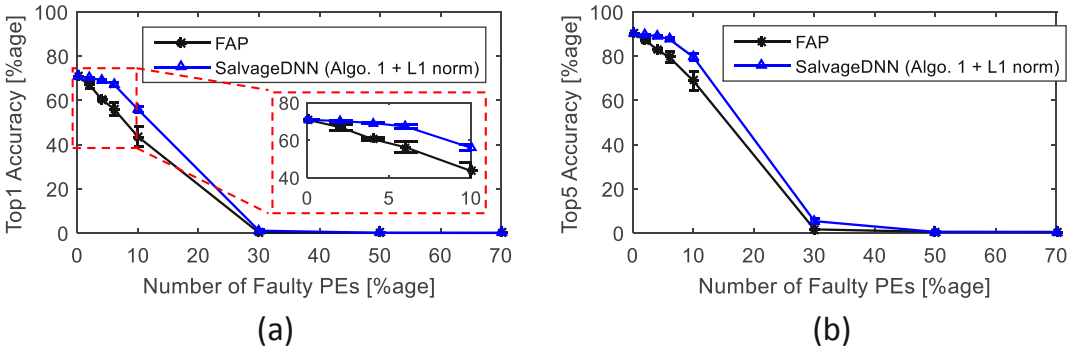


Figure 6.13: Comparison of SalvageDNN with the state-of-the-art FAP approach when used for the VGG16 network trained on the ImageNet dataset, which has to be mapped on a 256x256 sized systolic array. The two commonly used accuracy metrics, i.e., the Top1 and the Top5 accuracies, are shown for cases having different number of faulty PEs, in subfigures (a) and (b) respectively

accuracy, while the accuracy achieved with FAP approach is significantly lower, even when only 2% of the total PEs are faulty. Similar results are observed in Fig. 6.13 for the VGG16 network trained on the ImageNet dataset.

6.3.2.5 Comparison of execution time of SalvageDNN with retraining-based approaches

Table 6.4 presents a comparison between the execution time of the SalvageDNN and the FAP+T for different DNNs. The details of the DNNs and the datasets used in this comparison are presented in Tables 6.2 and 6.3 respectively. As can be seen from

Table 6.4: Execution time comparison of SalvageDNN with retraining-based approach, i.e., FAP+T [ZGBG18], for different networks trained on different datasets

		Execution Time (sec)		SalvageDNN
Network	Dataset	Retraining (Single Epoch)	SalvageDNN (Algo. 1 + L1-norm)	Speedup
VGG11	Cifar-10	24.63	1.12	21.99x
VGG11	ImageNet	6005.93	112.67	53.31x
VGG16	ImageNet	9871.38	113.26	87.16x

the tables, the execution time depends mainly on the complexity and size of the DNN. However, it also depends on the size of the dataset for retraining-based approaches. Note that, in all the cases, SalvageDNN requires an order of magnitude less time for execution than the time required for a single epoch of retraining. Note that the amount of savings grow with the increase in the size and complexity of the dataset and the DNN. For example, the time required for running SalvageDNN for the VGG16 trained on the ImageNet dataset is around 87x lesser than the time required for a single epoch of retraining.

6.3.2.6 Impact of Permanent Faults in MACs and MUXes and the Effectiveness of the Proposed Systolic Array Designs

Fig. 6.14 shows the impact of permanent faults on the accuracy of the VGG11 trained on the Cifar10 dataset when executed using different types and sizes of systolic arrays (for designs see Fig. 6.4 in Section 6.2.1. For these experiments, it is assumed that permanent faults can occur in the multiplexers of the fault-tolerant array as well. To have a fair comparison across arrays, relative areas of MAC and multiplexer units are also taken into consideration for fault injection. By analyzing Fig. 6.14, following observations can be made:

- SalvageDNN outperforms the state-of-the-art FAP approach in all the cases.
- The DNN accuracy drops significantly for all the cases in which the systolic array is composed of C_PEs, even at lower fault rates, as shown in Figs. 6.14a and 6.14b. This is due to the fact that all the PEs that have at least one of their downstream PEs containing a permanent fault are disconnected. Therefore, the probability of a PE getting disconnected at a specific fault rate is significantly higher in the arrays composed of C_PEs.
- With an increase in the number of bypass connections in a PE for fault mitigation, the average DNN accuracy at a specific fault rate increases. For example, this can be observed by analyzing the DNN accuracy in Figs. 6.14a, 6.14c and 6.14e at 2% fault rate mark. The $DBNC\_PE$ almost maintains the baseline accuracy, whereas the accuracy offered by C_PE equals 10% (i.e., equivalent to that of a random selection).

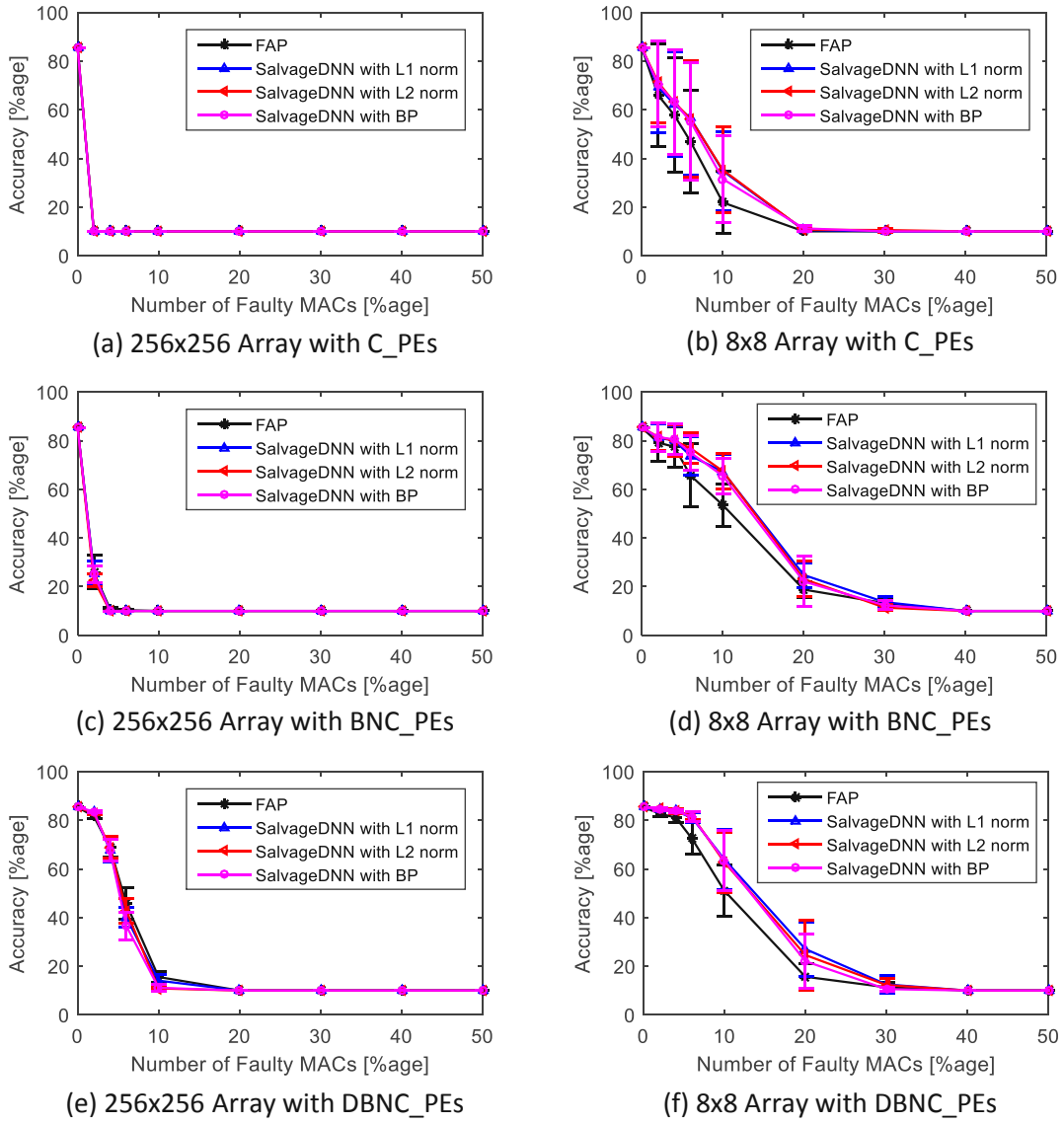


Figure 6.14: Impact of permanent faults in the proposed systolic array designs on the accuracy of the VGG-11 network trained on the Cifar-10 dataset. Here, BP corresponds to propagation-based method.

- The resultant DNN accuracy at any fault rate increases with the decrease in the size of the systolic array. This can be observed by comparing Figs. 6.14a, 6.14c and 6.14e with Figs. 6.14b, 6.14d and 6.14f, respectively. This is mainly due to the increased probability of disconnection of a MAC unit in the proposed systolic array designs with the increase in array size at any fault rate greater than zero.

In short, designs with more bypass connections are better for mitigating the effects of

permanent faults in the multiplexer units; however, they result in high area, power/energy, and delay overheads, which increases exponentially with the number of bypass connections.

6.4 Chapter Summary

Alongside the need for developing highly energy and performance efficient DNN accelerators, there is a need to improve the yield of the fabrication process as well to reduce the per unit cost of DNN accelerators. Towards this, this work presented *SalvageDNN*, a methodology to enable reliable execution of DNNs on faulty hardware accelerators. At the core, it is a fault-aware mapping technique that leverages the saliency of DNN parameters along with the concept of fault-aware pruning for mapping less significant weights onto faulty components to achieve fault mitigation at a lower cost and without any run-time overheads. The proposed technique uses rearrangement of DNN parameters at the software-level to avoid any additional data rearrangement operations at run-time. The work analyzed novel hardware modifications for mitigating faults in the *bypass* multiplexer units as well. The analysis shows that faults in multiplexers, in general, increase the number of disconnected PEs and, thereby, impact the effectiveness of the proposed as well as the state-of-the-art techniques. Comparison with the state-of-the-art techniques shows that *SalvageDNN* offers better accuracy results. Moreover, the results show that *SalvageDNN* can complement fault-aware retraining-based approaches as well.

CHAPTER 7

Aging Mitigation for Improving the Lifetime of On-Chip Weight Memories in Deep Neural Network Accelerators

Aging is one of the foremost reliability concerns in nanoscale devices. It results in gradual degradation of the fabricated hardware over time due to different phenomena such as Bias Temperature Instability (BTI) and Hot Career Injection (HCI). Studies have shown that Negative BTI (NBTI)-induced aging is the most dominant type of aging in CMOS devices fabricated using smaller technology nodes, specifically below 65 nm technology node. Given that DNN accelerators employ large on-chip SRAMs (along with other optimization techniques) for reducing the number of off-chip memory accesses and that DNNs are highly vulnerable to faults in sensitive locations, this work focuses on developing a low-cost technique for mitigating aging in on-chip SRAM cells.

This chapter presents a novel aging mitigation framework for mitigating NBTI-induced aging in on-chip weight memories of DNN accelerators. First, Section 7.1 presents the motivation behind this work in detail. Then, Section 7.2 presents an overview of the proposed framework. The framework comprises two main blocks: (1) analysis of the probability distribution of weight-bits under different settings; and (2) design of an effective aging mitigation scheme based on the insights gained from the analysis. Section 7.3 presents an analysis that highlights the impact of using different data representation formats and quantization methods for the weights of a DNN on the distribution of the weight-bits. The analysis provides useful insights for designing an effective aging mitigation scheme. The scheme and the supporting micro-architecture designed for mitigating NBTI-aging in on-chip weight memory of DNN accelerators

composed of 6T-SRAM cells is presented in Section 7.4.

7.1 Motivation and Problem Identification

DNNs usually require a large number of parameters to offer high accuracy, which comes at the cost of high memory requirements; see Fig. 7.1a. Dedicated memory hierarchies are designed to trade-off between the low-cost storage offered by the off-chip DRAMs and the energy-/performance-efficient access offered by the on-chip SRAMs [SCYE17]; see Fig. 7.1b for access energy statistics. The benefits of SRAMs and the inclination of the deep learning community towards designing larger and complex models to achieve ultra-high accuracy have led to an increasing trend towards using larger on-chip memories in DNN accelerators [CLL+14][JYP+17], with the recent wafer-scale chips having up to 18 GBs of on-chip memory [McL19]. However, due to continuous technology scaling, SRAMs are becoming increasingly vulnerable to different reliability threats such as soft errors and aging [HBD+13][SNT+20][HEAK13]. Studies have shown that even a single fault in a critical neuron can significantly degrade the application-level accuracy of a DNN-based system [HKP+18]. Earlier works have mainly focused on analyzing and mitigating the effects of faults in DNN accelerators through fault-aware retraining [KHM+18a]. *However, no prior work has analyzed and optimized the aging of the on-chip weight memories of DNN accelerators, especially considering diverse dataflows of different DNNs and the impact of different types of quantizations on the bit-level distribution of weights.*

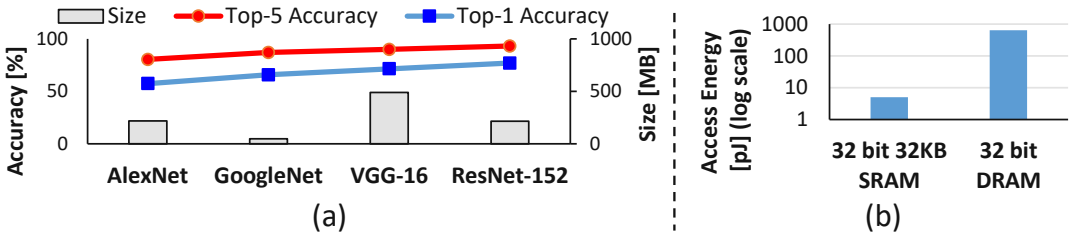


Figure 7.1: (a) Accuracy and size comparison of few of the state-of-the-art DNNs (b) Access energy comparison of SRAM with DRAM (data source: [SCYE17]).

Aging due to NBTI in CMOS devices: In PMOS transistors, when a negative gate-to-source voltage is applied, it can break down the Si-H bond at the oxide-interface and result in a gradual increase in the threshold voltage (V_{th}) over the device lifetime. This increase in the threshold voltage results in poor drive current and a reduction in the noise margin [KGPK08][1]. To overcome this V_{th} shift, the operating frequency of the device has to be reduced by more than 20% over its entire lifetime [GSK+15]. However, due to strict performance and energy constraints (specifically for embedded applications),

[1]A similar phenomenon called PBTI happens in NMOS transistors, though NBTI has been considered relatively more serious compared to PBTI [HBD+13].

the V_{th} shift cannot be addressed mainly through design-time delay margins or adaptive operating frequency adjustments [SZBP08], as they lead to a significant reduction in performance and energy efficiency. Therefore, alternate opportunities have to be explored to overcome this challenge [GSK+15]. One such opportunity lies in the fact that the NBTI aging phenomenon is partially reversed by removing the stress from the transistors.

NBTI Aging in On-chip Memories: On-chip memories are typically built using 6T-SRAM cells to achieve high density and energy efficiency. A 6T-cell is composed of two inverters coupled with two access transistors, as shown in Fig. 7.2a. The inverters store complementary values to store a bit. Each inverter has a PMOS transistor and an NMOS transistor. Depending on whether the cell is storing '0' or '1', one of the PMOS transistors is always under stress, when the transistor is on. As aging of a cell is defined by its most-aged transistor, the lowest aging is achieved when both the PMOS transistors have aged equally, i.e., they have received on-average the same amount of stress over their lifetime. In other words, this means that a cell experiences minimum NBTI-induced aging when the percentage of the entire lifetime for which the cell stores a '1' (*duty-cycle*) is 50% (see Fig. 7.2b). Note that NBTI aging strongly depends on average long-term stress and weakly on short-term statistics [AVG07]. *Therefore, the key challenge in aging mitigation of on-chip SRAMs is to balance the long-term stress (i.e., duty-cycle over the entire lifetime) on PMOS transistors without affecting system-level performance.*

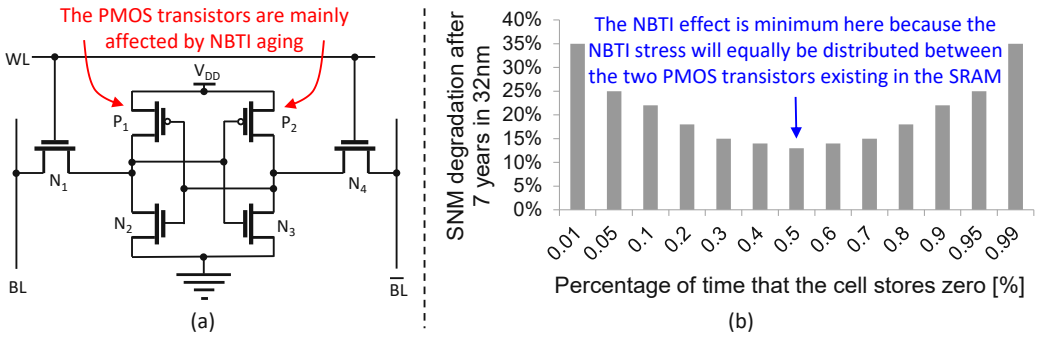


Figure 7.2: (a) A 6T-SRAM Cell; and (b) its SNM degradation after 7 years [KCR11]

Limitations of Earlier Works: Various techniques have been proposed at the circuit level and at the architecture level for mitigating aging in on-chip SRAM cells. At the circuit level, structural modifications have been proposed to reduce the aging rate [RSR+10][SZBP08]. For example, Ricketts et al. [RSR+10] proposed an asymmetric SRAM structure for workloads having biased bit distribution. However, due to high data dependence, this technique is applicable only in specific scenarios. Recovery boosting through dedicated recovery accelerating circuit is another method for enhancing the lifetime of SRAM cells [SG11]. However, it increases power/energy consumption due to the requirement of additional transistors per cell [ZSBH11]. At the architecture level, periodic inversion of data is used to reduce aging in on-chip SRAM-based caches [JW12].

However, periodic inversion of data cannot guarantee optimal duty-cycle, specifically in cases where the same data is reused periodically, e.g., in DNN-based systems. Calimera et al. in [CLMP11] improved recovery of un-utilized portions of memory. However, they achieve this at high energy & area cost incurred due to online monitoring support. Their technique also suffers from severe performance degradation in dynamic conditions. Another set of techniques employs circular shifts to cater NBTI aging in registers [KCR11]. However, such techniques are effective only in cases where the overall distribution of bits is relatively balanced. Moreover, these techniques employ barrel shifters that incur high area and power overheads. The work in [SKTH15] proposed a configurable micro-architecture for reducing aging rate of video memories, but it only works for streaming video applications.

In summary, the state-of-the-art techniques either incur high area and power overheads or are designed for specific workloads. All the above-mentioned techniques cannot be employed for DNN accelerators because of the unique properties of DNN workloads, which will be highlighted in the following sections.

Additional Challenges from the DNN Inference Perspective: The dataflow (i.e., computation scheduling) for processing data using a given DNN is defined based on the DNN architecture and the hardware architecture to achieve maximum energy/performance efficiency. Modifying the dataflow to balance the duty-cycle in on-chip SRAM cells can significantly degrade the system-level efficiency. *Therefore, an aging mitigation technique that does not require any alteration to the dataflow or the data mapping in on-chip SRAM is desired.*

7.2 Overview of DNN-Life Framework

This work presents DNN-Life, a novel aging analysis and mitigation framework for on-chip 6T-SRAM-based weight memories of DNN accelerators. To mitigate aging, it mainly exploits the fact that the NBTI aging can be minimized by balancing the duty-cycle in 6T-SRAM cells. DNN-Life achieves this through a low-cost data encoding scheme that accounts for diverse DNN workloads. The two main blocks of the framework are highlighted in Fig. 7.3 and are explained as follows:

1. **Analysis:** The first block studies the probability of occurrence of '1' at each bit location in a weight-word to find critical insights that can help develop a low-cost aging-mitigation scheme. The analysis is performed for multiple different pre-trained DNNs, number representation formats, and quantization methods to consider all the possible variations. The detailed analysis and insights are presented in Section 7.3.

2. **Architecture:** Based on the insights gathered from the analysis, a data encoder and an aging controller are designed. The encoder is responsible for encoding the weights before writing them to the on-chip weight memory. The aging controller is responsible for generating control signals for the encoder module such that the duty-cycle in each

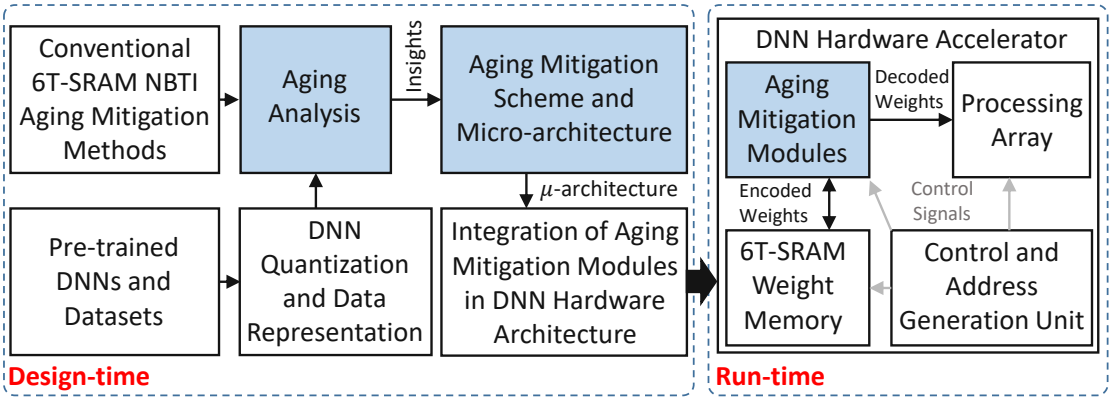


Figure 7.3: Overview of the design-time steps involved in the proposed DNN-Life framework. The right side shows a high-level view of how the proposed aging mitigation module is connected with rest of the modules in a DNN accelerator. The novel contributions of this work are represented using colored boxes.

SRAM cell is balanced. The control data is stored to be used later by the decoder module to transform the data back before sending it to the computational modules. The encoder module is deployed inside the accelerator right before the weight memory, and the decoder is placed right after the memory, as illustrated in Fig. 7.4a. The details of the micro-architecture are presented in Section 7.4.

7.2.1 DNN Hardware Architecture

The baseline DNN hardware architecture used in this work is based on a well-established DNN accelerator model, i.e., Dense CNN [DJS+18]. The accelerator is composed of an *Activation Buffer*, a *Weight Buffer*, a *Processing Array*, and an *Accumulation Unit*; see Fig. 7.4a. The proposed weight-memory aging-mitigation modules are also shown in the figure, integrated with the other modules (for details of the aging-mitigation modules, see Section 7.4). The activation and weight buffers act as intermediate storage for the activations and weights, respectively, to reduce the number of costly off-chip memory accesses. The buffers feed data to the processing array for computations. For this work, a memory hierarchy similar to Bit-Tactical [DJS+18], DaDianNao [CLL+14] and TPU [JYP+17] is considered, according to which: 1) the activation buffer is large enough to store all the activations of a single layer of a DNN; 2) the activation memory can feed N number of activation values to the processing array at a time; and 3) the weight memory can provide $f \times N$ weights to the processing array at a time. The processing array (see Fig. 7.4b) is composed of f number of *Processing Elements* (PEs) that share the activations, and therefore, the array can perform N number of multiplications for f different filters at the same time. Each PE has an adder tree to compute the sum of the products. The computed sum is passed to the accumulation unit where it is added to the corresponding partial sums to generate the output activation value. Note, as the filters

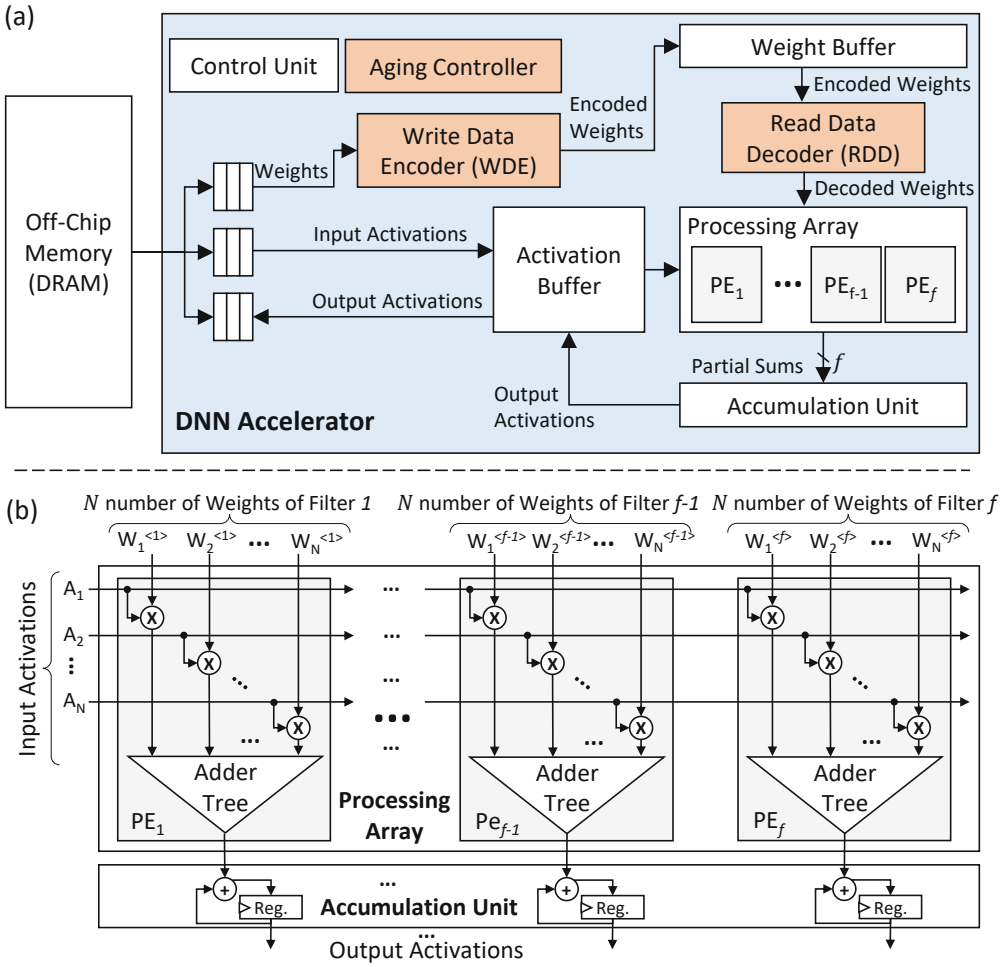


Figure 7.4: (a) Architecture of the baseline DNN accelerator. The highlighted boxes, i.e., Write Data Encoder (WDE), Read Data Decoder (RDD) and Aging Controller, are the proposed modules for mitigating NBTI aging of weight memory. (b) A detailed view of the processing array and the accumulation unit.

can be significantly large, the computation of each output activation can take several cycles, depending on the filter size.

7.2.2 Dataflow in the DNN Accelerator

To perform the computations of a DNN layer using the accelerator described in Sec. 7.2.1, the 4D weight tensors have to be partitioned into smaller blocks (so-called *tiles*) that can be accommodated in the on-chip weight memory. The main goal of the partitioning is to maximize the use of available PEs. The input/output feature maps and the filters/neurons all are divided into *tiles*, depending on the available on-chip storage for the corresponding data type. Works like SmartShuttle [LYL+18] have proposed methods to find effective

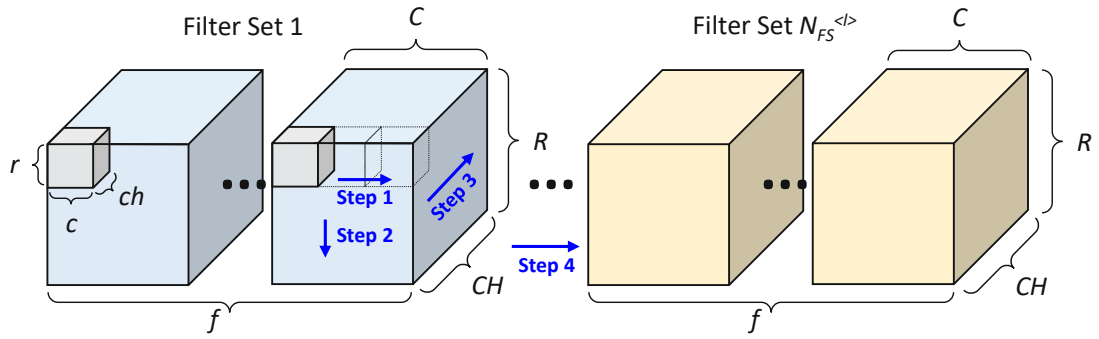


Figure 7.5: Division of filters of a CONV layer of a DNN into smaller blocks that can be accommodated in the on-chip weight memory. Different colors correspond to different sets of filters/blocks. The gray colored boxes define one block of $r \times c \times ch \times f$ size. The *steps* show the sequence in which the blocks are moved to the on-chip fabric for scheduling their computations.

tiling configurations and computation scheduling policy that optimize the number of off-chip memory accesses a given DNN and memory hierarchy.

Fig. 7.5 illustrates the policy employed for partitioning the weight tensors of a CONV layer. Note that the well-established tiling technique is employed to demonstrate that the proposed technique is beneficial for a wide-range of existing DNN hardware accelerators. The figure also illustrates the sequence in which the blocks are fetched and moved to the on-chip weight memory for computations. The filters are first divided into sets of f number of filters. Afterwards, a chunk of data (grey boxes in Fig. 7.5) is selected from a set to be moved onto the on-chip weight memory. The selected chunk contains a block of data of size $r \times c \times ch$ from the same location of each filter in the set. The sequence in which the grey boxes are traversed defines rest of the dataflow. The sequence followed for this work is illustrated with the help of steps in Fig. 7.5. Note that f is defined based on the size of the processing array, i.e., the number of filters it can process in parallel, r, c and ch values are defined based on the size of the PEs. Moreover, if the on-chip memory is large and can accommodate multiple sets of blocks, then multiple sets are moved to the memory.

7.3 Analysis of the Probability Distribution of Weight-Bits of DNNs & the Impact on Duty-Cycle

This section presents an analysis that highlights the rationale behind the proposed architectural modifications and hardware design.

7.3.1 Analysis of the Probability Distribution of Weight-Bits

For this analysis, two different DNNs the AlexNet and the VGG16 networks, both trained with the ImageNet dataset, are considered. To study the variations across different data

representations for weights, 32-bit floating point representation (IEEE 754 standard) and 8-bit integer format achieved using range-linear *symmetric* and *asymmetric* quantization techniques [LTA16] are considered. Fig. 7.6 illustrates the probability of observing a '1' at each bit-location of a word for all considered data representation formats and DNNs. By analyzing the probability distributions, the following observations can be made:

1. *The probability of observing a '1' at a particular bit-location of a randomly selected weight depends on the DNN, the data representation format, and the method used to transform the data to the particular format.* For example, the probability of observing a '1' at a particular bit-location in a randomly selected weight represented in a symmetric 8-bit representation format is almost the same across all the bit-locations within a DNN for both the considered DNNs, however, it varies across DNNs. Similarly, the probability of observing a '1' at lower bit-locations in 32-bit floating-point representation format is approximately 0.5; however, the distribution of bits at higher bit-locations varies significantly across bit-locations and DNNs.

2. *A particular data representation format does not guarantee a probability distribution that offers 0.5 probability at all the bit-locations, i.e., a distribution that can potentially lead to a balanced duty-cycle.* For example, out of all the studied cases, only the probability distribution of the AlexNet for 8-bit integer symmetric quantization offers close to 0.5 probability for all the bit-locations.

3. *The average probability of observing a '1' across bit-locations in a particular data representation format is also not guaranteed to be equal to 0.5.* For example, see the distributions of 8-bit asymmetrically quantized DNNs. Even for symmetric quantization, specifically for the VGG-16 case, the average across bit-locations is 0.4. Therefore, barrel shifter-based balancing techniques are also ineffective in such scenarios.

7.3.2 A Probabilistic Model-based Analysis for Aging of 6T-SRAM On-chip Weight Memory of a DNN Accelerator

The following subsections present an analysis based on a probabilistic model to analyze the effectiveness of different aging mitigation techniques.

7.3.2.1 Probabilistic Model

Assume the on-chip memory of a given DNN accelerator is a grid composed of $I \times J$ 6T-SRAM cells. For mapping the weights onto the memory, the following conditions are assumed: (1) the same dataflow as presented in Fig. 7.5 is considered for computations; (2) each block of weights is kept in the on-chip memory for a constant (CT) amount of time, and it is fetched only once per inference (similar to the dataflow in [DJS+18]); (3) each block of data has the same size as on-chip weight memory and fits perfectly on it, i.e., it covers all the cells. Based on the aforementioned conditions and the given DNN

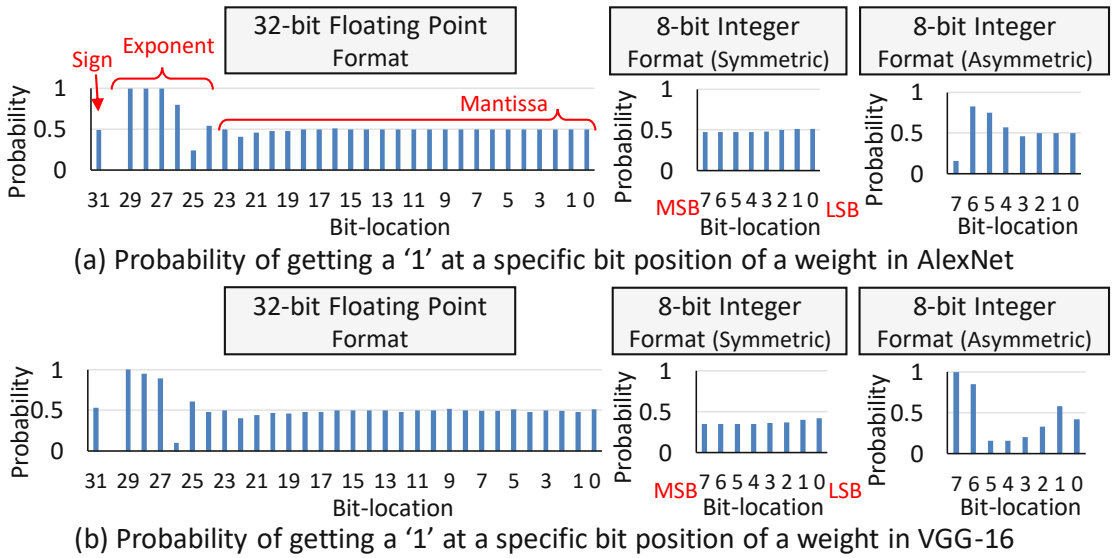


Figure 7.6: Distribution of bits of weights of different different DNNs when represented in different data representation formats. Symmetric and asymmetric represent which post-training quantization method is used to transform the data for the corresponding distribution.

size, the parameters of the DNN can be divided into K blocks, which translates to K number of data mappings onto the on-chip weight memory. Now, if the same DNN is used repeatedly for processing data, assuming the same dataflow, a single on-chip weight memory cell is mapped with only K different bits. Furthermore, if overall the probability of observing a '1' is given by ρ, the probability of getting a duty-cycle less than and equal to a ratio b/K, or greater than and equal to $1 - b/K$, can be computed using the following equation, except when $b/K = 0.5$ where the probability is 1.

$$P_{b/K} = \sum_{i=0}^{b} \binom{K}{i} \rho^i \times (1-\rho)^{K-i} + \sum_{i=K-b}^{K} \binom{K}{i} \rho^i \times (1-\rho)^{K-i} \tag{7.1}$$

Here, b is an arbitrary variable with the range from 0 to $floor(K/2)$. Note that we combine the cases in which duty-cycle $\leq b/K$ and duty-cycle $\geq 1 - b/K$, because in a symmetric 6T-SRAM cell both the types cause the same level of stress in one of the two PMOS transistors and, therefore, result in the same level of aging. Assuming the above computed probability to be the same for all the cells of the on-chip memory, i.e., considering uniform probability distribution, the probability of at least n number of cells (out of $I \times J$) experiencing duty-cycle $\leq b/K$ or $\geq 1 - b/K$ can be computed using the following equation.

$$P_n = \sum_{i=n}^{I \times J} \binom{I \times J}{i} P_b^i \times (1-P_b)^{I \times J - i} \tag{7.2}$$

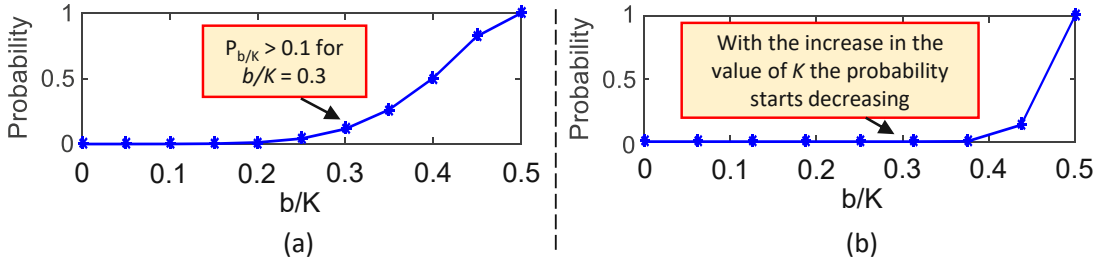


Figure 7.7: Probability of occurrence of $b/K \geq$ duty-cycle $\geq 1 - b/K$ when (a) $K = 20$, and (b) $K = 160$

7.3.2.2 An Example Case Study

Let us consider a scenario where $K = 20$ and $\rho = 0.5$ (i.e., the best-case scenario with balanced bit distribution), and $I \times J = 8192$. Fig. 7.7a shows the probability for each possible value of b computed using Eq. 7.1. Note that even for $b/K = 0.3$, the probability is over 0.1, i.e., more than 10% of the cells are expected to experience a duty-cycle ≤ 0.3, or ≥ 0.7. This highlights that even if the probability of occurrence of '1' at different bit locations is 0.5 (i.e., $\rho = 0.5$), it does not ensure a balanced duty-cycle in all the cells.

To study the case further, consider a bit-shifting-based aging mitigation technique that offers up to 7 shifts. Such a technique can increase the number of different bits mapped to a single cell. For the given example, it can theoretically increase the value of K to 160, assuming the bits to be independent of each other and the ideal shifting policy. Fig. 7.7b shows the updated probabilities for different b/K values when K is set to 160 in the above-mentioned example. By comparing Fig. 7.7b with Fig. 7.7a, it can be said that larger K results in lower probabilities at b/K other than 0.5. *This implies that by significantly increasing K, while having $\rho = 0.5$, close to ideal duty-cycle can be achieved for all the cells.*

Now, instead of a bit-shifting-based aging mitigation, if an inversion-based duty-cycle balancing technique is employed, where every other write to the same location is inverted, for the given scenario, the value of K remains the same, as it is even. Moreover, as ρ is defined to be 0.5, the inversion-based policy has no impact on ρ either. Therefore, the same probabilities are observed, as presented in Fig. 7.7a. However, note that *the inversion-based policy is highly useful for achieving $\rho = 0.5$ in cases where the distribution of bits is initially biased towards either '0' or '1'.*

7.3.3 Challenges for Designing an Efficient Aging Mitigation System

Based on the above analysis, the following key challenges are identified for designing a *generic* scheme for mitigating aging in 6T-SRAM-based on-chip weight memory of DNN accelerators.

1. The probability of a cell experiencing a non-ideal duty-cycle is considerable even with the state-of-the-art aging mitigation techniques that follow a *fixed* policy. Therefore,

a more robust method is required that can ensure close to ideal duty-cycle in all the SRAM cells without much overhead cost.

2. The probability distribution of bits and the duty-cycle is significantly affected by the datatype used for representing the weights. *Therefore, the mitigation technique should be generic and independent of the datatype used, so that it is beneficial for various DNN accelerators.*

Moreover, in practical scenarios, each layer of a DNN can have a different size. Therefore, each layer can take different amount of time for processing, which can vary significantly across layers. Also, different DNNs can have different number of layers. A method that keeps track of all these factors at a finer granularity can significantly reduce the aging rates. However, such methods are *super costly. This makes it extremely challenging to develop a generic method that offers effective aging mitigation at reasonable overhead costs.*

7.4 A Micro-architecture for Mitigating Aging of the On-Chip Weight Memory of DNN Accelerators

To address the above challenges, this work mainly exploits the fact that NBTI-aging is more dependent on the average duty-cycle over the entire lifetime of the device and less dependent on short-term averages [AVG07]. To achieve balanced duty-cycle in all the SRAM cells, this work proposes a *Write Data Encoder* (WDE), which encodes the weights before writing them to the on-chip weight memory, and a *Read Data Decoder* (RDD), which performs the inverse function of the WDE while reading the data from the on-chip memory and before passing it to the processing array. Fig. 7.4a shows the integration of the proposed modules in the DNN accelerator presented in Fig. 7.4. This work also proposes an aging mitigation controller that generates the required control signals for the read and write transducers. The proposed micro-architectures of the WDE and the aging mitigation controller are presented in Fig. 7.8.

Write Data Encoder (WDE): It leverages the inversion logic that, besides its low-overhead compared to other techniques (as shown later in Section 7.5), helps achieve perfectly balanced probability distribution of bits in the cells of the memory when the distribution is originally biased towards either '0' or '1'. The inversion logic in the proposed micro-architecture is implemented using XOR gates, as they allow the aging mitigation controller to enable or disable inversion using just a *1-bit enable* (E) signal. *Another advantage of the proposed design is that RDD has the same micro-architecture as WDE*, and the same E signal (metadata) that is used to encode the weights is used (at a later point in time) for decoding them before passing them for computations. Moreover, the proposed WDE and RDD modules are highly *scalable*, as increasing the width of the modules require only a linear increase in the number of XOR gates. Therefore, the widths of these modules can be defined directly based on the configuration of the given DNN accelerator, without affecting the energy efficiency of the system.

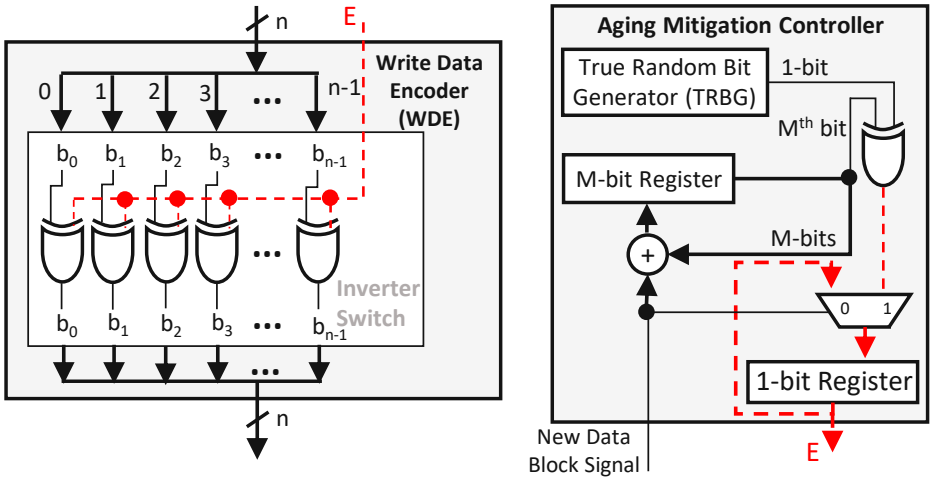


Figure 7.8: Proposed micro-architecture for effective aging mitigation of 6T-SRAM weight memory of DNN accelerators.

Aging Mitigation Controller: The controller is responsible for generating the *enable signal* (E) that controls the inversion logic in WDE. The design is based on the observations made in Section 7.3 that the higher the number of different bits to be written on an SRAM cell during its lifetime (i.e., K in Eq. 7.1) the lower the chances of observing a deviation from the ideal duty-cycle (see Fig. 7.7 and the ideal point highlighted in Fig. 7.2b). Therefore, to increase the number of different bits to be written on an SRAM cell, a True Random Bit Generator (TRBG) is employed to generate the enable signal. The TRBG module adds randomness to the data, which translates to extremely large K value, before it is written onto the 6T-SRAM cells.

Note, in practical scenarios, the output of TRBGs can be biased towards either '0' or '1', which can eventually affect the duty-cycle. Therefore, to mitigate this, the output of the TRBG is also periodically inverted using an M-bit register before forwarding it as the enable signal.

7.5 Results and Discussion

7.5.1 Experimental Setup

Fig. 7.9 shows the overall methodology followed for evaluating the proposed technique. The evaluation is divided into two parts: (1) overhead estimation; and (2) aging estimation. To quantify the overheads, the proposed aging mitigation modules are implemented in Verilog and synthesized for 65nm technology node using Cadence Genus with TSMC 65nm library and compared against different state-of-the-art aging balancing circuits synthesized using the same process.

To estimate aging, similar to [SKTH15][SKH16], Static Noise Margin (SNM) is used as a metric to quantify NBTI-induced aging in 6T-SRAM cells. The SNM defines

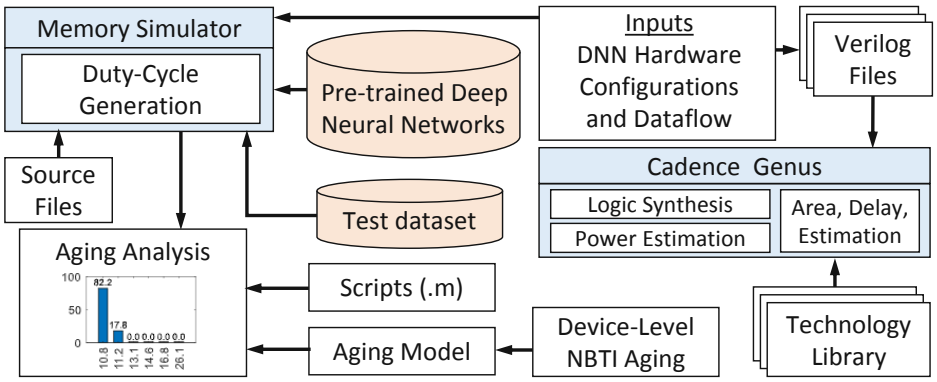


Figure 7.9: Overall experimental setup used for evaluation.

the noise tolerance that directly affects the read stability of a cell [AN06], i.e., if the SNM of a cell is low, the cell is highly susceptible to read failures. As highlighted in [KCR11][SKTH15][SKH16], SNM mainly depends on the duty-cycle over the entire lifetime of the device, and the least amount of SNM degradation is achieved at 50% duty-cycle mark, as shown in Fig. 7.2. To obtain SNM results, a device aging model similar to the ones used in works like [SKTH15][SKH16] is employed. Based on the models, the SNM degradation of a 6T-SRAM cell can be computed using the duty-cycle. From the analysis, it is observed that the best SNM degradation for 6T-SRAM cell after 7 years is 10.82% (at 50% duty-cycle), and the worst is 26.12% (at 0% and 100% duty-cycle).

For large-scale simulations, the output of the aging model is integrated into a memory simulator of the baseline DNN hardware (described in Section 7.2.1). The simulator takes the DNN hardware configuration, dataflow, pre-trained DNN architecture and test samples as inputs. To demonstrate the effectiveness of the proposed technique for multiple different hardware accelerators, a memory simulator for a TPU-like hardware architecture [JYP+17] is also implemented. The hardware configurations used for the evaluation are presented in Table 7.1. The DNNs used are AlexNet and VGG-16 with ImageNet dataset and a custom network with MNIST dataset. The details of the DNN architectures are presented in Table 7.2. For each setting the duty-cycles are estimated based on the values observed in 100 inferences. The bias balancing register is defined to be a 4-bit register (i.e., M=4), for all the corresponding cases.

Table 7.1: Hardware configurations and settings used in evaluation

	Baseline Accelerator (Section 7.2.1)	TPU-like NPU [JYP+17]
Weight memory size	512KB	256KB
Activation memory size	4MB	24MB
PE array size	8 PEs (1 PE = 8 Multipliers)	256 x 256 PEs (1 PE = 1 MAC)
Networks	AlexNet	AlexNet, VGG-16 and Custom

Table 7.2: DNNs and the datasets used for evaluation.

Network and Dataset	Network Architecture
AlexNet for ImageNet	Layers: CONV(96,3,11,11);CONV(256,48,5,5);CONV(384,256,3,3); CONV(384,192,3,3);CONV(256,192,3,3);FC(4096,9216); FC(4096,4096);FC(1000,4096)
VGG-16 for ImageNet	Layers: CONV(64,3,3,3);CONV(64,64,3,3);CONV(128,64,3,3); CONV(128,128,3,3);CONV(256,128,3,3);CONV(256,256,3,3); CONV(256,256,3,3);CONV(512,256,3,3);CONV(512,512,3,3); CONV(512,512,3,3);CONV(512,512,3,3);CONV(512,512,3,3); CONV(512,512,3,3);FC(4096,25088);FC(4096,4096); FC(1000,4096)
Custom for MNIST	Layers: CONV(16,1,5,5);CONV(50,16,5,5);FC(256,800); FC(10,256)

7.5.2 Aging Estimation Results and Comparisons

This subsection analyzes the impact of using different aging mitigation policies on the SNM degradation of the 6T-SRAM on-chip weight memory cells after 7 years. For this analysis, four different policies are mainly considered: (1) No aging mitigation, (2) Inversion-based, (3) Barrel shifter-based, and (4) DNN-Life. For the proposed DNN-Life, three different cases are considered: (i) TRBG is not biased and it generates 0s and 1s with equal probability (referred in the results as *Bias=0.5*); (ii) TRBG is biased and it generates 1s with 0.7 probability, and the aging controller does not have a bias balancing register (referred in the results as *without bias balancing with Bias=0.7*); and (iii) TRBG is biased and it generates 1s with 0.7 probability and the aging controller has a 4-bit bias balancing register (referred in the results as *with bias balancing with Bias=0.7*).

Moreover, three different data representation formats are considered for weights: (1) 32-bit floating point format; (2) 8-bit integer format when weights are quantized using symmetric quantization method; and (3) 8-bit integer format when weights are quantized using asymmetric quantization method.

Fig. 7.10 shows the distributions of SNM degradation in the memory cells obtained using different aging mitigation policies and the pre-trained AlexNet model. The Y-axis of each bar graph shows the percentage of the number of cells and the X-axis of each shows SNM degradation levels. Note that, for these experiments, the baseline DNN accelerator configuration presented in Table 7.1 and the dataflow shown in Fig. 7.5 with $f = 8$ are used. Also, it is assumed that only a single DNN (i.e., the AlexNet) is used for data inference throughout the lifetime of the device. As can be seen in the figure, the inversion-based and barrel shifter-based aging balancing reduce the SNM degradation of the SRAM cells, however, they do not offer minimum SNM degradation (see ❷ and ❸ in comparison with ❶ in Fig. 7.10). This behavior is observed to be consistent across all the data representation formats (see ❷ till ❼ in comparison with their respective *without aging mitigation* graphs in Fig. 7.10). Specifically, the inversion-based aging balancing offers sub-optimal aging mitigation in case of the 32-bit floating point format (see ❷

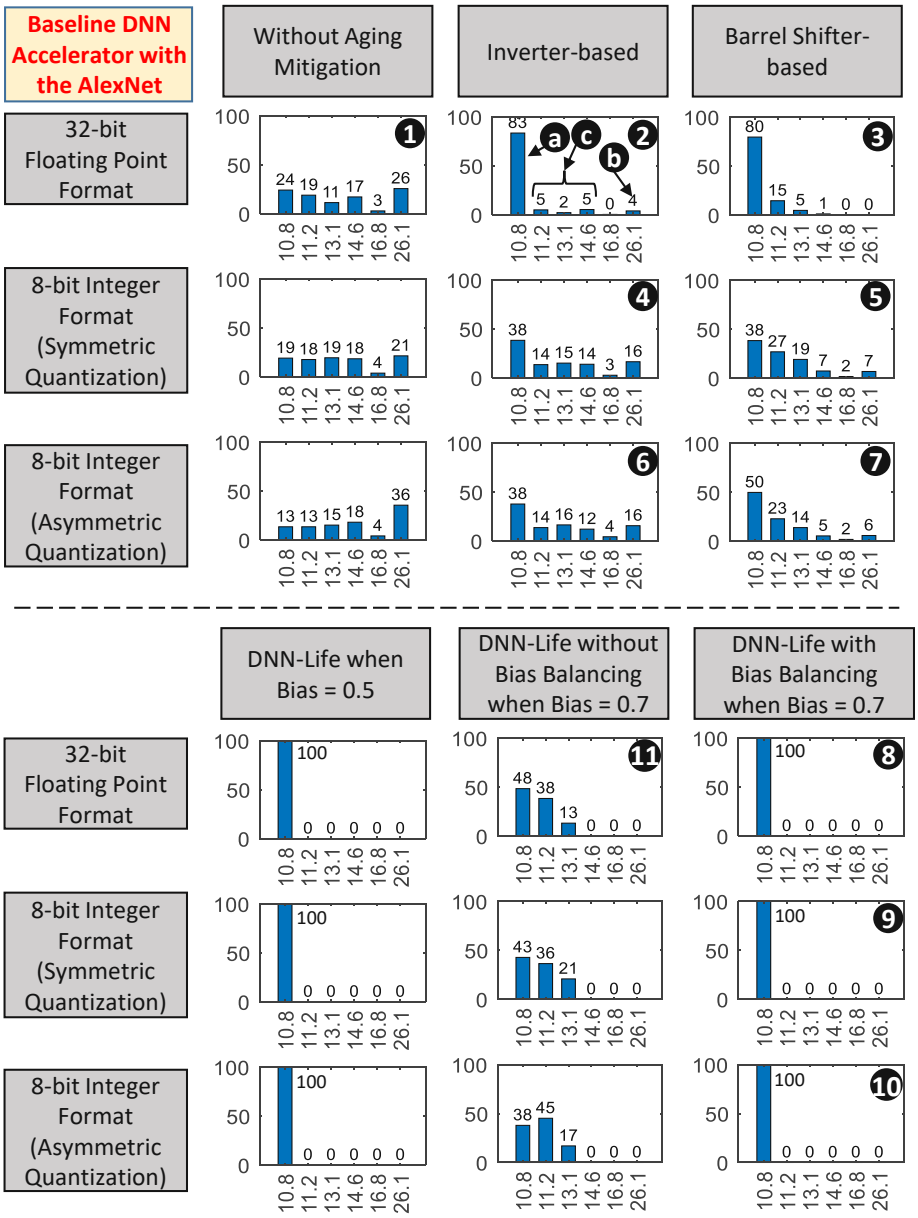


Figure 7.10: SNM degradation of 6T-SRAM on-chip weight memory cells of the baseline DNN accelerator when used for performing inferences only using the AlexNet network. Each bar graph shows the percentage of the number of cells (Y-axis) experiencing different level of SNM degradation (X-axis).

in Fig. 7.10), where most of the cells experience around 10.8% SNM degradation (see ⓐ in Fig. 7.10). However, this is not the ideal scenario as there are 4% cells that experience highest level of SNM degradation (see ⓑ in Fig. 7.10) and a few that experience moderate level of SNM degradation (see ⓒ in Fig. 7.10). Now, from the results of the proposed

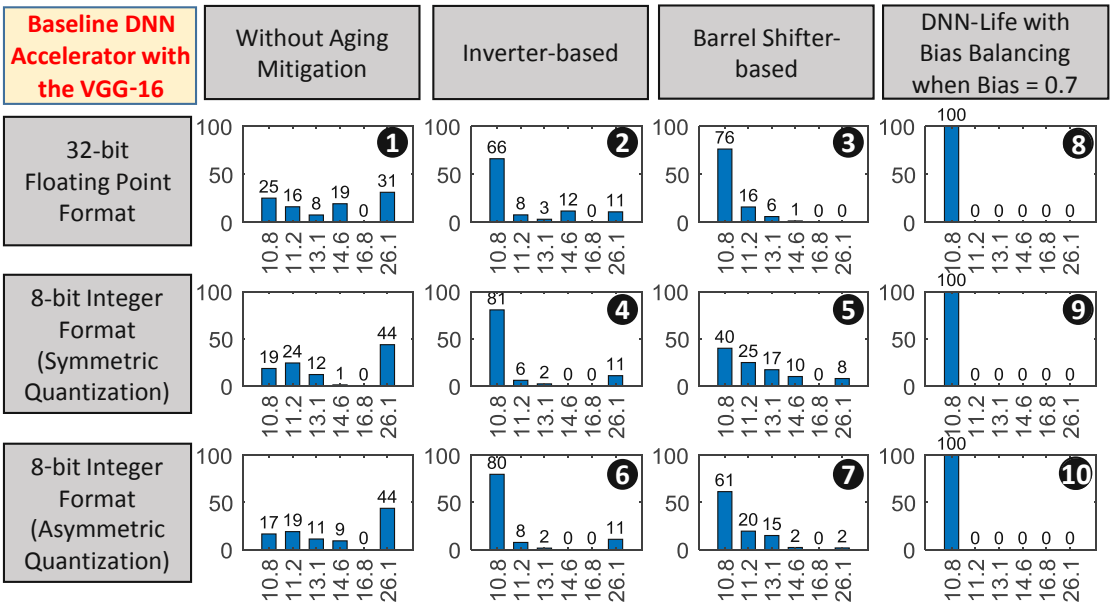


Figure 7.11: SNM degradation of 6T-SRAM on-chip weight memory cells of the baseline DNN accelerator when used for performing inferences only using the VGG-16 network. Each bar graph shows the percentage of the number of cells (Y-axis) experiencing different level of SNM degradation (X-axis).

DNN-Life with bias balancing, it can be observed that it offers maximum aging-mitigation (i.e., all the cells experience around 10.8% SNM degradation) in all the cases (see ❽, ❾ and ❿ in Fig. 7.10). Fig. 7.11 shows that the same pattern is observed for the VGG-16 network, i.e., comparing corresponding cases in ❶ till ❼ with ❽ till ❿, it can be said that *DNN-Life with bias balancing* offers optimal aging mitigation.

Impact of biased TRBG on aging balancing of 6T-SRAM on-chip weight memory: Fig. 7.10 also illustrates the impact of using proposed design without bias correction when the duty-cycle of TRBG is 0.7. As can be seen in the figure, for all the data representation formats, having biased TRBG and no bias correction leads to less reduction in SNM degradation of the 6T-SRAM cells (e.g., see ⓫ in comparison with ❽ in Fig. 7.10). This behavior is consistent across all the data representation formats.

Impact of on-chip memory size on aging balancing for 6T-SRAM on-chip weight memory: Fig. 7.12 shows the impact of on-chip weight memory size on the aging of on-chip 6T-SRAM cells. For this analysis, the baseline DNN accelerator with 8-bit symmetrically quantized VGG-16 is considered. The weight memory sizes used are mentioned at the top of the figure and the considered aging mitigation techniques are mentioned on the left side of the figure. As can be seen in the figure, the effectiveness of the state-of-the-art inversion-based and shifter-based techniques for large memory size is relatively less. For example, the inversion-based approach provides reasonable aging mitigation when the memory size is 512KB (see ❶ in comparison with ❷ in Fig. 7.12);

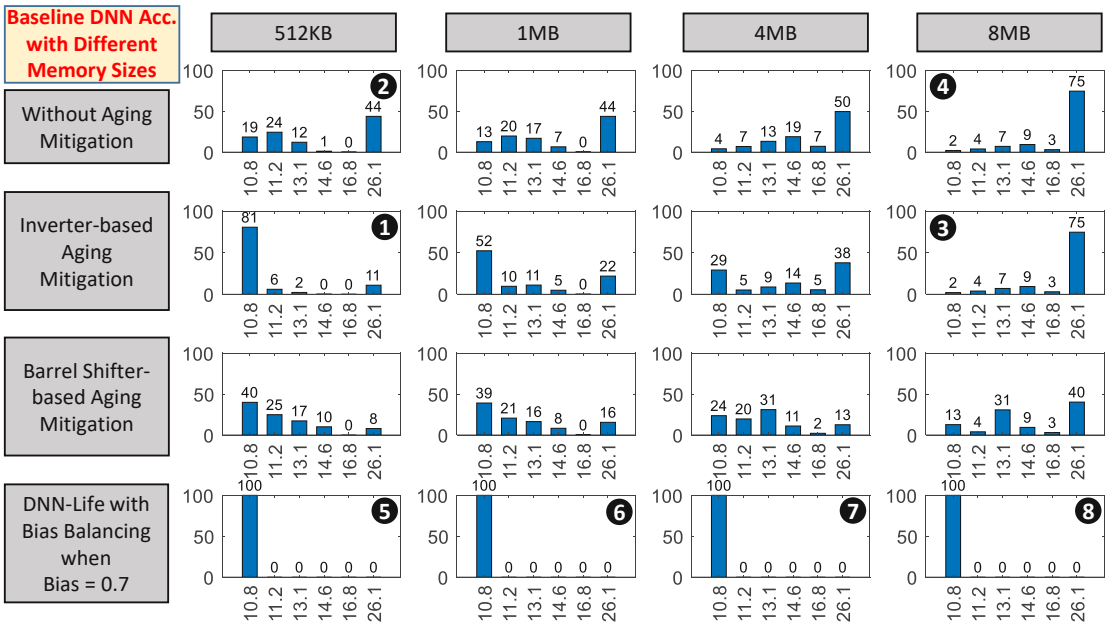


Figure 7.12: Impact of different on-chip weight memory sizes in the baseline DNN accelerator on the aging mitigation achieved using different techniques, when the inference is performed using 8-bit quantized VGG-16 (quantized using symmetric range-linear quantization method).

however, it does not offer good results when the size is 8MB (see ❸ in comparison with ❹ in Fig. 7.12). This is because of the fact that the number of distinct bits written onto the same bit location decreases with the increase in the on-chip memory size, as explained in Section 7.3. However, the proposed DNN-Life offers optimal aging-mitigation in all the cases (see ❺ till ❽ in Fig. 7.12).

Impact across different hardware accelerators: Fig. 7.13 shows the impact of using the proposed aging-mitigation technique for a TPU-like [JYP+17] Neural Processing Unit (NPU) architecture that has an on-chip weight FIFO which is four tiles deep, where one tile is equivalent to weights for 256×256 PEs. Each PE has a single MAC unit that can perform 8-bit multiplication. For the implementation, the weight FIFO is assumed to be a circular buffer-based design. The analysis is performed using the three different networks mentioned earlier. All the DNNs are quantized to 8-bits using post-training symmetric quantization. Considering the dataflow of the NPU, the parameter f was set to 256. As can be seen in Fig. 7.13, the inversion-based aging mitigation policy offers optimal results for the AlexNet and the VGG-16 networks (see ❶ and ❷ in Fig. 7.13). However, when used for the custom DNN, almost all the memory cells experience significant SNM degradation (see ❸ in Fig. 7.13). The barrel shifter-based approach also offer sub-optimal results (see ❹ till ❻ in Fig. 7.13). However, the proposed DNN-Life with bias balancing offers maximum aging mitigation (see ❼ till ❾ in Fig. 7.13). This shows that DNN-Life can be used for a wide range of DNN accelerators.

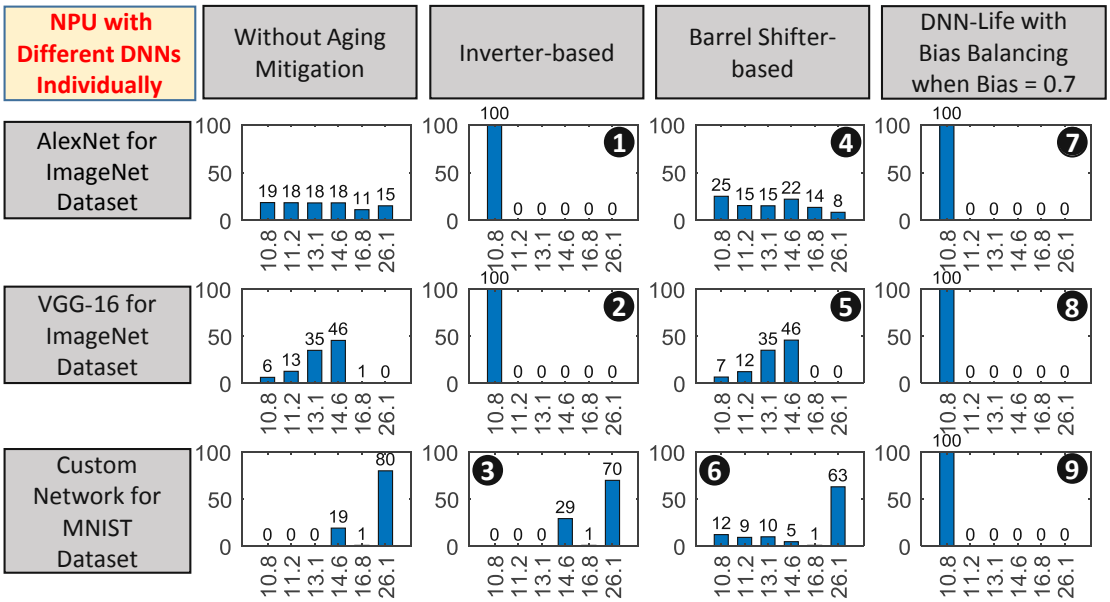


Figure 7.13: SNM degradation of 6T-SRAM on-chip weight memory cells of a TPU-like NPU when used for performing inferences using the AlexNet, the VGG-16 and the custom DNN, individually. The networks are quantized to 8-bit format using symmetric range-linear quantization method.

7.5.3 Area and Power Results

The area, power and delay characteristics of three different WDEs composed of different aging balancing units are shown in Table 7.3. All three WDEs are designed for a data bus width of 64-bits. It can be observed from the table that the barrel shifter-based WDE consumes the most amount of area and power. The proposed design consumes slightly more power and area compared to the inversion-based WDE. However, as shown in the previous subsection, it offers best aging-mitigation under all possible scenarios, regardless of the size of the given DNN, the distribution of weights, the data representation format and the on-chip weight memory size. Note that, at the hardware level, we realized TRBG using a 5-stage ring oscillator; however, it can be implemented using a pseudo-random bit sequence generator as well.

Table 7.3: Hardware results of different Write Data Encoders (WDEs)

	Delay [ps]	Power [nW]	Area [cell area]
Barrel Shifter based WDE	977.7	345190	9035
Inversion based WDE	811.6	10716	195
Proposed WDE with Aging Mitigation Controller	581.8	13747	295

7.6 Chapter Summary

To achieve high energy efficiency, DNN accelerators use large on-chip weight memories composed of 6T-SRAM cells. However, 6T-SRAM cells are highly vulnerable to Negative Biased Temperature Instability (NBTI)-induced aging, which can significantly reduce the lifetime of DNN accelerators. Towards this, this chapter presented DNN-Life, an aging-mitigation framework for reducing aging rates in the on-chip weight memories of DNN accelerators. In the framework, first, the effects of using different data representation formats on NBTI aging are studied. Then, based on the analysis, an aging-mitigation scheme is proposed that makes use of memory write and read transducers along with a True Random Bit Generator (TRBG) to reduce aging rates with minimal area and power overheads. The results clearly show that the proposed DNN-Life framework enables optimal aging mitigation in the on-chip weight memories of DNN accelerates at minimal run-time costs.

CHAPTER 8

Summary and Future Outlook

8.1 Thesis Summary

Deep Neural Networks have emerged as a promising set of techniques for solving complex AI problems. The state-of-the-art accuracy of these networks has enabled their use in safety-critical applications as well, e.g., autonomous driving, smart transportation, smart healthcare, smart grids, security & surveillance and robotics. The foremost challenge associated with deploying high-accuracy DNNs in real-world systems is that they are highly compute and memory-intensive, which leads to high inference costs. Moreover, specialized hardware accelerators are used to process these DNNs in an efficient manner; however, they bring in various reliability concerns (e.g., soft errors, device aging, and manufacturing defects) that can affect the functionality (or performance) of a DNN inference system and, thereby, are extremely critical specifically for safety-critical applications.

To reduce the compute and memory requirements of DNNs and offer energy-efficient DNN inference, various optimization techniques have been proposed (e.g., pruning, quantization, hardware approximations and data approximations). However, most of these techniques have been tested individually, which leads to limited efficiency gains. Besides that, not all the techniques are applicable in all scenarios, e.g., some techniques work better than others when coupled with retraining while others offer better trade-offs in cases where retraining is not possible (because of data IP and privacy reasons). Therefore, systematic application and scenario-specific cross-layer methodologies are required to significantly improve the energy and performance-efficiency of DNN inference systems. Besides that, to address reliability concerns, redundancy-based fault-mitigation techniques (e.g., dual-/triple-modular redundancy, instruction duplication, and error-correcting codes) are conventionally used, which lead to high overheads specifically when coupled with the compute and memory-intensive nature of state-of-the-art DNNs. Therefore, alternative approaches are required that can exploit the intrinsic error-resilience characteristics of DNNs to offer improved reliability at low overhead costs.

Towards addressing the above challenges, *this thesis aimed at exploiting the unique error-resilience characteristics of DNNs to improve the energy efficiency and robustness of DNN inference systems.*

For improving the efficiency, this work explored judicious approximations (i.e., carefully crafted designer-induced faults in less-sensitive neurons) that can be tolerated due to error-resilience characteristics of DNNs while offering significant efficiency gains. Opportunities both at the software level and the hardware level have been explored. For hardware-level approximations, first, different error estimation and design space exploration methods for approximate adders have been proposed and studied in Chapter 3. It presented *QuAd*, a configurable adder model that covers the entire design space of low-latency approximate adders. The analysis of *QuAd* configurations showed that, given a latency constraint, the optimal low-latency approximate adder configuration ($QuAd_o$) can be selected effortlessly. Then, to efficiently explore the design space of low-power approximate adders, PEMACx, a methodology for efficient error estimation of low-power approximate adders composed of smaller cascaded adder units, has been proposed. Afterwards, an analysis is presented which shows that data distribution of inputs can have a significant impact on the output quality of an approximate adder. Therefore, in cases where input distribution is not uniform or input bits cannot be assumed independent, a data-driven approach is necessary to accurately estimate the quality of an approximate module. Towards this, DAEM, a data and application-driven error estimation methodology, is proposed. The evaluation showed that DAEM is suitable for shallow datapaths and cannot produce highly accurate results for deeper datapaths, which pointed towards the significance of simulation-based evaluation and also the need for specialized approximation/optimization techniques that can effectively trade application-level accuracy for efficiency without involving intermediate error estimation.

Several specialized optimization techniques have been proposed for DNNs at different abstraction layers of the computing stack. To enable the use of DNNs in highly resource-constrained devices, in this work (in Chapter 4), a novel cross-layer methodology is proposed. At the software level, the methodology employs structured pruning along with quantization of activations and network parameters to reduce the computational complexity as well as memory footprint of DNNs. At the hardware level, it deploys functional approximations in the arithmetic modules of DNN accelerators to further boost the efficiency by exploiting the error-resilience of DNNs. The results highlighted that software-level optimizations offer far higher efficiency gains compared to hardware-level approximations; however, hardware approximation does contribute towards pushing the efficiency gains further. The results also highlighted that functional approximations of the arithmetic modules in DNN accelerators can lead to undesirable accuracy degradation. *This is mainly because the error introduced in the system due to approximations also depends on the data distribution of inputs, which highlights the need for data-driven approximations.*

Another key contribution of this thesis is a novel non-linear DNN quantization approach presented in Chapter 5. The technique exploits the data distribution characteristics of

DNNs, mainly the fact that DNN data structures have long tails where most of the values are concentrated near zero. In general, to achieve lower overall quantization error, values near zero should be quantized using higher resolution while the values far away from zero can be quantized using lower resolution. To significantly benefit from this concept, this work proposed a novel data representation format, i.e., Efficient Low-Precision Binary Signed Digit (ELP_BSD) format, which is (on a broader scale) based on concept of power-of-two quantization. The proposed data representation format not only reduces the complexity of the MAC units used in neural arrays but also helps in reducing the bitwidth of DNN parameters, which significantly reduces the DNN memory footprint and inference cost. A correlation-based error compensation strategy is also presented to reduce the impact of quantization errors introduced due to the non-linear quantization of DNN parameters. The proposed CoNLoCNN methodology then combines the novel non-linear quantization, ELP_BSD data representation format and the error compensation strategy to offer about 75% PDP reduction at the cost of 1.44% accuracy loss for the AlexNet network without involving retraining. Further, to enable functional approximation of hardware modules without any application-level accuracy loss, this work proposed the concept of curable approximations. The neural array designed based on the concept showed over 30% PDP reduction while offering the same DNN accuracy as an accurate (non-approximate) array design. Although the concept of curable approximations has been tested only for DNNs in this work, its applicability is not limited to deep learning domain.

Towards improving the resilience of DNN systems against hardware-induced reliability threats, this thesis focused on developing low-cost techniques for addressing permanent faults (manufacturing defects) in neural arrays and NBTI-induced aging in the on-chip weight memory of DNN accelerators. For addressing permanent faults, this work presented *SalvageDNN* in Chapter 6, a methodology to enable reliable execution of DNNs on faulty hardware accelerators. At the core, it is a fault-aware mapping technique that leverages the saliency of DNN parameters along with the concept of fault-aware pruning for mapping less significant weights onto faulty components to achieve low-cost fault mitigation. The proposed technique uses rearrangement of DNN parameters at the software-level to avoid any run-time overheads. Comparison with state-of-the-art techniques showed that *SalvageDNN* can offer comparable results at low and moderate fault rates without incurring any retraining or run-time overheads.

For addressing NBTI aging in the on-chip weight memory of DNN accelerators, this work presented *DNN-Life* in Chapter 7, an aging-mitigation framework for reducing NBTI-aging rates in the on-chip weight memory cells of DNN accelerators. In the framework, first, the effects of using different data representation formats on NBTI aging are studied. Then, based on the analysis, an aging-mitigation scheme is proposed that makes use of memory read and write transducers along with a random bit generator to reduce the aging rates without incurring high area and power overheads. The results showed that the proposed DNN-Life framework enables optimal aging mitigation in the on-chip weight memories of DNN accelerates at negligible overhead cost.

In summary, this thesis proposed various techniques that exploit the intrinsic characteristics of DNNs to significantly improve the energy efficiency and robustness of DNN inference systems.

8.2 Future Work

The field of deep learning is progressing at a rapid pace and new architectures are being proposed and published on a daily basis. Moreover, the spectrum of deep learning applications is also increasing at an accelerated pace. Towards this, there are several opportunities to build upon the techniques proposed in this thesis. In the short term, the following are some potential directions that can be explored:

1. **Tuning the proposed techniques for newly published architectures:** In pursuit of high accuracy, new DNN architectures are being explored and reported on daily basis. For example, recently, google proposed vision transformers [DBK+20][DLLT21] to achieve better accuracy on image classification tasks. Most of the techniques proposed in this thesis, as well as the ones reported in the literature, have been mainly tested for conventional CNNs such as AlexNet, VGG, MobileNet, ResNet and DenseNet models. Therefore, a possible direction is to explore the resilience of newly proposed DNN architectures to different types of errors and fine-tune the existing fault-mitigation, optimization and approximation techniques for such architectures.

2. **Low-cost techniques for mitigating soft error:** Soft errors are non-deterministic in nature and can occur in any component of the hardware. As these errors can significantly degrade the accuracy of DNNs, mitigating these is essential, specifically for safety-critical applications. Therefore, sophisticated low-cost soft error detection and mitigation techniques are required to be explored to effectively detect and address soft errors in DNN inference systems without incurring high overheads. Works like [Cea20] have proposed range-restriction methods to prevent high-intensity errors from propagating towards the output. These studies have mainly been performed for MLPs and conventional CNNs and have not considered recurrent networks or auto-encoders. Therefore, a possible future direction could be to study the effectiveness of range-restriction methods for advanced recurrent and transformer networks and employ neuron-level redundancy together with range restriction to develop more robust methods to mitigate soft errors in advanced high-accuracy DNN inference systems.

3. **Improving fault-aware (re)training method:** Fault-Aware Training (FAT) methods, such as [ZGBG18], have emerged to be the best for addressing permanent faults in DNN accelerators. However, due to their high computational requirements for tuning a given DNN for each faulty chip using its unique fault map, they can incur huge training overheads depending on the number and types of faults in the

hardware and the complexity of the DNN and the application. Towards this, a possible direction could be to design frameworks that can reduce the overheads of FAT techniques while still benefiting from their advantages. We have published a proof-of-concept of this in [HS23].

In the long term, further technological advancements such as Processing In-Memory (PIM) and Wafer-Scale Integration (WSI) technologies are likely to open new avenues for building highly efficient deep learning systems.

1. **PIM**, specifically where the computations are performed directly using memory cells, is an appealing solution to reduce the data movement required during DNN execution. However, as in such systems (e.g., ReRAM devices) the dot-product operations are carried out in the analog domain, they suffer from various practical issues such as wire IR drop and limited precision, which make it challenging to build reliable DNN accelerators. Therefore, improvements are required at the software level as well as the hardware level to enable the true benefits of PIM for the deep learning domain.

2. **WSI** is being exploited to build high-performance Wafer-Scale Engines (WSEs) for large-scale DNNs, as it enables to integrate unprecedented amounts of on-chip memory and computational units. However, the large size of WSEs implies higher number of manufacturing defects, which can translate to significant yield loss. Therefore, low-cost yield enhancement techniques are required for such systems to increase the manufacturing yield without affecting the performance of the chips or the DNNs. Apart from the increased likelihood of manufacturing defects, WSEs are expected to experience higher number of soft and timing errors as well. Therefore, methods for systematic integration of different low-cost fault-mitigation techniques are required that can analyze the interactions between different techniques and tune the available knobs in a way that leads to the best performance-reliability trade-offs.

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

Abbreviation	Description	Page
AAU	Approximate Adder Unit	69
ABFT	Algorithm-Based Fault Tolerance	35
AC	Approximate Computing	26
ACA	Accuracy-Configurable Adder	71
AI	Artificial Intelligence	1
ASICs	Application-Specific Integrated Circuit	51
AUs	Adder Units	58
BIST	Built-In-Self-Test	32
BTI	Bias Temperature Instability	28
BW	Bit-Width	126
CA	Canonical Approximate	128
CBW_A	Critical Bit-Width for Activations	125
C&DAx	Cure & Deterministic Approximate	106
C_{in}	Carry-in	58
CMOS	Complementary Metal-Oxide-Semiconductor	33
CNNs	Convolutional Neural Networks	15
CONV	Convolutional	16
C_{out}	Carry-out	58
CPS	Cyber–Physical Systems	1
Cu	Cure	106
DAx	Deterministic Approximate	106
DM	Disconnection Map	138
DMR	Dual Modular Redundancy	30
DNNs	Deep Neural Networks	1
DRAM	Dynamic Random Access Memory	17
ECC	Error Correction Codes	29
EIE	Efficient Inference Engine	22
ELP_BSD	Encoded Low-Precision Binary Signed Digit	122
EM	Electromigration	28
E_{MAP}	Error Map	71
ESA	Equally Segmented Adders	71
FA	Full Adder	58
FAP	Fault-Aware Pruning	32
FAP+T	Fault-Aware Pruning + Training	32
FC	Fully-Connected	16
FLOPs	Floating-Point Operations	2
FP	Fixed-Point	113
FPGA	Field Programmable Gate Array	51
GFLOPs	Giga Floating-Point Operations	2
GNNs	Graph Neural Networks	17
GPU	Graphics Processing Unit	1

Abbreviation	Description	Page
HCI	Hot Carrier Injection	28
HW	Hardware	31
IoT	Internet of Things	1
KSA	Kogge-Stone Adder	51
LLAAs	Low-Latency Approximate Adders	10
L_{max}	Maximum Latency Constraint	42
LPAAs	Low-Power Approximate Adders	10
LRN	Local Response Normalization	141
MAC	Multiply–Accumulate	9
Max_E	Maximum Error Magnitude	45
MED	Mean Error Distance	45
MFLOPs	Mega Floating-Point Operations	2
MLP	Multi-Layer Perceptron	14
MSE	Mean Square Error	45
MUX	Multiplexer	145
NAS	Neural Architecture Search	20
NBTI	Negative-Bias Temperature Instability	33
NN	Neural Network	13
NPUs	Neural Processing Units/Arrays	11
PDP	Power Delay Product	112
PE	Processing Element	17
PE_I	Input PMF of Error	60
PE_O	PMF of Error at the Output of the Current Stage	60
PIM	Processing In-Memory	179
PM	Pruning Matrix	138
PMF	Probability Mass Function	10
PMOS	P-channel Metal-Oxide Semiconductor	156
PPE	Partial PMF of Error	60
$QuAd_o$	Quality-Area Optimal	10
RCA	Ripple Carry Adder	109
RDD	Read Data Decoder	160
ReLU	Rectified Linear Unit	14
RNNs	Recurrent Neural Networks	17
SCSA	Carry Select Adders	71
SIMD	Single Instruction, Multiple Data	23
SNM	Static Noise Margin	167
SRAM	Static Random Access Memory	7
SSIM	Structural Similarity Index	54
SW	Software	31
$TCost$	Total Cost	140
TDDB	Time-Dependent Dielectric Breakdown	28
TMR	Triple Modular Redundancy	30
TPU	Tensor Processing Unit	17

Abbreviation	Description	Page
TQL	Table of possible Quantization Levels	125
TRBG	True Random Bit Generator	168
TSMC	Taiwan Semiconductor Manufacturing Company	98
VHDL	Very High-Speed Integrated Circuit Hardware Description Language	126
ViT	Vision Transformers	3
V_{TH}	Threshold Voltage	29
WDE	Write Data Encoder	160
WSI	Wafer-Scale Integration	179

Bibliography

[ABF+87] Jacob A Abraham, Prithviraj Banerjee, WK Fuchs, AL Narasimha Reddy, et al. Fault tolerance techniques for systolic arrays. *Computer*, (7):65–75, 1987.

[ACV05] Carlos Alvarez, Jesus Corbal, and Mateo Valero. Fuzzy memoization for floating-point multimedia applications. *IEEE Transactions on Computers*, 54(7):922–927, 2005.

[AGGC18] Arash Azizimazreah, Yongbin Gu, Xiang Gu, and Lizhong Chen. Tolerating soft errors in deep learning accelerators with reliable on-chip memory designs. In *2018 IEEE International Conference on Networking, Architecture and Storage (NAS)*, pages 1–10, 2018.

[AHS17a] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 13(3):1–18, 2017.

[AHS17b] Muhammad Kamran Ayub, Osman Hasan, and Muhammad Shafique. Statistical error analysis for low power approximate adders. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2017.

[AKL16] Haider AF Almurib, T Nandha Kumar, and Fabrizio Lombardi. Inexact designs for approximate low power addition by cell replacement. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 660–665. IEEE, 2016.

[AKS93] Vishwani D Agrawal, Charles R Kime, and Kewal K Saluja. A tutorial on built-in self-test. 2. applications. *IEEE Design & Test of Computers*, 10(2):69–77, 1993.

[AN06] Kanak Agarwal and Sani Nassif. Statistical analysis of sram cell stability. In *Proceedings of the 43rd annual design automation conference*, pages 57–62, 2006.

[AQBAQR17] Mohammed Al-Qizwini, Iman Barjasteh, Hothaifa Al-Qassab, and Hayder Radha. Deep learning algorithm for autonomous driving using googlenet. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 89–96. IEEE, 2017.

[ATBS11] Bartomeu Alorda, Gabriel Torrens, Sebastià Bota, and Jaume Segura. 8t vs. 6t sram cell radiation robustness: A comparative analysis. *Microelectronics reliability*, 51(2):350–359, 2011.

[AVG07] Jaume Abella, Xavier Vera, and Antonio Gonzalez. Penelope: The NBTI-aware processor. In *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*, pages 85–96. IEEE, 2007.

[ban] bankmycell. How many smartphones are in the world? `https://www.bankmycell.com/blog/how-many-phones-are-in-the-world`. Accessed: 2022-10-28.

[Bau05] Robert C Baumann. Radiation-induced soft errors in advanced semiconductor technologies. *IEEE Transactions on Device and materials reliability*, 5(3):305–316, 2005.

[BMO+21] Hadjer Benmeziane, Kaoutar El Maghraoui, Hamza Ouarnoughi, Smail Niar, Martin Wistuba, and Naigang Wang. A comprehensive survey on hardware-aware neural architecture search. *arXiv preprint arXiv:2101.09336*, 2021.

[BOEMN23] Hadjer Benmeziane, Hamza Ouarnoughi, Kaoutar El Maghraoui, and Smail Niar. Multi-objective hardware-aware neural architecture search with pareto rank-preserving surrogate models. *ACM Transactions on Architecture and Code Optimization*, 20(2):1–21, 2023.

[BOO+23] Halima Bouzidi, Mohanad Odema, Hamza Ouarnoughi, Mohammad Abdullah Al Faruque, and Smail Niar. Hadas: Hardware-aware dynamic neural architecture search for edge performance scaling. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE, 2023.

[BW73a] Charles R Baugh and Bruce A Wooley. A two's complement parallel array multiplication algorithm. *IEEE Transactions on computers*, 100(12):1045–1047, 1973.

[BW73b] Charles R Baugh and Bruce A Wooley. A two's complement parallel array multiplication algorithm. *IEEE Transactions on Computers*, C-22(12):1045–1047, Dec 1973.

[Cea19] Z. Chen et al. Binfi: an efficient fault injector for safety-critical machine learning systems. In *ACM HPCA*, page 69, 2019.

[Cea20] Z. Chen et al. Ranger: Boosting error resilience of deep neural networks through range restriction. *arXiv preprint arXiv:2003.13874*, 2020.

[CHS+16] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[CKES17] 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, Jan 2017.

[CLL+14] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, et al. Dadiannao: A machine-learning supercomputer. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 609–622. IEEE, 2014.

[CLMP11] Andrea Calimera, Mirko Loghi, Enrico Macii, and Massimo Poncino. Partitioned cache architectures for reduced nbti-induced aging. In *2011 Design, Automation & Test in Europe*, pages 1–6, 2011.

[Con03] Cristian Constantinescu. Trends and challenges in vlsi circuit reliability. *IEEE micro*, 23(4):14–19, 2003.

[CW90] L-C Chu and Benjamin W Wah. Fault tolerant neural networks with hybrid redundancy. In *1990 IJCNN international joint conference on neural networks*, pages 639–649. IEEE, 1990.

[CYES19] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze. Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(2):292–308, 2019.

[DBK+20] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[DDS+09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, June 2009.

[DFC+15] 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. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, pages 92–104, 2015.

[DFD+15] Jiacnao Deng, Yuntan Fang, Zidong Du, Ymg Wang, Huawei Li, Olivier Temam, Paolo Ienne, David Novo, Xiaowei Li, Yunji Chen, et al. Retraining-based timing error mitigation for hardware neural networks. In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 593–596. IEEE, 2015.

[DJS+18] Alberto Delmas, Patrick Judd, Dylan Malone Stuart, Zissis Poulos, Mostafa Mahmoud, Sayeh Sharify, Milos Nikolic, and Andreas Moshovos. Bit-tactical: Exploiting ineffectual computations in convolutional neural networks: Which, why, and how. *preprint arXiv:1803.03688*, 2018.

[DLLT21] Zihang Dai, Hanxiao Liu, Quoc V Le, and Mingxing Tan. Coatnet: Marrying convolution and attention for all data sizes. *Advances in Neural Information Processing Systems*, 34:3965–3977, 2021.

[EAKHAK94] MO Esonu, AJ Al-Khalili, S Hariri, and D Al-Khalili. Fault-tolerant design methodology for systolic array architectures. *IEE Proceedings-Computers and Digital Techniques*, 141(1):17–28, 1994.

[EES+20] Sara Elkerdawy, Mostafa Elhoushi, Abhineet Singh, Hong Zhang, and Nilanjan Ray. To filter prune, or to layer prune, that is the question. In *Proceedings of the Asian Conference on Computer Vision*, 2020.

[EET] EETimes. Acceleration case: Jury finds toyota liable. `https://www.eetimes.com/acceleration-case-jury-finds-toyota-liable/`. Accessed: 2022-10-28.

[EMH19] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019.

[ERR+19] Andre Esteva, Alexandre Robicquet, Bharath Ramsundar, Volodymyr Kuleshov, Mark DePristo, Katherine Chou, Claire Cui, Greg Corrado, Sebastian Thrun, and Jeff Dean. A guide to deep learning in healthcare. *Nature medicine*, 25(1):24, 2019.

[FFR16] Donato O Forlenza, Orazio P Forlenza, and Bryan J Robbins. Logic-built-in-self-test diagnostic method for root cause identification, January 26 2016. US Patent 9,244,757.

[GAGN15] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *International Conference on Machine Learning*, pages 1737–1746, 2015.

[GHK+18] Ghayoor A Gillani, Muhammad Abdullah Hanif, M Krone, Sabih H Gerez, Muhammad Shafique, and André BJ Kokkeler. SquASH: Approximate square-accumulate with self-healing. *IEEE Access*, 6:49112–49128, 2018.

[GHV+19] Ghayoor A Gillani, Muhammad Abdullah Hanif, Bart Verstoep, Sabih H Gerez, Muhammad Shafique, and Andre BJ Kokkeler. Macish: Designing approximate mac accelerators with internal-self-healing. *IEEE Access*, 7:77142–77160, 2019.

[GKD+21] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference. *arXiv preprint arXiv:2103.13630*, 2021.

[GMP+11] Vaibhav Gupta, Debabrata Mohapatra, Sang Phill Park, Anand Raghunathan, and Kaushik Roy. Impact: imprecise adders for low-power approximate computing. In *IEEE/ACM International Symposium on Low Power Electronics and Design*, pages 409–414. IEEE, 2011.

[GMRR12] Vaibhav Gupta, Debabrata Mohapatra, Anand Raghunathan, and Kaushik Roy. Low-power digital signal processing using approximate adders. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(1):124–137, 2012.

[GSK+15] Dennis Gnad, Muhammad Shafique, Florian Kriebel, Semeen Rehman, Duo Sun, and Jörg Henkel. Hayat: Harnessing dark silicon and variability for aging deceleration and balancing. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2015.

[GW08] Rafael C. Gonzalez and Richard E. Woods. *Digital image processing*. pearson education, 2008.

[GXM14] Jing Guo, Liyi Xiao, and Zhigang Mao. Novel low-power and highly reliable radiation hardened memory cell for 65 nm cmos technology. *IEEE TCAS-I*, 61(7):1994–2001, 2014.

[HBD+13] Jörg Henkel, Lars Bauer, Nikil Dutt, Puneet Gupta, Sani Nassif, Muhammad Shafique, Mehdi Tahoori, and Norbert Wehn. Reliable on-chip systems in the nano-era: Lessons learnt and future trends. In *Proceedings of the 50th Annual Design Automation Conference*, page 99. ACM, 2013.

[HCS+16] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. *Advances in neural information processing systems*, 29, 2016.

[HEAK13] Jörg Henkel, Thomas Ebi, Hussam Amrouch, and Heba Khdr. Thermal management for dependable on-chip systems. In *2013 18th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 113–118. IEEE, 2013.

[HHHS17] Muhammad Abdullah Hanif, Rehan Hafiz, Osman Hasan, and Muhammad Shafique. QuAd: Design and analysis of quality-area optimal low-latency

approximate adders. In *Design Automation Conference (**DAC**)*, pages 1–6, 2017.

[HHHS20] Muhammad Abdullah Hanif, Rehan Hafiz, Osman Hasan, and Muhammad Shafique. PEMACx: A probabilistic error analysis methodology for adders with cascaded approximate units. In *Design Automation Conference (**DAC**)*, pages 1–6, 2020.

[HHS18] Muhammad Abdullah Hanif, Rehan Hafiz, and Muhammad Shafique. Error resilience analysis for systematically employing approximate computing in convolutional neural networks. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 913–916. IEEE, 2018.

[HHS20a] Muhammad Abdullah Hanif, Le-Ha Hoang, and Muhammad Shafique. Cross-layer approaches for improving the dependability of deep learning systems. In *Proceedings of the 23th International Workshop on Software and Compilers for Embedded Systems*, pages 78–81, 2020.

[HHS20b] 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. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2020.

[HKP+18] Muhammad Abdullah Hanif, Faiq Khalid, Rachmad Vidya Wicaksana Putra, Semeen Rehman, and Muhammad Shafique. Robust machine learning systems: Reliability and security for deep neural networks. In *2018 IEEE 24th International Symposium on On-Line Testing And Robust System Design (IOLTS)*, pages 257–260. IEEE, 2018.

[HKS19] Muhammad Abdullah Hanif, Faiq Khalid, and Muhammad Shafique. CANN: Curable approximations for high-performance deep neural network accelerators. In *Design Automation Conference (**DAC**)*, pages 1–6. IEEE, 2019.

[HLM+16] 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. *ACM SIGARCH Computer Architecture News*, 44(3):243–254, 2016.

[HMA+18] M. A. Hanif, A. Marchisio, T. Arif, R. Hafiz, S. Rehman, and M. Shafique. X-dnns: Systematic cross-layer approximations for energy-efficient deep neural networks. *Journal of Low Power Electronics*, 14(4):520–534, 2018.

[HMD15] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[HPN+16] Song Han, Jeff Pool, Sharan Narang, Huizi Mao, Enhao Gong, Shijian Tang, Erich Elsen, Peter Vajda, Manohar Paluri, John Tran, et al. Dsd: Dense-sparse-dense training for deep neural networks. *arXiv preprint arXiv:1607.04381*, 2016.

[HPTD15] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.

[Hru] Joel Hruska. Some amd rx 460s can be modded to unlock missing cores, additional performance. Online; accessed 20-October-2019.

[HS92] Babak Hassibi and David Stork. Second order derivatives for network pruning: Optimal brain surgeon. *Advances in neural information processing systems*, 5, 1992.

[HS20a] Muhammad Abdullah Hanif and Muhammad Shafique. Dependable deep learning: Towards cost-efficient resilience of deep neural network accelerators against soft errors and permanent faults. In *2020 IEEE 26th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 1–4. IEEE, 2020.

[HS20b] Muhammad Abdullah Hanif and Muhammad Shafique. SalvageDNN: Salvaging deep neural network accelerators with permanent faults through saliency-driven fault-aware mapping. *Philosophical Transactions of the Royal Society A (**RSTA**)*, 378(2164):20190164, 2020.

[HS21] 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 *2021 Design, Automation & Test in Europe Conference & Exhibition (**DATE**)*, pages 729–734. IEEE, 2021.

[HS22] Muhammad Abdullah Hanif and Muhammad Shafique. A cross-layer approach towards developing efficient embedded deep learning systems. *Microprocessors and Microsystems*, page 103609, 2022.

[HS23] Muhammad Abdullah Hanif and Muhammad Shafique. Reduce: A framework for reducing the overheads of fault-aware retraining. *arXiv preprint arXiv:2305.12595*, 2023.

[HSM+22] Muhammad Abdullah Hanif, Giuseppe Maria Sarda, Alberto Marchisio, Guido Masera, Maurizio Martina, and Muhammad Shafique. CoNLoCNN: Exploiting correlation and non-uniform quantization for energy-efficient low-precision deep convolutional neural networks. In *2022 International Joint Conference on Neural Networks (**IJCNN**)*, pages 1–8. IEEE, 2022.

[HSTK21] 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, 2021.

[HWT+15] Brody Huval, Tao Wang, Sameep Tandon, Jeff Kiske, Will Song, Joel Pazhayampallil, Mykhaylo Andriluka, Pranav Rajpurkar, Toki Migimatsu, Royce Cheng-Yue, et al. An empirical evaluation of deep learning on highway driving. *arXiv preprint arXiv:1504.01716*, 2015.

[Inc] Google Inc. Tensorflow lite. https://www.tensorflow.org/mobile/tflite/.

[ISO11] Road vehicles – Functional Safety. Standard, International Organization for Standardization, Geneva, CH, 2011.

[JKC+18] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2704–2713, 2018.

[JVS+18] Shubham Jain, Swagath Venkataramani, Vijayalakshmi Srinivasan, Jungwook Choi, Pierce Chuang, and Leland Chang. Compensated-dnn: energy efficient low-precision deep neural networks by compensating quantization errors. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pages 1–6, 2018.

[JVS+19] Shubham Jain, Swagath Venkataramani, Vijayalakshmi Srinivasan, Jungwook Choi, Kailash Gopalakrishnan, and Leland Chang. Biscaled-dnn: Quantizing long-tailed datastructures with two scale factors for deep neural networks. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2019.

[JW12] Tao Jin and Shuai Wang. Aging-aware instruction cache design by duty cycle balancing. In *2012 IEEE Computer Society Annual Symposium on VLSI*, pages 195–200. IEEE, 2012.

[JYP+17] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pages 1–12. IEEE, 2017.

[KCR11] Saurabh Kothawade, Koushik Chakraborty, and Sanghamitra Roy. Analysis and mitigation of nbti aging in register file: An end-to-end approach. In *2011 12th International Symposium on Quality Electronic Design*, pages 1–7, 2011.

[KGE11] Parag Kulkarni, Puneet Gupta, and Milos Ercegovac. Trading accuracy for power with an underdesigned multiplier architecture. In *2011 24th Internatioal Conference on VLSI Design*, pages 346–351. IEEE, 2011.

[KGPK08] Kunhyuk Kang, Saakshi Gangwal, Sang Phill Park, and Roy Kaushik. NBTI induced performance degradation in logic and memory circuits: How effectively can we approach a reliability solution? In *2008 Asia and South Pacific Design Automation Conference*, pages 726–731. IEEE, 2008.

[KH+09] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[KHM+18a] Sung Kim, Patrick Howe, Thierry Moreau, Armin Alaghi, Luis Ceze, and Visvesh Sathe. Matic: Learning around errors for efficient low-voltage neural network accelerators. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE, 2018.

[KHM+18b] Sung Kim, Patrick Howe, Thierry Moreau, Armin Alaghi, Luis Ceze, and Visvesh S Sathe. Energy-efficient neural network acceleration in the presence of bit-level memory errors. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 65(12):4285–4298, 2018.

[KK98] Israel Koren and Zahava Koren. Defect tolerance in vlsi circuits: techniques and yield analysis. *Proceedings of the IEEE*, 86(9):1819–1838, 1998.

[KK12] Andrew B Kahng and Seokhyeong Kang. Accuracy-configurable adder for approximate arithmetic designs. In *Proc. 49th Annual Des. Autom. Conf.*, pages 820–825, 2012.

[KKS09] Sanjay V Kumar, Chris H Kim, and Sachin S Sapatnekar. Adaptive techniques for overcoming performance degradation due to aging in digital circuits. In *2009 Asia and South Pacific Design Automation Conference*, pages 284–289. IEEE, 2009.

[KKS15] Georgios Keramidas, Chrysa Kokkala, and Iakovos Stamoulis. Clumsy value cache: An approximate memoization technique for mobile gpu fragment shaders. In *Workshop on approximate computing (WAPCO'15)*, page 2, 2015.

[KL83] HT Kung and Monica S Lam. Fault-tolerance and two-level pipelining in vlsi systolic arrays. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE, 1983.

[KP86] Israel Koren and Dhiraj K Pradhan. Yield and performance enhancement through redundancy in vlsi and wsi multiprocessor systems. *Proceedings of the IEEE*, 74(5):699–711, 1986.

[KR89] Jung Hwan Kim and Sudhakar M. Reddy. On the design of fault-tolerant two-dimensional systolic arrays for yield enhancement. *IEEE Transactions on Computers*, 38(4):515–525, 1989.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[KSK18] Hyoukjun Kwon, Ananda Samajdar, and Tushar Krishna. Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 461–475. ACM, 2018.

[LAT18] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*, 2018.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[LCX+21] Cheng Liu, Cheng Chu, Dawen Xu, Ying Wang, Qianlong Wang, Huawei Li, Xiaowei Li, and Kwang-Ting Cheng. Hyca: A hybrid computing architecture for fault tolerant deep learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2021.

[LDS89] Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989.

[Leo] George Leopold. Boeing crashes highlight a worsening reliability crisis. `https://www.eetimes.eu/boeing-crashes-highlight-a-worsening-reliability-crisis/`. Accessed: 2022-10-28.

[LGLG08] Alberto Leon-Garcia and Alberto. Leon-Garcia. *Probability, statistics, and random processes for electrical engineering*. Pearson/Prentice Hall 3rd ed. Upper Saddle River, NJ, 2008.

[LHL13] Jinghang Liang, Jie Han, and Fabrizio Lombardi. New metrics for the reliability of approximate and probabilistic adders. *IEEE Transactions on Computers*, 62(9):1760–1771, 2013.

[LHL15] Cong Liu, Jie Han, and Fabrizio Lombardi. An analytical framework for evaluating the error characteristics of approximate adders. *IEEE Transactions on Computers*, 64(5):1268–1281, 2015.

[LHS+17] 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*, pages 1–12, 2017.

[LJL89] Hon F. Li, R Jayakumar, and C Lam. Restructuring for fault-tolerant systolic arrays. *IEEE transactions on computers*, 38(2):307–311, 1989.

[LKB+17] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen Awm Van Der Laak, Bram Van Ginneken, and Clara I Sánchez. A survey on deep learning in medical image analysis. *Medical image analysis*, 42:60–88, 2017.

[LKD+16] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[LM76] Len Levine and Ware Meyers. Special feature: Semiconductor memory reliability with error detecting and correcting codes. *Computer*, 9(10):43–50, 1976.

[LMB+14] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[LPMZ11] Song Liu, Karthik Pattabiraman, Thomas Moscibroda, and Benjamin G Zorn. Flikker: Saving dram refresh-power through critical data partitioning. In *Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems*, pages 213–224, 2011.

[LTA16] Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. Fixed point quantization of deep convolutional networks. In *International conference on machine learning*, pages 2849–2858, 2016.

[LV62] Robert E Lyons and Wouter Vanderkulk. The use of triple-modular redundancy to improve computer reliability. *IBM journal of research and development*, 6(2):200–209, 1962.

[LYL+17] Wenyan Lu, Guihai Yan, Jiajun Li, Shijun Gong, Yinhe Han, and Xiaowei Li. Flexflow: A flexible dataflow accelerator architecture for convolutional neural networks. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 553–564. IEEE, 2017.

[LYL+18] Jiajun Li, Guihai Yan, Wenyan Lu, Shuhao Jiang, Shijun Gong, Jingya Wu, and Xiaowei Li. Smartshuttle: Optimizing off-chip memory accesses for deep learning accelerators. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 343–348. IEEE, 2018.

[MAFL10] Hamid Reza Mahdiani, Ali Ahmadi, Sied Mehdi Fakhraie, and Caro Lucas. Bio-inspired imprecise computational blocks for efficient vlsi implementation of soft-computing applications. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 57(4):850–862, 2010.

[MAFSG18] Mehdi Mohammadi, Ala Al-Fuqaha, Sameh Sorour, and Mohsen Guizani. Deep learning for iot big data and streaming analytics: A survey. *IEEE Communications Surveys & Tutorials*, 20(4):2923–2960, 2018.

[MAJD+20] Sajjad Mozaffari, Omar Y Al-Jarrah, Mehrdad Dianati, Paul Jennings, and Alexandros Mouzakitis. Deep learning-based vehicle behavior prediction for autonomous driving applications: A review. *IEEE Transactions on Intelligent Transportation Systems*, 23(1):33–47, 2020.

[McL19] Paul McLellan. Hot chips: The biggest chip in the world. `https://community.cadence.com/cadence_blogs_8/b/breakfast-bytes/posts/the-biggest-chip-in-the-world`, 2019. Accessed: 2019-09-10.

[ME19] Marian Mazzone and Ahmed Elgammal. Art, creativity, and the potential of artificial intelligence. In *Arts*, volume 8, page 26. MDPI, 2019.

[MHH+16] Sana Mazahir, Osman Hasan, Rehan Hafiz, Muhammad Shafique, and Jörg Henkel. An area-efficient consolidated configurable error correction for approximate hardware accelerators. In *Proceedings of the 53rd Annual Design Automation Conference*, pages 1–6, 2016.

[MHH+17] Sana Mazahir, Osman Hasan, Rehan Hafiz, Muhammad Shafique, and Jörg Henkel. Probabilistic error modeling for approximate adders. *IEEE Transactions on Computers*, 66(3):515–530, 2017.

[MHS19] Sana Mazahir, Osman Hasan, and Muhammad Shafique. Self-compensating accelerators for efficient approximate computing. *Microelectronics Journal*, 88:9–17, 2019.

[Mit16] Sparsh Mittal. A survey of techniques for approximate computing. *ACM Comput. Surv.*, 48(4):62:1–62:33, March 2016.

[mmc] `https://mmclassification.readthedocs.io/en/latest/model_zoo.html`. Accessed: 2022-10-28.

[MPC16] Homay Danaei Mehr, Huseyin Polat, and Aydin Cetin. Resident activity recognition in smart homes by using artificial neural networks. In *2016 4th international istanbul smart grid congress and fair (ICSG)*, pages 1–5. IEEE, 2016.

[MSZ+11] Evelyn Mintarno, Joëlle Skaf, Rui Zheng, Jyothi Bhaskar Velamala, Yu Cao, Stephen Boyd, Robert W Dutton, and Subhasish Mitra. Self-tuning for maximized lifetime energy-efficiency in the presence of circuit aging. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30(5):760–773, 2011.

[MVS+19] Vojtech Mrazek, Zdenek Vasícek, Lukás Sekanina, Muhammad Abdullah Hanif, and Muhammad Shafique. Alwann: automatic layer-wise approximation of deep neural network accelerators without retraining. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2019.

[MW07] Yury Markovsky and John Wawrzynek. On the opportunity to improve system yield with multi-core architectures. In *Proc. of the IEEE Workshop on Design Manufacturability & Yield*, 2007.

[MWW+18] Riccardo Miotto, Fei Wang, Shuang Wang, Xiaoqian Jiang, and Joel T Dudley. Deep learning for healthcare: review, opportunities and challenges. *Briefings in bioinformatics*, 19(6):1236–1246, 2018.

[NANK19] Mohamed A Neggaz, Ihsen Alouani, Smail Niar, and Fadi Kurdahi. Are cnns reliable enough for critical applications? an exploratory study. *IEEE Design & Test*, 37(2):76–83, 2019.

[NHK+99] Shigeru Nakahara, Keiichi Higeta, Masaki Kohno, Toshiaki Kawamura, and Keizo Kakitani. Built-in self-test for ghz embedded srams using flexible pattern generator and new repair algorithm. In *International Test Conference 1999. Proceedings (IEEE Cat. No. 99CH37034)*, pages 301–310. IEEE, 1999.

[NML+20] Wei Niu, Xiaolong Ma, Sheng Lin, Shihao Wang, Xuehai Qian, Xue Lin, Yanzhi Wang, and Bin Ren. Patdnn: Achieving real-time dnn execution on mobile devices with pattern-based weight pruning. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 907–922, 2020.

[OSM02] Nahmsuk Oh, Philip P Shirvani, and Edward J McCluskey. Error detection by duplicated instructions in super-scalar processors. *IEEE Transactions on Reliability*, 51(1):63–75, 2002.

[pap] `https://paperswithcode.com/sota/image-classification-on-imagenet`. Accessed: 2022-10-28.

[PBCR19] Pramesh Pandey, Prabal Basu, Koushik Chakraborty, and Sanghamitra Roy. Greentpu: Improving timing error resilience of a near-threshold tensor processing unit. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2019.

[PBGM09] Michael D Powell, Arijit Biswas, Shantanu Gupta, and Shubhendu S Mukherjee. Architectural core salvaging in a multi-core processor for hard-error tolerance. In *ACM SIGARCH Computer Architecture News*, volume 37, pages 93–104. ACM, 2009.

[PRM+17] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W Keckler, and William J Dally. Scnn: An accelerator for compressed-sparse convolutional neural networks. *ACM SIGARCH Computer Architecture News*, 45(2):27–40, 2017.

[pro] Product binning. Online; accessed 20-October-2019 [*Available at* `https://en.wikipedia.org/wiki/Product_binning`].

[RBKS17] Rengarajan Ragavan, Benjamin Barrois, Cedric Killian, and Olivier Sentieys. Pushing the limits of voltage over-scaling for error-resilient applications. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 476–481. IEEE, 2017.

[REHS+16] Semeen Rehman, Walaa El-Harouni, Muhammad Shafique, Akash Kumar, Jorg Henkel, and Jörg Henkel. Architectural-space exploration of approximate multipliers. In *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2016.

[RHK+20] Mohsin Riaz, Rehan Hafiz, Salman Abdul Khaliq, Muhammad Faisal, Hafiz Talha Iqbal, Mohsen Ali, and Muhammad Shafique. CAxCNN: Towards the use of canonic sign digit based approximation for hardware-friendly convolutional neural networks. *IEEE Access*, 8:127014–127021, 2020.

[RORF16] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, pages 525–542. Springer, 2016.

[RPP21] E Ramanujam, Thinagaran Perumal, and S Padmavathi. Human activity recognition with smartphone and wearable sensors using deep learning techniques: A review. *IEEE Sensors Journal*, 21(12):13029–13040, 2021.

[RSR+10] A Ricketts, Jawar Singh, Krishnan Ramakrishnan, Narayanan Vijaykrishnan, and Dhiraj K Pradhan. Investigating the impact of nbti on different power saving cache strategies. In *2010 Design, Automation & Test in*

Europe Conference & Exhibition (DATE 2010), pages 592–597. IEEE, 2010.

[RTGM13] Bharathwaj Raghunathan, Yatish Turakhia, Siddharth Garg, and Diana Marculescu. Cherry-picking: exploiting process variations in dark-silicon homogeneous chip multi-processors. In *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 39–44. IEEE, 2013.

[RVF+15] Ashish Ranjan, Swagath Venkataramani, Xuanyao Fong, Kaushik Roy, and Anand Raghunathan. Approximate storage for energy efficient spintronic memories. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2015.

[RWA+16] 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. IEEE, 2016.

[SAHH15] Muhammad Shafique, Waqas Ahmad, Rehan Hafiz, and Jorg Henkel. A low latency generic accuracy configurable adder. In *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*, pages 1–6. IEEE, 2015.

[Sar21] Iqbal H Sarker. Deep learning: a comprehensive overview on techniques, taxonomy, applications and research directions. *SN Computer Science*, 2(6):1–20, 2021.

[SBS+18] Niko Sünderhauf, Oliver Brock, Walter Scheirer, Raia Hadsell, Dieter Fox, Jürgen Leitner, Ben Upcroft, Pieter Abbeel, Wolfram Burgard, Michael Milford, et al. The limits and potentials of deep learning for robotics. *The International journal of robotics research*, 37(4-5):405–420, 2018.

[SCYE17] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017.

[SG11] Taniya Siddiqua and Sudhanva Gurumurthi. Enhancing nbti recovery in sram arrays through recovery boosting. *IEEE transactions on very large scale integration (VLSI) systems*, 20(4):616–629, 2011.

[Sha] Stephen Shankland. Meet Tesla's self-driving car computer and its two AI brains. `https://www.cnet.com/news/meet-tesla-self-driving-car-computer-and-its-two-ai-brains/`.

[SHR+16] Muhammad Shafique, Rehan Hafiz, Semeen Rehman, Walaa El-Harouni, and Jörg Henkel. Cross-layer approximate computing: From logic to

architectures. In *Proceedings of the 53rd Annual Design Automation Conference*, pages 1–6, 2016.

[SJLM14] Mehrzad Samadi, Davoud Anoushe Jamshidi, Janghaeng Lee, and Scott Mahlke. Paraprox: Pattern-based approximation for data parallel applications. In *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems*, pages 35–50, 2014.

[SKH16] Muhammad Shafique, Muhammad Usman Karim Khan, and Jörg Henkel. Content-aware low-power configurable aging mitigation for sram memories. *IEEE Transactions on Computers*, 65(12):3617–3630, 2016.

[SKMB03] Premkishore Shivakumar, Stephen W Keckler, Charles R Moore, and Doug Burger. Exploiting microarchitectural redundancy for defect tolerance. In *Proceedings 21st International Conference on Computer Design*, pages 481–488. IEEE, 2003.

[SKTH15] Muhammad Shafique, Muhammad Usman Karim Khan, Orcun Tüfek, and Jörg Henkel. EnAAM: Energy-efficient anti-aging for on-chip video memories. In *Proceedings of the 52nd Annual Design Automation Conference*, pages 1–6, 2015.

[SLJ+13] Mehrzad Samadi, Janghaeng Lee, D Anoushe Jamshidi, Amir Hormati, and Scott Mahlke. Sage: Self-tuning approximation for graphics engines. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 13–24, 2013.

[SMBJ14] Joshua San Miguel, Mario Badr, and Natalie Enright Jerger. Load value approximation. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 127–139. IEEE, 2014.

[SNT+20] Muhammad Shafique, Mahum Naseer, Theocharis Theocharides, Christos Kyrkou, Onur Mutlu, Lois Orosa, and Jungwook Choi. Robust machine learning systems: Challenges, current trends, perspectives, and the road ahead. *IEEE Design & Test*, 37(2):30–57, 2020.

[SPS+18] 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 *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 764–775. IEEE, 2018.

[SS15] Deepashree Sengupta and Sachin S Sapatnekar. Femto: Fast error analysis in multipliers through topological traversal. In *Computer-Aided Design (ICCAD), 2015 IEEE/ACM International Conference on*, pages 294–299. IEEE, 2015.

[SSMJ15] Mark Sutherland, Joshua San Miguel, and Natalie Enright Jerger. Texture cache approximation on gpus. In *Workshop on approximate computing across the stack*, page 2, 2015.

[SSSG17] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE international conference on computer vision*, pages 843–852, 2017.

[SUK18] Behzad Salami, Osman S Unsal, and Adrian Cristal Kestelman. On the resilience of RTL NN accelerators: Fault characterization and mitigation. In *2018 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 322–329. IEEE, 2018.

[SV05] Ethan Schuchman and TN Vijaykumar. Rescue: A microarchitecture for testability and defect tolerance. In *32nd International Symposium on Computer Architecture (ISCA'05)*, pages 160–171. IEEE, 2005.

[SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[SZBP08] Jeonghee Shin, Victor Zyuban, Pradip Bose, and Timothy M Pinkston. A proactive wearout recovery approach for exploiting microarchitectural redundancy to extend cache sram lifetime. volume 36, pages 353–362, 2008.

[TF17] Itsuo Takanami and Masaru Fukushi. A built-in circuit for self-repairing mesh-connected processor arrays with spares on diagonal. In *2017 IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC)*, pages 110–117. IEEE, 2017.

[TH12] Itsuo Takanami and Tadayoshi Horita. A built-in circuit for self-repairing mesh-connected processor arrays by direct spare replacement. In *2012 IEEE 18th Pacific Rim International Symposium on Dependable Computing*, pages 96–104. IEEE, 2012.

[TM18] Frederick Tung and Greg Mori. Clip-q: Deep network compression learning by in-parallel pruning-quantization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7873–7882, 2018.

[TYRW14] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1701–1708, 2014.

[Vai] Lionel Sujay Vailshery. Number of internet of things (iot) connected devices worldwide from 2019 to 2021, with forecasts from 2022

to 2030. https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/. Accessed: 2022-10-28.

[VBI08] Ajay K Verma, Philip Brisk, and Paolo Ienne. Variable latency speculative addition: A new paradigm for arithmetic circuit design. In *Proceedings of the conference on Design, automation and test in Europe*, pages 1250–1255. ACM, 2008.

[VCC+13] Swagath Venkataramani, Vinay K Chippa, Srimat T Chakradhar, Kaushik Roy, and Anand Raghunathan. Quality programmable vector processors for approximate computing. In *2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–12. IEEE, 2013.

[VCRR15] S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan. Approximate computing and the quest for computing efficiency. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2015.

[VKAK+17] Shaghayegh Vahdat, Mehdi Kamal, Ali Afzali-Kusha, Massoud Pedram, and Zainalabedin Navabi. Truncapp: A truncation-based approximate divider for energy efficient dsp applications. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 1635–1638. IEEE, 2017.

[VL15] A. Vedaldi and K. Lenc. Matconvnet: Convolutional neural networks for matlab. In *Proceedings of the 23rd ACM international conference on Multimedia*, pages 689–692. ACM, 2015.

[VRRR14] Swagath Venkataramani, Ashish Ranjan, Kaushik Roy, and Anand Raghunathan. Axnn: energy-efficient neuromorphic systems using approximate computing. In *2014 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 27–32. IEEE, 2014.

[VZBT10] Ramakrishna Vadlamani, Jia Zhao, Wayne Burleson, and Russell Tessier. Multicore soft error rate stabilization using adaptive dual modular redundancy. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 27–32. European Design and Automation Association, 2010.

[Wit] With cpu chips having billions of transistors, what happens if a few go bad? Online; accessed 20-October-2019.

[WLG+18] Yi Wu, You Li, Xiangxuan Ge, Yuan Gao, and Weikang Qian. An efficient method for calculating the error statistics of block-based approximate adders. *IEEE Transactions on Computers*, 68(1):21–38, 2018.

[WLL+19] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8612–8620, 2019.

[XKM16] Qiang Xu, Nam Sung Kim, and T Mytkowicz. Approximate computing: A survey. *IEEE Des. & Test*, 33(1):8–22, 2016.

[XLHL20] Qizhe Xie, Minh-Thang Luong, Eduard Hovy, and Quoc V Le. Self-training with noisy student improves imagenet classification. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10687–10698, 2020.

[XXL+19] Dawen Xu, Kouzi Xing, Cheng Liu, Ying Wang, Yulin Dai, Long Cheng, Huawei Li, and Lei Zhang. Resilient neural network training for accelerators with computing errors. In *2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, volume 2160, pages 99–102. IEEE, 2019.

[YLP+17] Jiecao Yu, Andrew Lukefahr, David Palframan, Ganesh Dasika, Reetuparna Das, and Scott Mahlke. Scalpel: Customizing dnn pruning to the underlying hardware parallelism. *ACM SIGARCH Computer Architecture News*, 45(2):548–560, 2017.

[Yos] Junko Yoshida. Toyota case: Single bit flip that killed. `https://www.eetimes.com/toyota-case-single-bit-flip-that-killed/`. Accessed: 2022-10-28.

[YPT+16] Amir Yazdanbakhsh, Gennady Pekhimenko, Bradley Thwaites, Hadi Esmaeilzadeh, Onur Mutlu, and Todd C Mowry. Rfvp: Rollback-free value prediction with safe-to-approximate loads. *ACM Transactions on Architecture and Code Optimization (TACO)*, 12(4):1–26, 2016.

[YWY+13] Rong Ye, Ting Wang, Feng Yuan, Rakesh Kumar, and Qiang Xu. On reconfiguration-oriented approximate adder design and its application. In *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 48–54. IEEE, 2013.

[ZAJ+19] Maheen Zahid, Fahad Ahmed, Nadeem Javaid, Raza Abid Abbasi, Hafiza Syeda Zainab Kazmi, Atia Javaid, Muhammad Bilal, Mariam Akbar, and Manzoor Ilahi. Electricity price and load forecasting using enhanced convolutional neural network and enhanced support vector regression in smart grids. *Electronics*, 8(2):122, 2019.

[ZBG19] Jeff Jun Zhang, Kanad Basu, and Siddharth Garg. Fault-tolerant systolic array based accelerators for deep neural network execution. *IEEE Design & Test*, 36(5):44–53, 2019.

[ZDL+20] Kai Zhao, Sheng Di, Sihuan Li, Xin Liang, Yujia Zhai, Jieyang Chen, Kaiming Ouyang, Franck Cappello, and Zizhong Chen. FT-CNN: Algorithm-based fault tolerance for convolutional neural networks. *IEEE Transactions on Parallel and Distributed Systems*, 32(7):1677–1689, 2020.

[ZDZ+16] Shijin Zhang, Zidong Du, Lei Zhang, Huiying Lan, Shaoli Liu, Ling Li, Qi Guo, Tianshi Chen, and Yunji Chen. Cambricon-x: An accelerator for sparse neural networks. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–12. IEEE, 2016.

[ZGBG18] Jeff Jun Zhang, Tianyu Gu, Kanad Basu, and Siddharth Garg. Analyzing and mitigating the impact of permanent faults on a systolic array based neural network accelerator. In *2018 IEEE 36th VLSI Test Symposium (VTS)*, pages 1–6. IEEE, 2018.

[ZGY09] Ning Zhu, Wang Ling Goh, and Kiat Seng Yeo. An enhanced low-power high-speed adder for error-tolerant application. In *Proc. 12th Int. Symp. Integ. Circuits,*, pages 69–72, 2009.

[ZHXL08] Lei Zhang, Yinhe Han, Qiang Xu, and Xiaowei Li. Defect tolerance in homogeneous manycore processors using core-level redundancy with unified topology. In *Proceedings of the conference on Design, automation and test in Europe*, pages 891–896. ACM, 2008.

[ZJZ+18] Neta Zmora, Guy Jacob, Lev Zlotnik, Bar Elharar, and Gal Novik. Neural network distiller, June 2018.

[ZKKK19] Fotios Zantalis, Grigorios Koulouras, Sotiris Karabetsos, and Dionisis Kandris. A review of machine learning and iot in smart transportation. *Future Internet*, 11(4):94, 2019.

[ZRGG18] Jeff Zhang, Kartheek Rangineni, Zahra Ghodsi, and Siddharth Garg. Thundervolt: enabling aggressive voltage underscaling and timing error resilience for energy efficient deep learning accelerators. In *Proceedings of the 55th Annual Design Automation Conference*, pages 1–6, 2018.

[ZSBH11] Bruno Zatt, Muhammad Shafique, Sergio Bampi, and Jörg Henkel. A low-power memory architecture with application-aware power management for motion & disparity estimation in multiview video coding. In *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 40–47. IEEE, 2011.

[ZWN+16] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.